

See the Visual

Python

Data Visualization
and Manipulation

100 Exercises

Index

Chapter 1 Introduction

1. Purpose

2. About the Execution Environment for Source Code

Chapter 2 For beginners

1. Histogram Generation from Random Data

2. Height vs Weight Scatter Plot

3. Plotting a Sine Wave with Matplotlib

4. Creating Box Plots of Exam Scores

5. Heatmap of Correlation Matrix

6. Simple Time Series Data Plotting

7. Quarterly Sales Stacked Bar Chart

8. Visualizing Multiple Functions with Python

9. Bubble Chart of Population vs GDP

10. Create a Pair Plot of Iris Dataset

11. 3D Scatter Plot of Customer Data

12. Creating a Violin Plot for Age Distribution

13. Density Plot of Random Data

14. Creating a Donut Chart for Budget Allocation

15. Creating Polar Plots of Trigonometric Functions for Weather Analysis

16. Creating a Sunburst Chart with Hierarchical Data

17. Waterfall Chart for Financial Data Analysis

18. Funnel Chart of Sales Conversion

19. Candlestick Chart of Stock Prices

20. Creating a Treemap of Product Categories

21. Streamgraph of Web Traffic Data

22. Visualizing Network Connections with Chord Diagrams

23. Create a Sankey Diagram of Energy Flow

Chapter 3 For advanced

- [1. Bubble Map of Sales Data](#)
- [2. Hierarchical Clustering Dendrogram](#)
- [3. Parallel Coordinates Plot for Customer Data Analysis](#)
- [4. Word Cloud Generation from Text Data](#)
- [5. Social Network Graph Visualization](#)
- [6. Visualizing Spatial Data with Voronoi Diagrams](#)
- [7. Creating a Lollipop Chart for Survey Results](#)
- [8. Dot Plot of Categorical Data](#)
- [9. Creating a Dumbbell Plot for Comparative Data Analysis](#)
- [10. Generate a Ridgeline Plot of Distribution Data](#)
- [11. Matrix Plot of Confusion Matrix](#)
- [12. Plotting a Wind Rose Diagram](#)
- [13. Bullet Chart for Performance Targets](#)
- [14. Creating a Horizon Chart with Time Series Data](#)
- [15. Network Flow Diagram for Traffic Data Visualization](#)
- [16. Heatmap of Missing Data Visualization](#)
- [17. Connected Scatter Plot for Sales Trend Analysis](#)
- [18. Nested Pie Chart of Demographic Data](#)
- [19. Creating a Dumbbell Dot Plot for Sales Comparison](#)
- [20. Creating a Circular Packing Plot of Hierarchical Data](#)
- [21. Generating a Beeswarm Plot of Distribution Data](#)
- [22. Joy Plot of Distribution Data](#)
- [23. Heatmap of Correlation Matrix](#)
- [24. Generating Pair Grid Plot for Customer Satisfaction Analysis](#)
- [25. Facet Grid Plot of Categorical Data](#)
- [26. Plotting a Linear Regression](#)
- [27. Creating a Residual Plot for Regression Analysis](#)
- [28. Categorical Plot of Survey Data](#)

- [29. Creating a Strip Plot of Categorical Data](#)
- [30. Swarm Plot of Distribution Data](#)
- [31. Factor Plot of Categorical Data](#)
- [32. Comparative Point Plot Generation](#)
- [33. Creating a Bar Plot with Categorical Data](#)
- [34. Count Plot of Categorical Data](#)
- [35. KDE Plot of Distribution Data](#)
- [36. Violin Plot Creation with Seaborn](#)
- [37. Boxen Plot Visualization of Distribution Data](#)
- [38. Joint Plot of Bivariate Data](#)
- [39. Lmplot Regression Analysis](#)
- [40. Creating a Pair Plot for Customer Data Analysis](#)
- [41. Heatmap of Correlation Matrix](#)
- [42. Scatter Matrix Plot of Multivariate Data](#)
- [43. Parallel Coordinates Plot Creation](#)
- [44. Andrews Curves Plot for Multivariate Data Analysis](#)
- [45. RadViz Plot for Multivariate Data Visualization](#)
- [46. Creating a Lag Plot for Time Series Analysis](#)
- [47. Autocorrelation Plot of Time Series Data](#)
- [48. Bootstrap Plot of Statistical Data](#)
- [49. Creating a Hexbin Plot with Pandas](#)
- [50. Creating a Scatter Plot Matrix for Customer Data Analysis](#)
- [51. Generate a Box Plot Using Pandas](#)
- [52. Violin Plot for Sales Data Analysis](#)
- [53. Plotting a KDE Plot Using Pandas](#)
- [54. Creating a Density Plot with Pandas](#)
- [55. Bar Plot Visualization with Pandas](#)
- [56. Creating an Area Plot Using Pandas](#)
- [57. Scatter Plot Creation with Pandas](#)

- [58. Box Plot Visualization Using Plotly](#)
 - [59. Creating a Violin Plot Using Plotly](#)
 - [60. Creating Interactive Line Plots with Plotly](#)
 - [61. Bar Plot Visualization with Plotly](#)
 - [62. Creating a Pie Chart with Plotly](#)
 - [63. Creating a Treemap with Plotly](#)
 - [64. Plotting a Funnel Chart Using Plotly](#)
 - [65. Creating a Waterfall Chart with Plotly for Financial Analysis](#)
 - [66. Generate a Candlestick Chart Using Plotly](#)
 - [67. Creating a Heatmap with Plotly](#)
 - [68. Plotting a Contour Plot with Plotly](#)
 - [69. Creating 3D Scatter Plots with Plotly](#)
 - [70. 3D Surface Plot with Plotly for Customer Satisfaction Analysis](#)
 - [71. Creating a 3D Line Plot with Plotly](#)
 - [72. 3D Mesh Plot Visualization](#)
 - [73. Creating 3D Volume Plots with Plotly for Data Visualization](#)
 - [74. Creating a 3D Cone Plot with Plotly for Sales Data Visualization](#)
 - [75. Creating a 3D Streamline Plot with Plotly](#)
 - [76. 3D Box Plot Creation Using Plotly](#)
 - [77. Creating a 3D Violin Plot with Plotly](#)
 - [78. 3D Parallel Coordinates Plot](#)
 - [79. 3D Andrews Curves Plot with Plotly](#)
- [Chapter 4 Request for review evaluation](#)
- [Appendix: Execution Environment](#)

Chapter 1 Introduction

1. Purpose

This e-book is designed for readers who already possess a foundational understanding of programming and wish to delve deeper into data manipulation and visualization using Python.

Through 100 meticulously crafted exercises, readers will explore various techniques and tools essential for effective data handling and visualization.

Each exercise is accompanied by source code and detailed explanations of the output, making the learning process straightforward and intuitive.

This book allows readers to expand their knowledge effortlessly during commutes or in short breaks, and running the provided code further deepens comprehension.

Whether you are looking to enhance your data analysis skills or simply seek practical coding exercises, this book offers a comprehensive guide to mastering Python for data manipulation and visualization.

2. About the Execution Environment for Source Code

For information on the execution environment used for the source code in this book, please refer to the appendix at the end of the book.

Chapter 2 For beginners

1. Histogram Generation from Random Data

Importance★★★★☆

Difficulty★★☆☆☆

You are a data analyst working for a marketing company. The company wants to understand the distribution of customer ages in their database.

Your task is to create a histogram that visualizes this distribution using randomly generated age data.

Generate a dataset of 1000 customer ages, assuming the ages range from 18 to 80 years old.

Then, create a histogram to visualize the distribution of these ages.

Make sure to include appropriate labels and a title for the histogram.

The histogram should have 10 bins.

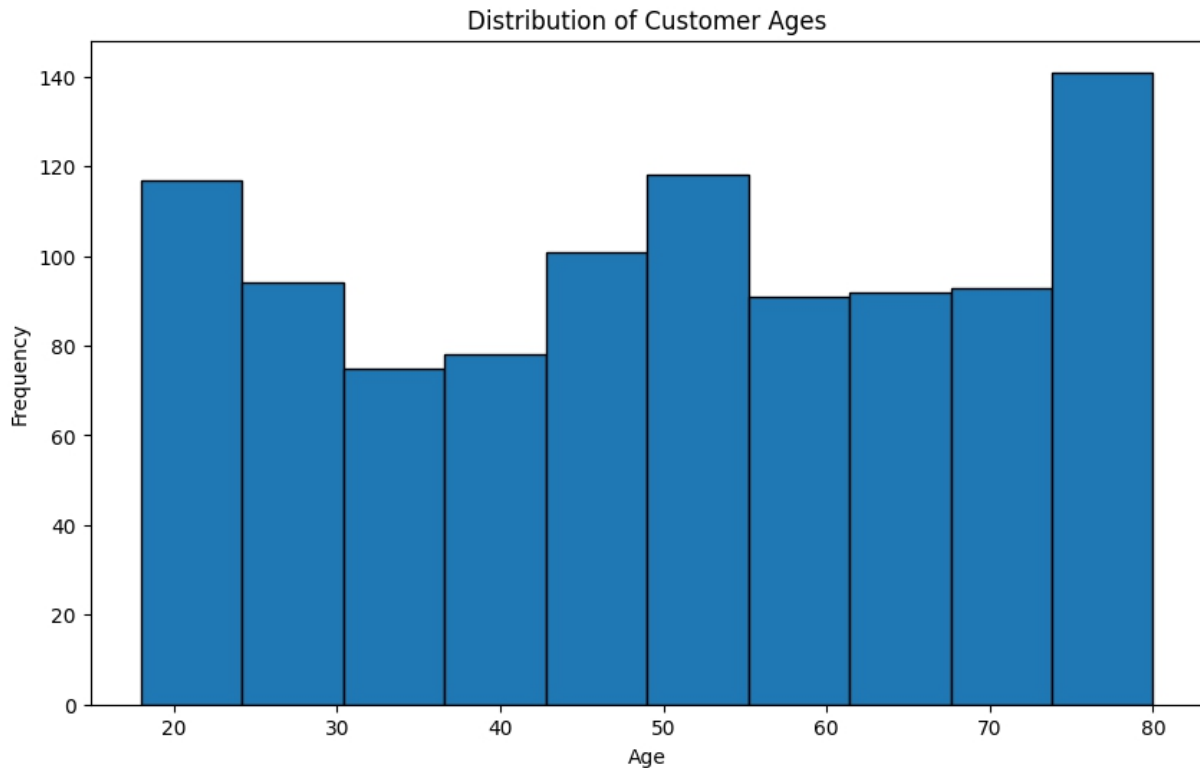
Use matplotlib for visualization.

Generate the sample data within your code.

【Data Generation Code Example】

```
import numpy as np
# Generate random age data
ages = np.random.randint(18, 81, 1000)
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import matplotlib.pyplot as plt
# Generate random age data
ages = np.random.randint(18, 81, 1000)
# Create the histogram
plt.figure(figsize=(10, 6))
plt.hist(ages, bins=10, edgecolor='black')
# Add labels and title
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Distribution of Customer Ages')
# Display the plot
```

```
plt.show()
```

This code demonstrates how to create a histogram using randomly generated data in Python.

Let's break down the process step by step:

Importing necessary libraries:

We import numpy (as np) for generating random data.

We import matplotlib.pyplot (as plt) for creating the visualization.

Generating random data:

We use numpy's random.randint() function to generate 1000 random integers between 18 and 80 (inclusive).

This simulates a dataset of customer ages.

Creating the histogram:

We use plt.figure(figsize=(10, 6)) to set the size of the plot.

plt.hist() is the main function for creating the histogram:

The first argument is our data (ages).

bins=10 specifies that we want 10 bins in our histogram.

edgecolor='black' adds a black outline to each bar for better visibility.

Adding labels and title:

plt.xlabel('Age') adds a label to the x-axis.

plt.ylabel('Frequency') adds a label to the y-axis.

plt.title('Distribution of Customer Ages') adds a title to the plot.

Displaying the plot:

plt.show() is called to display the final histogram.

This code effectively visualizes the distribution of ages in the randomly generated dataset.

The histogram allows us to quickly see which age ranges are most common among the customers.

This type of visualization is crucial in data analysis as it provides an immediate understanding of data distribution, which can inform marketing strategies or other business decisions.

【Trivia】

- ▶ Histograms are excellent tools for visualizing the distribution of continuous data.
- ▶ The number of bins in a histogram can significantly affect its interpretation. Too few bins might obscure important details, while too many can make the overall pattern hard to discern.
- ▶ Matplotlib is just one of many visualization libraries in Python. Others include Seaborn, which is built on top of Matplotlib and provides a higher-level interface, and Plotly, which is great for interactive visualizations.
- ▶ When working with real-world data, it's often necessary to clean and preprocess the data before visualization. This might involve handling missing values, removing outliers, or transforming the data.
- ▶ In addition to histograms, other types of plots like box plots, violin plots, or kernel density estimation plots can also be useful for understanding data distributions.
- ▶ The random seed in numpy can be set for reproducibility of random number generation, which is crucial in scientific computing and data analysis.
- ▶ When dealing with large datasets, consider using libraries like pandas for data manipulation before visualization.
- ▶ In a professional setting, it's often good practice to save your visualizations as image files (e.g., PNG or PDF) for inclusion in reports or presentations.

2. Height vs Weight Scatter Plot

Importance★★★★☆

Difficulty★★☆☆☆

A fitness center wants to analyze the relationship between the height and weight of their clients to provide better personalized training programs.

You are tasked with creating a scatter plot to visualize this relationship.

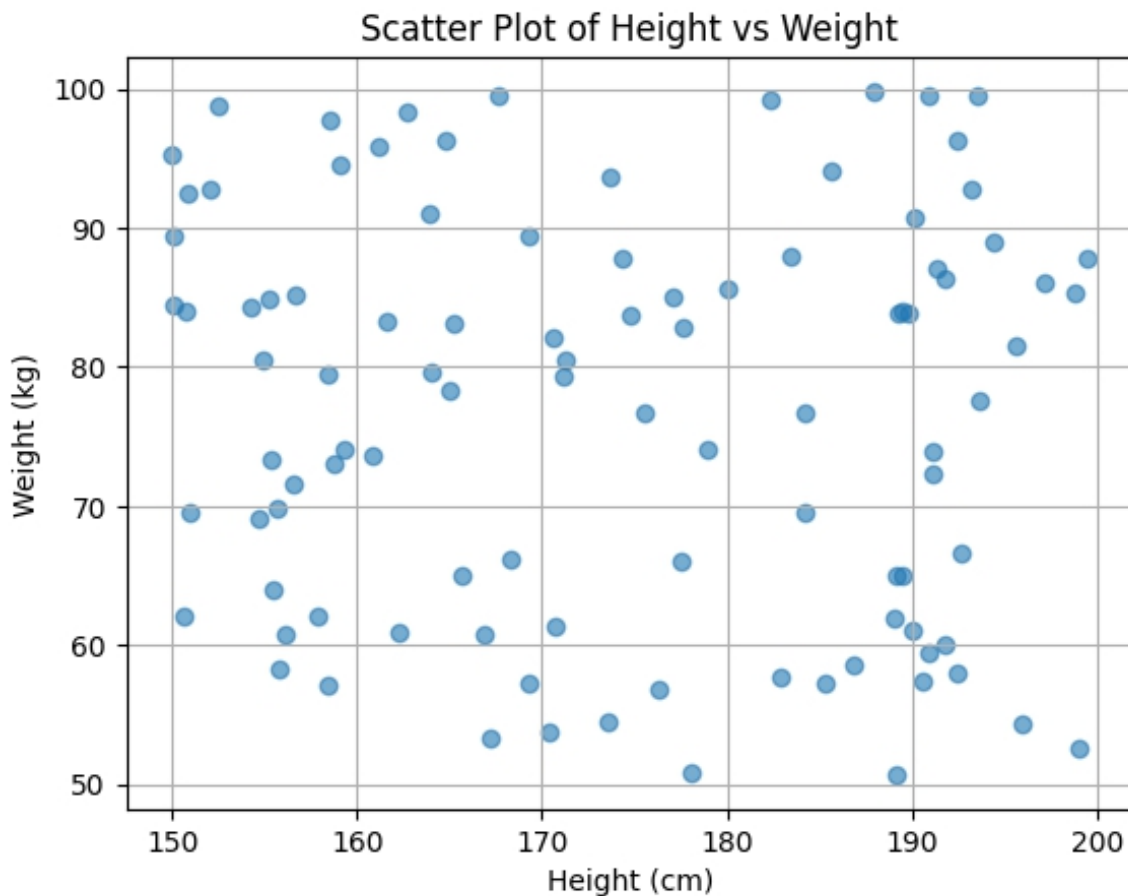
Generate a sample dataset of 100 clients, each with random height (in cm) and weight (in kg).

Use Python to create this scatter plot.

【Data Generation Code Example】

```
import random
import pandas as pd
import numpy as np
np.random.seed(42)
height = [random.uniform(150, 200) for _ in range(100)]
weight = [random.uniform(50, 100) for _ in range(100)]
data = pd.DataFrame({'Height': height, 'Weight': weight})
```


【Diagram Answer】



【Code Answer】

```
import matplotlib.pyplot as plt
import pandas as pd
import random
import numpy as np
np.random.seed(42)
height = [random.uniform(150, 200) for _ in range(100)]
weight = [random.uniform(50, 100) for _ in range(100)]
data = pd.DataFrame({'Height': height, 'Weight': weight})
plt.scatter(data['Height'], data['Weight'], alpha=0.6)
plt.title('Scatter Plot of Height vs Weight')
```

```
plt.xlabel('Height (cm)')  
plt.ylabel('Weight (kg)')  
plt.grid(True)  
plt.show()
```

The code first imports necessary libraries, including matplotlib.pyplot for plotting, pandas for data manipulation, and random and numpy for generating random data. To ensure reproducibility, `np.random.seed(42)` is used. The height and weight lists are generated using list comprehensions, with random values between specified ranges for each of the 100 clients. A pandas DataFrame is then created with this data. The scatter plot is created using `plt.scatter()`, which takes the height and weight data as inputs. The transparency of the points is set using the alpha parameter. The plot is then customized with a title, x-axis label, y-axis label, and grid lines for better readability. Finally, `plt.show()` displays the plot.

【Trivia】

Scatter plots are widely used to visualize relationships between two quantitative variables. They can help identify patterns, trends, and possible correlations within the data. In this case, examining the scatter plot can reveal if there's any correlation between height and weight, which can inform decisions in creating personalized fitness programs.

3. Plotting a Sine Wave with Matplotlib

Importance★★★★☆

Difficulty★★☆☆☆

A local weather station wants to visualize temperature fluctuations throughout the day.

They've asked you to create a sine wave plot that represents the typical daily temperature pattern.

Your task is to generate a sine wave using numpy, and then plot it using matplotlib.

The x-axis should represent 24 hours of the day, and the y-axis should represent temperature in Celsius.

The sine wave should have an amplitude of 5°C and be centered around 20°C.

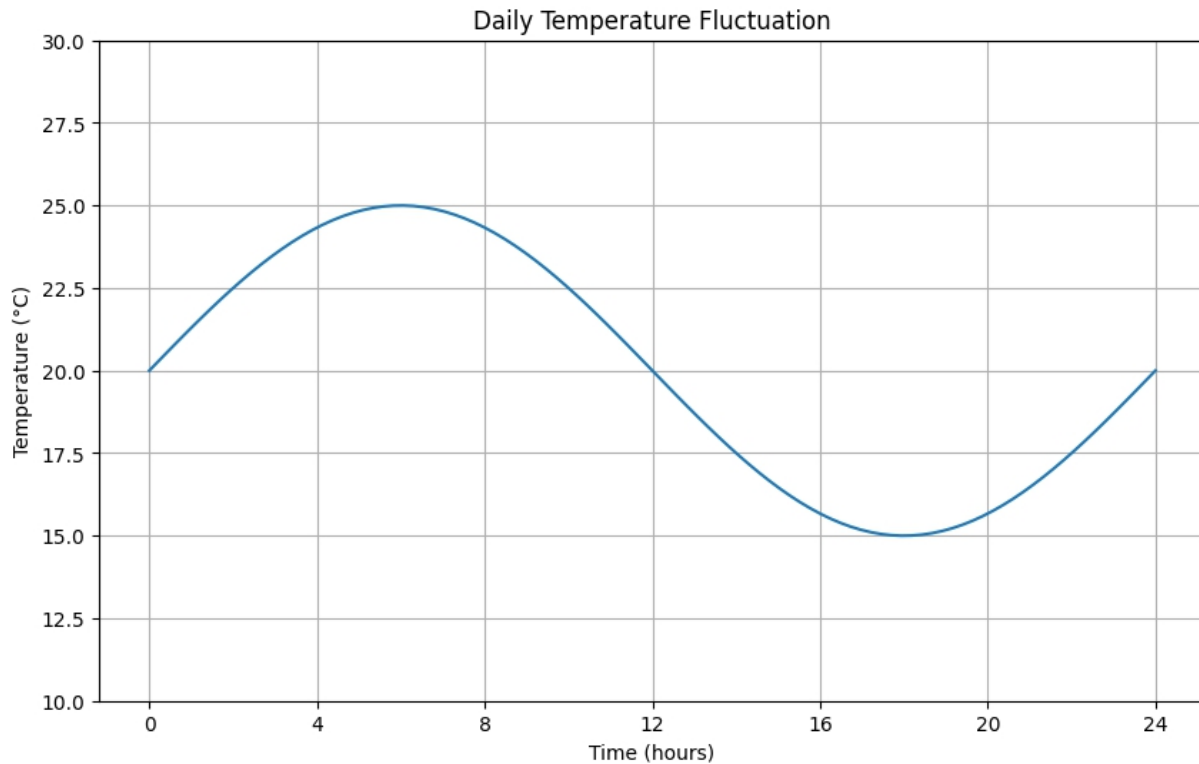
Please create the input data within your code and produce a clear, labeled plot.

Make sure to include appropriate titles and labels for the axes.

【Data Generation Code Example】

```
import numpy as np
x = np.linspace(0, 24, 100)
y = 5 * np.sin(2 * np.pi * x / 24) + 20
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 24, 100)
y = 5 * np.sin(2 * np.pi * x / 24) + 20
plt.figure(figsize=(10, 6))
plt.plot(x, y)
plt.title('Daily Temperature Fluctuation')
plt.xlabel('Time (hours)')
plt.ylabel('Temperature (°C)')
plt.grid(True)
plt.xticks(np.arange(0, 25, 4))
plt.ylim(10, 30)
```

```
plt.show()
```

This code demonstrates how to create a sine wave plot using numpy and matplotlib in Python.

Let's break down the code and explain each part in detail:

Importing necessary libraries:

numpy (as np): Used for numerical operations and creating arrays.

matplotlib.pyplot (as plt): Used for creating plots and visualizations.

Creating the input data:

np.linspace(0, 24, 100): This function generates 100 evenly spaced points between 0 and 24, representing the hours in a day.

The sine wave equation: $5 * \text{np.sin}(2 * \text{np.pi} * x / 24) + 20$
5: Amplitude of the sine wave (temperature variation)

np.sin(): Sine function from numpy

$2 * \text{np.pi} * x / 24$: Adjusts the sine wave to complete one cycle in 24 hours

20: Shifts the sine wave vertically to center it around 20°C

Creating the plot:

plt.figure(figsize=(10, 6)): Sets the size of the plot

plt.plot(x, y): Plots the data points

plt.title(): Adds a title to the plot

plt.xlabel() and plt.ylabel(): Label the x and y axes

plt.grid(True): Adds a grid to the plot for better readability

plt.xticks(np.arange(0, 25, 4)): Sets x-axis ticks at 4-hour intervals

plt.ylim(10, 30): Sets the y-axis range from 10°C to 30°C

plt.show(): Displays the plot

This code effectively visualizes daily temperature fluctuations using a sine wave.

The resulting plot shows how temperature varies over 24 hours, with the peak occurring at the midpoint (12 hours) and the trough at the beginning/end of the day. This representation is a simplified model of daily temperature changes, which can be useful for understanding general patterns in weather data.

【Trivia】

- ▶ Sine waves are fundamental in many natural phenomena, including sound waves, light waves, and alternating electrical currents.
- ▶ In meteorology, while daily temperature patterns are often approximated by sine waves, real temperature fluctuations are typically more complex due to factors like cloud cover, precipitation, and wind.
- ▶ Matplotlib is one of the most popular plotting libraries in Python, offering a MATLAB-like interface for creating a wide variety of static, animated, and interactive visualizations.
- ▶ The use of numpy in this example demonstrates its power in handling mathematical operations efficiently, especially when working with large arrays of data.
- ▶ In more advanced weather modeling, Fourier analysis (which involves decomposing signals into sine and cosine components) is often used to analyze and predict temperature patterns over longer periods.

4. Creating Box Plots of Exam Scores

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst working for a school district. The principal has asked you to analyze the exam scores of students from different classes to compare their performance.

Your task is to create a box plot that visualizes the distribution of exam scores for each class.

The data consists of exam scores for three classes: Class A, Class B, and Class C.

Each class has 30 students.

The scores are on a scale of 0 to 100.

Your objectives are:

Generate sample data for the three classes.

Create a box plot using matplotlib to compare the exam scores across the three classes.

Properly label the x-axis with class names and the y-axis with "Exam Scores".

Add a title to the plot: "Comparison of Exam Scores Across Classes".

Display the plot.

Write a Python script that accomplishes these tasks.

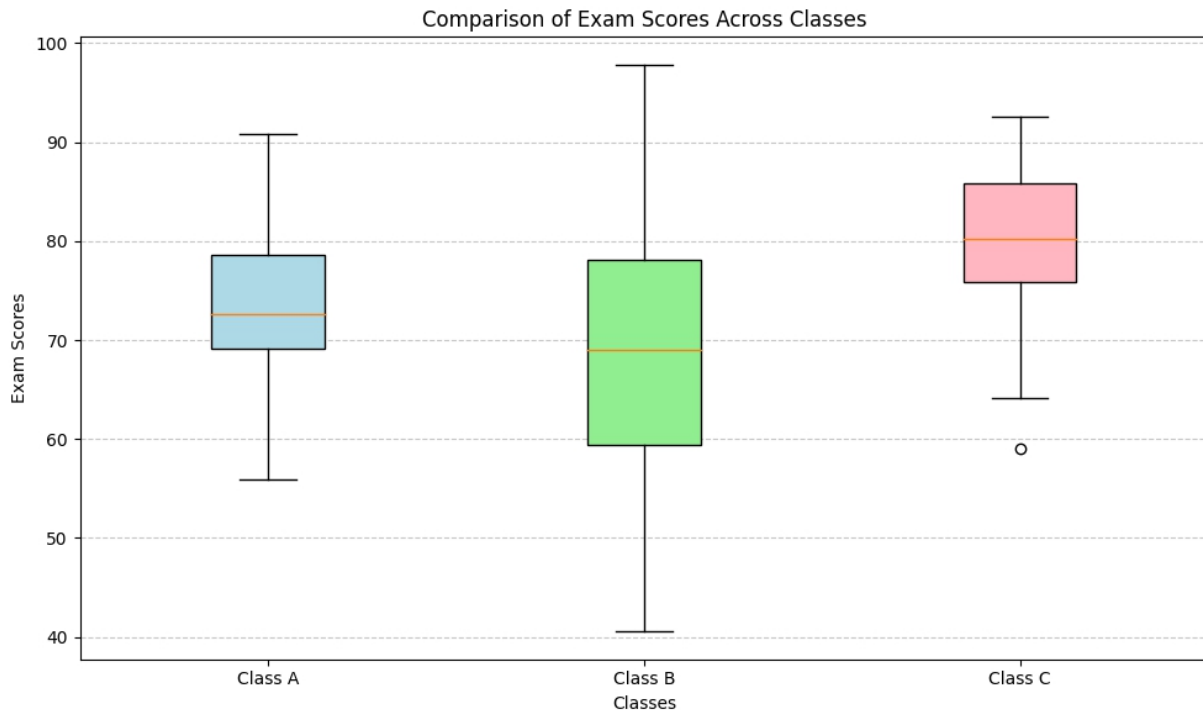
Make sure to use appropriate data structures and matplotlib functions to create an informative and visually appealing box plot.

【Data Generation Code Example】

```
import numpy as np
np.random.seed(42)
class_a = np.random.normal(75, 10, 30).clip(0, 100)
class_b = np.random.normal(70, 15, 30).clip(0, 100)
```

```
class_c = np.random.normal(80, 8, 30).clip(0, 100)
data = [class_a, class_b, class_c]
```


【Diagram Answer】



【Code Answer】

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)
# Generate sample data for three classes
class_a = np.random.normal(75, 10, 30).clip(0, 100)
class_b = np.random.normal(70, 15, 30).clip(0, 100)
class_c = np.random.normal(80, 8, 30).clip(0, 100)
# Combine data into a list
data = [class_a, class_b, class_c]
# Create the box plot
fig, ax = plt.subplots(figsize=(10, 6))
box_plot = ax.boxplot(data, labels=['Class A', 'Class B',
'Class C'], patch_artist=True)
```

```

# Customize the box plot colors
colors = ['lightblue', 'lightgreen', 'lightpink']
[box_plot['boxes'][i].set_facecolor(colors[i]) for i in
range(len(colors))]
# Set labels and title
ax.set_xlabel('Classes')
ax.set_ylabel('Exam Scores')
ax.set_title('Comparison of Exam Scores Across Classes')
# Add grid lines for better readability
ax.yaxis.grid(True, linestyle='--', alpha=0.7)
# Display the plot
plt.tight_layout()
plt.show()

```

This Python script creates a box plot to compare exam scores across three different classes.

Let's break down the code and explain the data processing and visualization steps:

Data Generation:

We use NumPy's random number generation to create synthetic exam scores for three classes.

The `np.random.normal()` function generates normally distributed random numbers.

For each class, we specify a mean score, standard deviation, and number of students (30).

The `.clip(0, 100)` method ensures all scores are between 0 and 100.

Data Structure:

The exam scores for each class are stored in separate NumPy arrays.

These arrays are then combined into a list called data for easy plotting.

Creating the Plot:

We use matplotlib to create the box plot.

`fig, ax = plt.subplots(figsize=(10, 6))` creates a new figure and axes with a specified size.

`ax.boxplot()` is used to create the box plot, passing in our data list.

Customizing the Plot:

We set labels for each box using the labels parameter in `boxplot()`.

The `patch_artist=True` argument allows us to color the boxes.

We define a list of colors and use a list comprehension to set the face color of each box.

Adding Labels and Title:

`ax.set_xlabel()` and `ax.set_ylabel()` are used to label the x and y axes.

`ax.set_title()` adds a title to the plot.

Enhancing Readability:

We add horizontal grid lines using `ax.yaxis.grid()` to make it easier to read the score values.

Displaying the Plot:

`plt.tight_layout()` adjusts the plot layout to prevent overlapping.

`plt.show()` displays the final plot.

This box plot provides a visual comparison of exam score distributions across the three classes.

Each box represents a class, showing the median, quartiles, and potential outliers.

The different colors make it easy to distinguish between classes at a glance.

【Trivia】

- ▶ Box plots, also known as box-and-whisker plots, were introduced by John Tukey in 1970 as part of Exploratory Data Analysis.
- ▶ In a box plot, the box represents the interquartile range (IQR), which contains the middle 50% of the data.
- ▶ The line inside the box represents the median, while the whiskers typically extend to 1.5 times the IQR.
- ▶ Points beyond the whiskers are often considered outliers and are plotted individually.
- ▶ Box plots are particularly useful for comparing distributions across groups and identifying skewness in data.
- ▶ While matplotlib is used here, other Python libraries like Seaborn can create box plots with even less code.
- ▶ In educational data analysis, box plots are frequently used to compare test scores across different groups, such as classes, schools, or demographic categories.
- ▶ The `np.random.seed()` function is used to ensure reproducibility of random number generation, which is crucial for scientific and educational demonstrations.

5. Heatmap of Correlation Matrix

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst at a retail company, and you have been given a dataset containing various sales metrics. Your task is to create a heatmap of the correlation matrix to identify the relationships between different metrics. This will help the company understand which metrics are closely related and can influence each other.

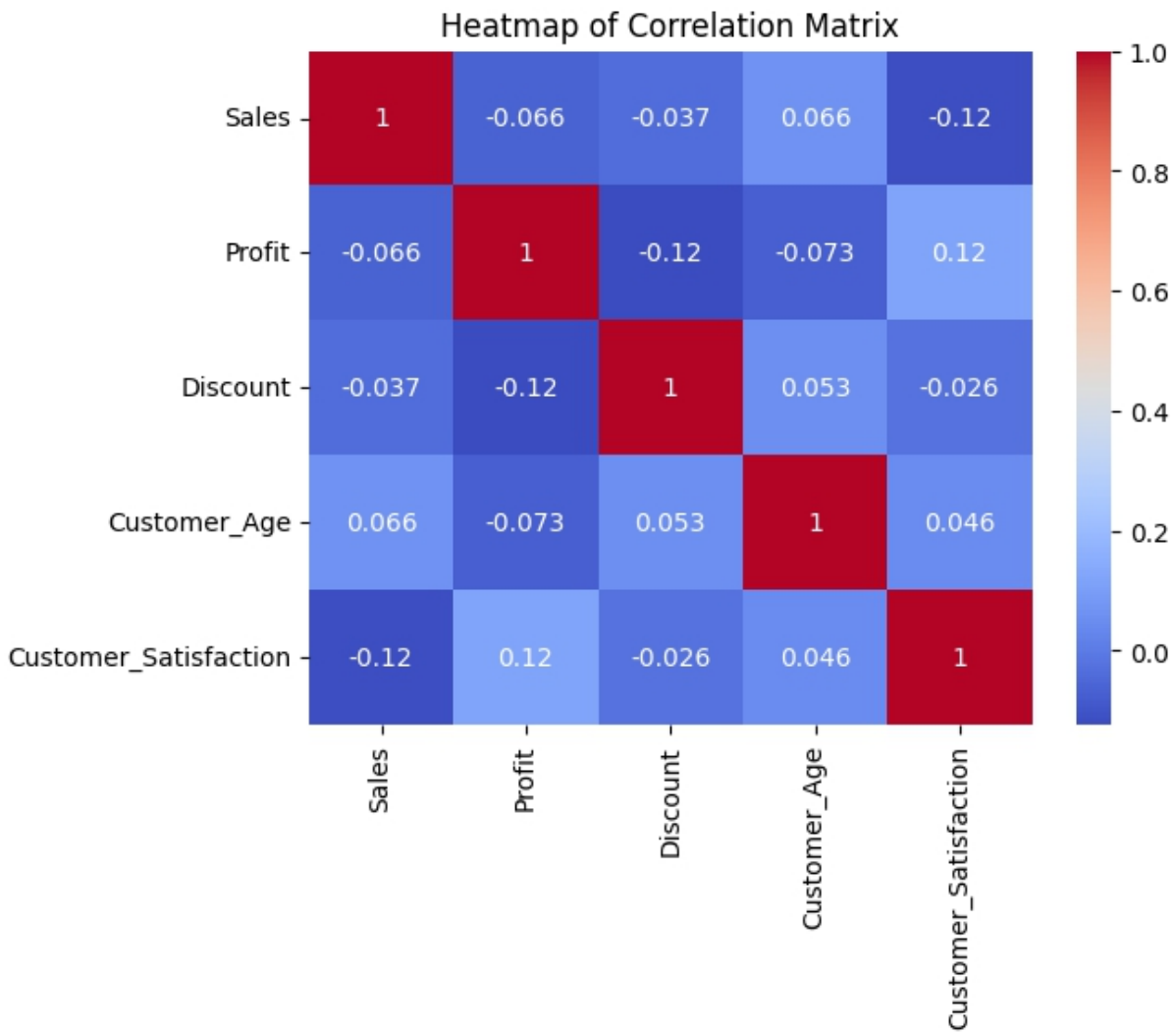
Generate a sample dataset with at least five different sales metrics and create a heatmap of the correlation matrix using Python. The heatmap should clearly show the correlation values between the metrics.

Use the following code to generate the sample dataset:

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(0)
data = pd.DataFrame({
'Sales': np.random.rand(100) * 1000,
'Profit': np.random.rand(100) * 500,
'Discount': np.random.rand(100) * 50,
'Customer_Age': np.random.randint(18, 70, size=100),
'Customer_Satisfaction': np.random.randint(1, 10, size=100)
})
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
np.random.seed(0)
data = pd.DataFrame({
    'Sales': np.random.rand(100) * 1000,
```

```
'Profit': np.random.rand(100) * 500,  
'Discount': np.random.rand(100) * 50,  
'Customer_Age': np.random.randint(18, 70, size=100),  
'Customer_Satisfaction': np.random.randint(1, 10, size=100)  
)  
correlation_matrix = data.corr()  
sns.heatmap(correlation_matrix, annot=True,  
cmap='coolwarm')  
plt.title('Heatmap of Correlation Matrix')  
plt.show()
```

To create a heatmap of the correlation matrix, follow these steps:

Import necessary libraries: You need pandas for data manipulation, numpy for numerical operations, seaborn for creating the heatmap, and matplotlib for displaying the plot.

Generate the sample dataset: The dataset includes five metrics: 'Sales', 'Profit', 'Discount', 'Customer_Age', and 'Customer_Satisfaction'. Random values are generated for each metric using numpy.

Calculate the correlation matrix: Use the `corr()` method from pandas to compute the correlation matrix of the dataset. This matrix shows the correlation coefficients between each pair of metrics.

Create the heatmap: Use seaborn's `heatmap()` function to create the heatmap. The `annot=True` parameter adds the correlation values to the heatmap cells, and `cmap='coolwarm'` sets the color scheme.

Display the heatmap: Use matplotlib's `show()` function to display the heatmap. The title is set using `plt.title()`.

This process helps visualize the relationships between different metrics, making it easier to identify which metrics are positively or negatively correlated.

【Trivia】

- ▶ Correlation Coefficient: The correlation coefficient ranges from -1 to 1. A value close to 1 indicates a strong positive correlation, while a value close to -1 indicates a strong negative correlation. A value around 0 indicates no correlation.
- ▶ Heatmap Color Schemes: The color scheme used in a heatmap can significantly impact its readability. Common schemes include 'coolwarm', 'viridis', and 'plasma'. Each scheme has its advantages depending on the data distribution and the audience's preference.
- ▶ Seaborn Library: Seaborn is built on top of Matplotlib and provides a high-level interface for drawing attractive and informative statistical graphics. It is particularly useful for visualizing complex datasets.

6. Simple Time Series Data Plotting

Importance★★★★☆

Difficulty★★☆☆☆

You are a data analyst at a retail company. Your manager has asked you to analyze the sales data for the past 12 months and visualize it to identify any trends or patterns. Create a time series plot using Python to display the sales data.

Generate the sales data within your code.

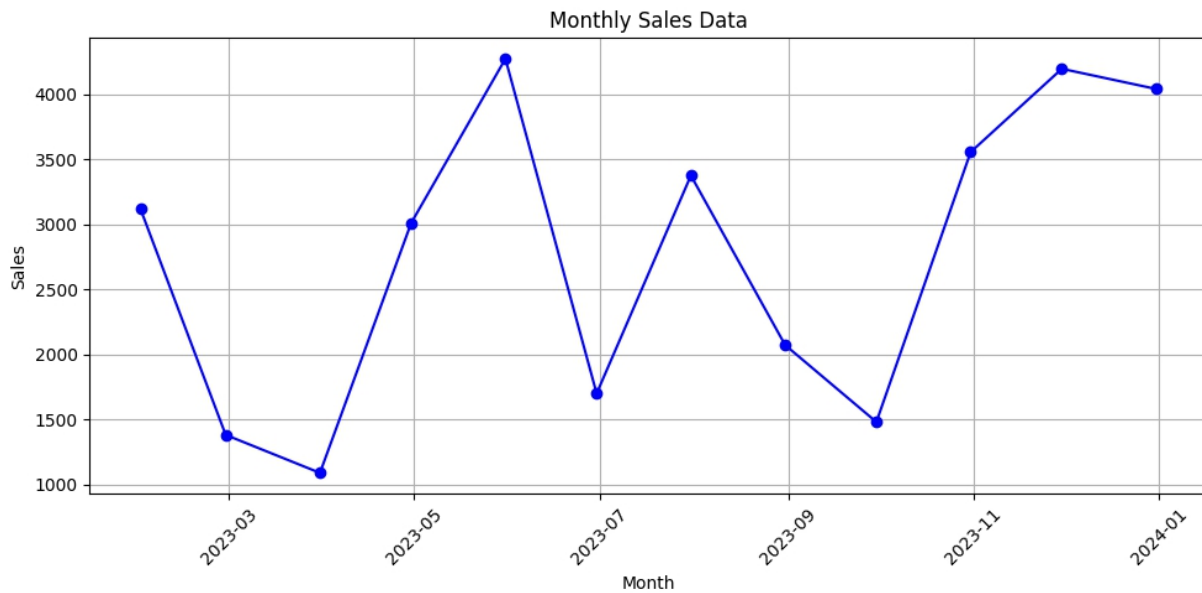
The x-axis should represent the months, and the y-axis should represent the sales figures.

Ensure that the plot is clear and well-labeled.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
months = pd.date_range(start='2023-01-01', periods=12,
freq='M')
sales = np.random.randint(1000, 5000, size=12)
data = pd.DataFrame({'Month': months, 'Sales': sales})
```

【Diagram Answer】



【Code Answer】

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
months = pd.date_range(start='2023-01-01', periods=12,
freq='M')
sales = np.random.randint(1000, 5000, size=12)
data = pd.DataFrame({'Month': months, 'Sales': sales})
plt.figure(figsize=(10, 5))
plt.plot(data['Month'], data['Sales'], marker='o', linestyle='-',
, color='b')
plt.title('Monthly Sales Data')
plt.xlabel('Month')
plt.ylabel('Sales')
plt.grid(True)
plt.xticks(rotation=45)
```

```
plt.tight_layout()  
plt.show()
```

To create a time series plot in Python, we use the matplotlib library, which is a powerful tool for data visualization. First, we import the necessary libraries: matplotlib.pyplot for plotting, pandas for handling data, and numpy for generating random sales data.

We generate a range of dates representing the last 12 months using `pd.date_range` and create random sales figures using `np.random.randint`.

These are combined into a DataFrame for easy manipulation.

In the plotting section, we use `plt.plot` to create the line plot, specifying the x and y data.

We add a title, labels for the axes, and grid lines for better readability.

The `plt.xticks(rotation=45)` command rotates the x-axis labels to prevent overlap.

Finally, `plt.tight_layout()` ensures that the plot elements fit within the figure area, and `plt.show()` displays the plot.

【Trivia】

- ▶ Time series analysis is crucial in various fields such as finance, economics, and retail to understand trends and make forecasts.
- ▶ The matplotlib library is highly customizable, allowing for detailed and complex visualizations.
- ▶ Pandas is often used in conjunction with matplotlib because it simplifies data manipulation and preparation for plotting.
- ▶ Random data generation using numpy is a common practice for creating sample datasets for testing and demonstration purposes.

7. Quarterly Sales Stacked Bar Chart

Importance★★★★☆

Difficulty★★★☆☆

You are a data analyst at a retail company. Your manager has asked you to create a stacked bar chart to visualize the quarterly sales of three different products (Product A, Product B, and Product C) over the year 2023. The data should be generated within the code. Your task is to write a Python script that generates this data and then creates a stacked bar chart to display it.

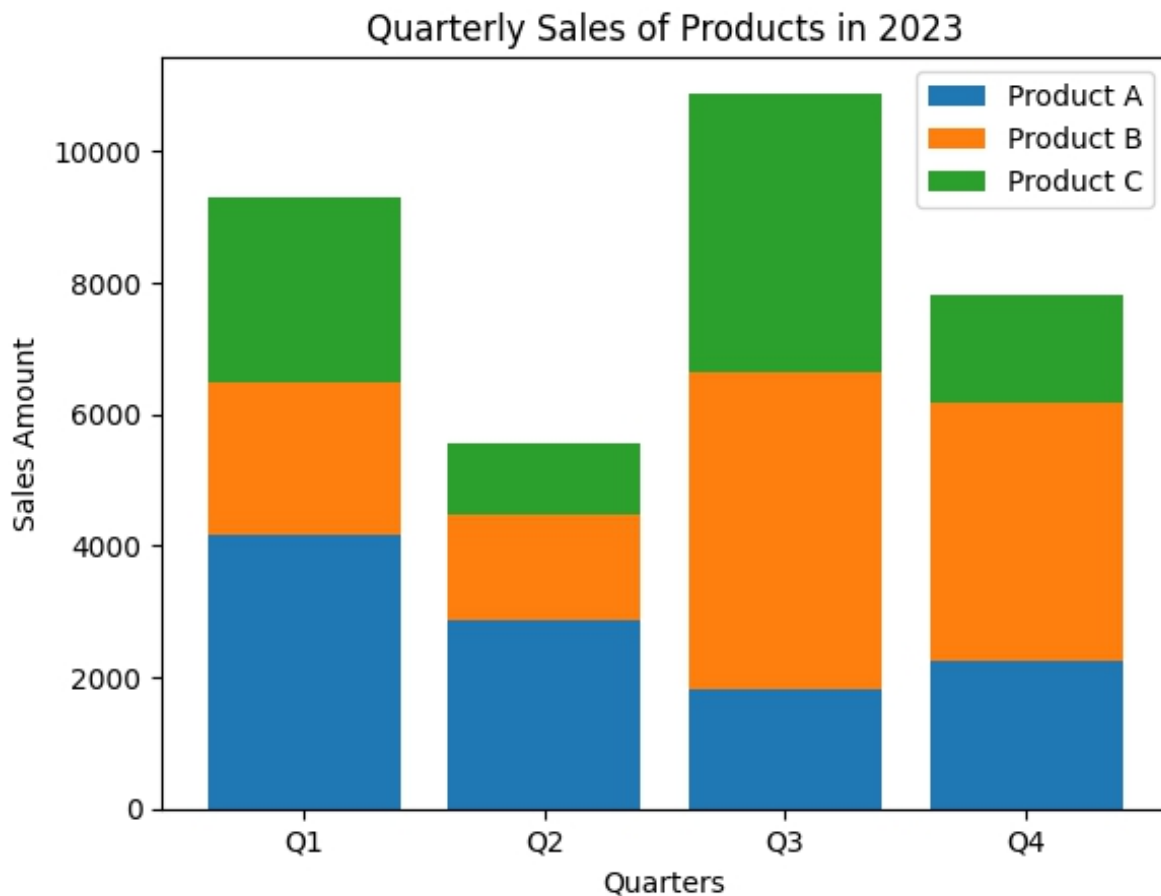
The chart should have:

- ▶ The x-axis representing the quarters (Q1, Q2, Q3, Q4).
- ▶ The y-axis representing the sales amount.
- ▶ Different colors for each product.
- ▶ A legend to distinguish between the products.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
quarters = ['Q1', 'Q2', 'Q3', 'Q4']
products = ['Product A', 'Product B', 'Product C']
data = {product: np.random.randint(1000, 5000, size=4) for
product in products}
data['Quarter'] = quarters
df = pd.DataFrame(data)
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
quarters = ['Q1', 'Q2', 'Q3', 'Q4']
products = ['Product A', 'Product B', 'Product C']
data = {product: np.random.randint(1000, 5000, size=4) for
product in products}
data['Quarter'] = quarters
df = pd.DataFrame(data)
fig, ax = plt.subplots()
```

```
ax.bar(df['Quarter'], df['Product A'], label='Product A')
ax.bar(df['Quarter'], df['Product B'], bottom=df['Product A'],
label='Product B')
ax.bar(df['Quarter'], df['Product C'], bottom=df['Product
A']+df['Product B'], label='Product C')
ax.set_xlabel('Quarters')
ax.set_ylabel('Sales Amount')
ax.set_title('Quarterly Sales of Products in 2023')
ax.legend()
plt.show()
```

To create a stacked bar chart of quarterly sales, we start by generating the data.

We use the numpy library to create random sales data for three products across four quarters.

The data is stored in a dictionary and then converted into a pandas DataFrame for easy manipulation.

Next, we use the matplotlib library to create the stacked bar chart.

We initialize a figure and axis using `plt.subplots()`.

We then plot the sales data for each product using the `ax.bar()` function.

The `bottom` parameter is used to stack the bars on top of each other.

For example, the sales of Product B are stacked on top of Product A, and Product C is stacked on top of both.

We set the labels for the x-axis and y-axis, and add a title to the chart.

Finally, we add a legend to distinguish between the products and display the chart using `plt.show()`.

【Trivia】

- ▶ Stacked bar charts are useful for comparing the total values across categories as well as the individual contributions to the total.
- ▶ The pandas library is widely used for data manipulation and analysis in Python, making it a powerful tool for preparing data for visualization.
- ▶ matplotlib is one of the most popular plotting libraries in Python, known for its flexibility and extensive customization options.

8. Visualizing Multiple Functions with Python

Importance★★★★☆

Difficulty★★★☆☆

You are a data analyst working for a renewable energy company.

The company wants to compare the power output of different types of solar panels over a 24-hour period.

You have been given data for three types of solar panels: monocrystalline, polycrystalline, and thin-film.

Your task is to create a multi-line graph that visualizes the power output of these three types of solar panels from 0 to 24 hours.

The data should be generated within your code using the following guidelines:

Time range: 0 to 24 hours (x-axis)

Power output range: 0 to 100 watts (y-axis)

Monocrystalline panels: Highest efficiency, peaking around midday

Polycrystalline panels: Slightly lower efficiency than monocrystalline

Thin-film panels: Lower efficiency but more consistent throughout the day

Create a Python script that generates this data and plots a multi-line graph with the following requirements:

Use appropriate libraries for data manipulation and visualization.

Generate data points for each hour from 0 to 24.

Create three lines on the graph, one for each type of solar panel.

Include a legend to identify each line.

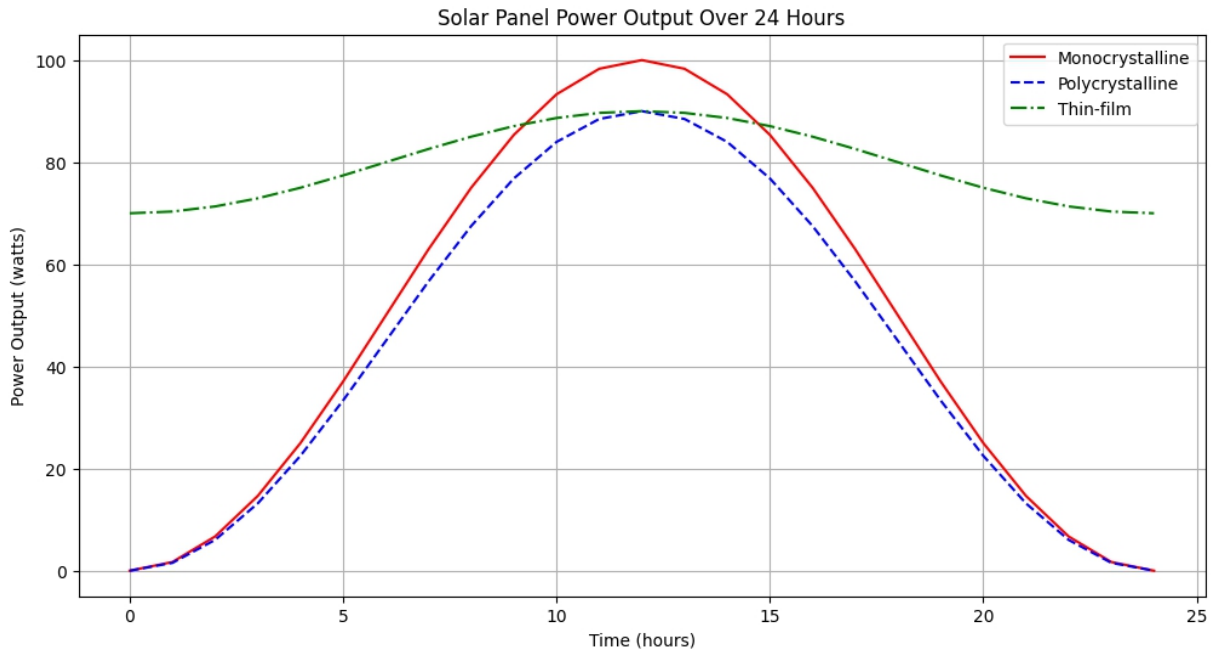
Label the x-axis as "Time (hours)" and the y-axis as "Power Output (watts)".

Give the graph an appropriate title.
Use different colors and/or line styles to distinguish between the three types of panels.
Your code should be efficient, well-commented, and follow Python best practices.

【Data Generation Code Example】

```
import numpy as np
hours = np.arange(25)
monocrystalline = 100 * np.sin(np.pi * hours / 24) ** 2
polycrystalline = 90 * np.sin(np.pi * hours / 24) ** 2
thin_film = 70 + 20 * np.sin(np.pi * hours / 24) ** 2
data = np.column_stack((hours, monocrystalline,
polycrystalline, thin_film))
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import matplotlib.pyplot as plt
# Generate time data
hours = np.arange(25)
# Generate power output data for each panel type
monocrystalline = 100 * np.sin(np.pi * hours / 24) ** 2
polycrystalline = 90 * np.sin(np.pi * hours / 24) ** 2
thin_film = 70 + 20 * np.sin(np.pi * hours / 24) ** 2
# Create the plot
plt.figure(figsize=(12, 6))
plt.plot(hours, monocrystalline, 'r-', label='Monocrystalline')
plt.plot(hours, polycrystalline, 'b--', label='Polycrystalline')
plt.plot(hours, thin_film, 'g-.', label='Thin-film')
# Customize the plot
```

```
plt.title('Solar Panel Power Output Over 24 Hours')
plt.xlabel('Time (hours)')
plt.ylabel('Power Output (watts)')
plt.legend()
plt.grid(True)
# Display the plot
plt.show()
```

This Python script creates a multi-line graph to visualize the power output of three different types of solar panels over a 24-hour period.

Let's break down the code and explain each part in detail:

Importing necessary libraries:

numpy (as np): Used for efficient numerical operations and array manipulations.

matplotlib.pyplot (as plt): Used for creating the plot and customizing its appearance.

Data Generation:

We use numpy's arange function to create an array of hours from 0 to 24 (inclusive).

For each type of solar panel, we generate power output data using numpy's sin function and element-wise operations:

Monocrystalline: Highest efficiency, modeled as a perfect sine wave squared, reaching 100 watts at peak.

Polycrystalline: Slightly lower efficiency, modeled similarly but with a peak of 90 watts.

Thin-film: More consistent output, modeled as a sine wave squared with a base of 70 watts and a peak addition of 20 watts.

Creating the Plot:

`plt.figure(figsize=(12, 6))`: Creates a new figure with a width of 12 inches and a height of 6 inches.

`plt.plot()`: This function is called three times, once for each type of solar panel. It plots the hours on the x-axis and the corresponding power output on the y-axis.

The first argument is the x-axis data (hours).

The second argument is the y-axis data (power output for each panel type).

The third argument is a string that specifies the color and line style:

'r-': Red solid line for monocrystalline

'b--': Blue dashed line for polycrystalline

'g-.': Green dash-dot line for thin-film

The label argument assigns a name to each line for the legend.

Customizing the Plot:

`plt.title()`: Sets the title of the graph.

`plt.xlabel()` and `plt.ylabel()`: Label the x and y axes, respectively.

`plt.legend()`: Adds a legend to the plot, using the labels specified in the `plt.plot()` calls.

`plt.grid(True)`: Adds a grid to the plot for easier reading of values.

Displaying the Plot:

`plt.show()`: This function displays the plot.

This code demonstrates several key concepts in Python data visualization:

Using numpy for efficient data generation and manipulation

Creating a multi-line plot with matplotlib

Customizing plot appearance (colors, line styles, labels, title, legend, grid)

Modeling real-world phenomena (solar panel efficiency) with mathematical functions

The resulting graph effectively compares the power output of the three types of solar panels, allowing for easy visual analysis of their performance throughout the day.

【Trivia】

- ▶ Solar panel efficiency is typically measured as a percentage of the sun's energy that is converted into electricity. Monocrystalline panels are generally the most efficient, with some models reaching over 22% efficiency.
- ▶ The power output of solar panels is affected by various factors including temperature, shading, and the angle of sunlight. This is why the curves in our graph are not perfect semicircles.
- ▶ Thin-film solar panels, while less efficient, can perform better in low-light conditions and at high temperatures compared to crystalline silicon panels.
- ▶ The time of peak solar energy production is typically around solar noon, which is why our simulated data peaks at the 12-hour mark.
- ▶ Python's matplotlib library, used in this example, is based on MATLAB's plotting interface. It was originally developed as a patch to make Python more MATLAB-like for a group of neuroimaging researchers.
- ▶ The sine function used to generate our data is a fundamental trigonometric function that repeats every 2π radians. By manipulating this function (squaring it and adjusting its amplitude and offset), we can model a wide variety of natural phenomena, including the daily cycle of solar energy production.
- ▶ In real-world applications, more complex models would be used to predict solar panel output, taking into account factors such as weather conditions, panel orientation, and geographical location.

9. Bubble Chart of Population vs GDP

Importance★★★★☆

Difficulty★★★★☆☆

A multinational corporation is evaluating potential markets for expansion. They are considering countries based on their population and GDP.

Your task is to generate a bubble chart that visualizes the relationship between population and GDP for various countries.

The size of the bubbles should represent the population size.

Use the following dataset:

Country: ['Country A', 'Country B', 'Country C', 'Country D', 'Country E']

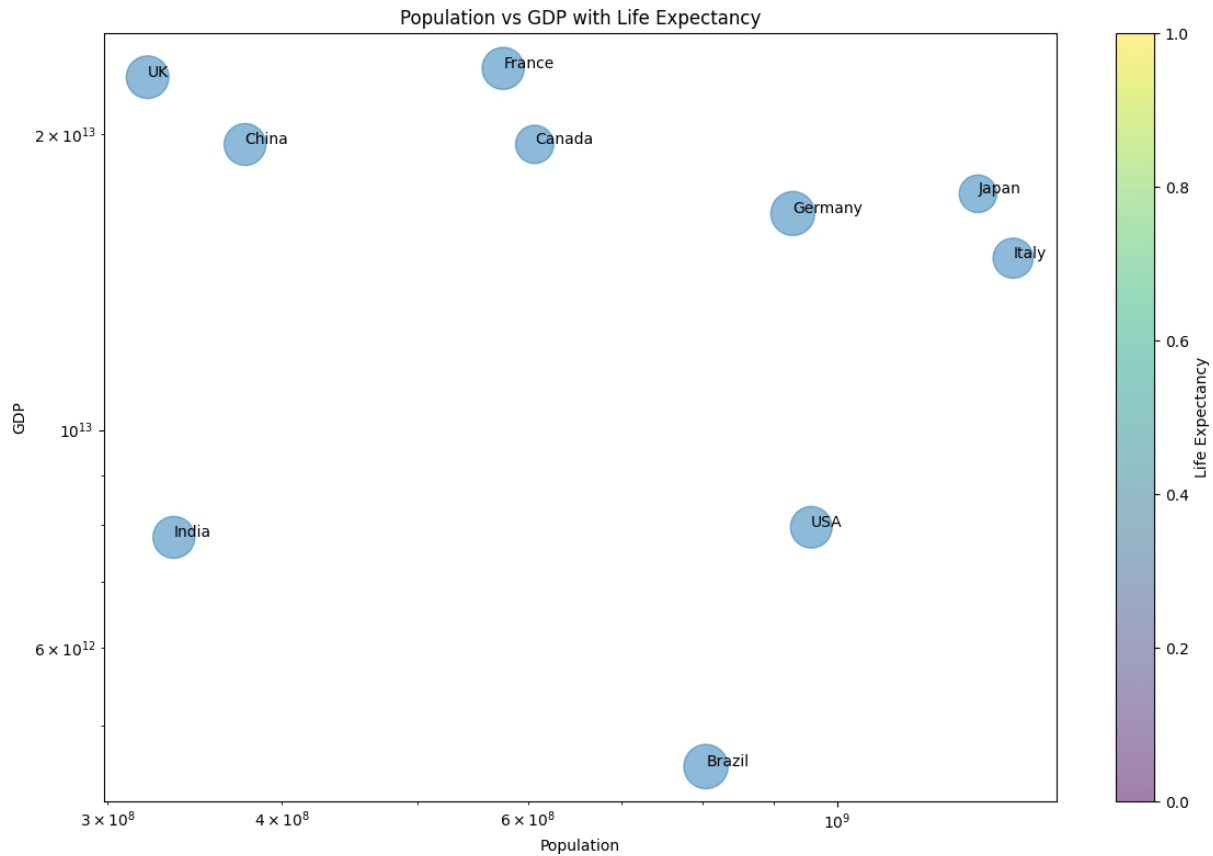
Population (in millions): [50, 80, 120, 60, 90]

GDP (in billion USD): [500, 700, 1500, 800, 1000]

【Data Generation Code Example】

```
## Create data for the bubble chart
import pandas as pd
countries=['Country A', 'Country B', 'Country C', 'Country D',
'Country E']
population=[50, 80, 120, 60, 90]
gdp=[500, 700, 1500, 800, 1000]
data=pd.DataFrame({'Country','Population','GDP'})
```

【Diagram Answer】



【Code Answer】

```
## Importing required libraries
import pandas as pd
import matplotlib.pyplot as plt
## Create data for the bubble chart
countries=['Country A','Country B','Country C','Country D','Country E']
population=[50,80,120,60,90]
gdp=[500,700,1500,800,1000]
data=pd.DataFrame({'Country','Population','GDP'})
## Create bubble chart
```

```
plt.scatter(data['GDP'],data['Population'],s=[pop*10 for pop
in data['Population']],alpha=0.5)
for i in range(len(data)).text(data['GDP'][i],data['Population']
[i],data['Country'][i],fontsize=9)
plt.title('Population vs GDP Bubble Chart')
plt.xlabel('GDP (in billion USD)')
plt.ylabel('Population (in millions)')
plt.grid(True)
plt.show()
```

This exercise involves creating a bubble chart to visualize the relationship between population and GDP for different countries.

First, we import the necessary libraries: pandas for data manipulation and matplotlib for plotting.

We create a DataFrame using pandas with the provided data, listing countries, their populations, and GDPs. The population data will be used to determine the size of the bubbles.

The scatter function from matplotlib is used to create the bubble chart. The x-axis represents GDP, and the y-axis represents population. The size of the bubbles (s parameter) is set proportional to the population, with an alpha value to make the bubbles semi-transparent.

A loop adds text labels to each bubble, showing the country name at the corresponding (GDP, population) coordinates.

Titles and labels are added to the chart for better readability, and grid lines are enabled for easier interpretation.

Finally, plt.show() displays the chart.

This problem helps users practice data visualization, specifically creating and customizing bubble charts, which is useful for comparing multiple variables simultaneously.

【Trivia】

Bubble charts are a variation of scatter plots where a third dimension of the data is shown through the size of markers. They are especially useful in situations where you need to show the relationship between three different measures. For example, in business, they can be used to plot financial data, such as revenue, profit, and market share, giving a multi-dimensional view of performance. Additionally, bubble charts can help in identifying patterns, trends, and outliers in complex datasets, making them a valuable tool in data analysis and presentation.

10. Create a Pair Plot of Iris Dataset

Importance★★★★☆

Difficulty★★★★☆☆

You are working as a data analyst for a botanical research organization.

Your task is to analyze the famous Iris dataset to understand the relationships between different features.

Specifically, you need to create a pair plot to visualize these relationships.

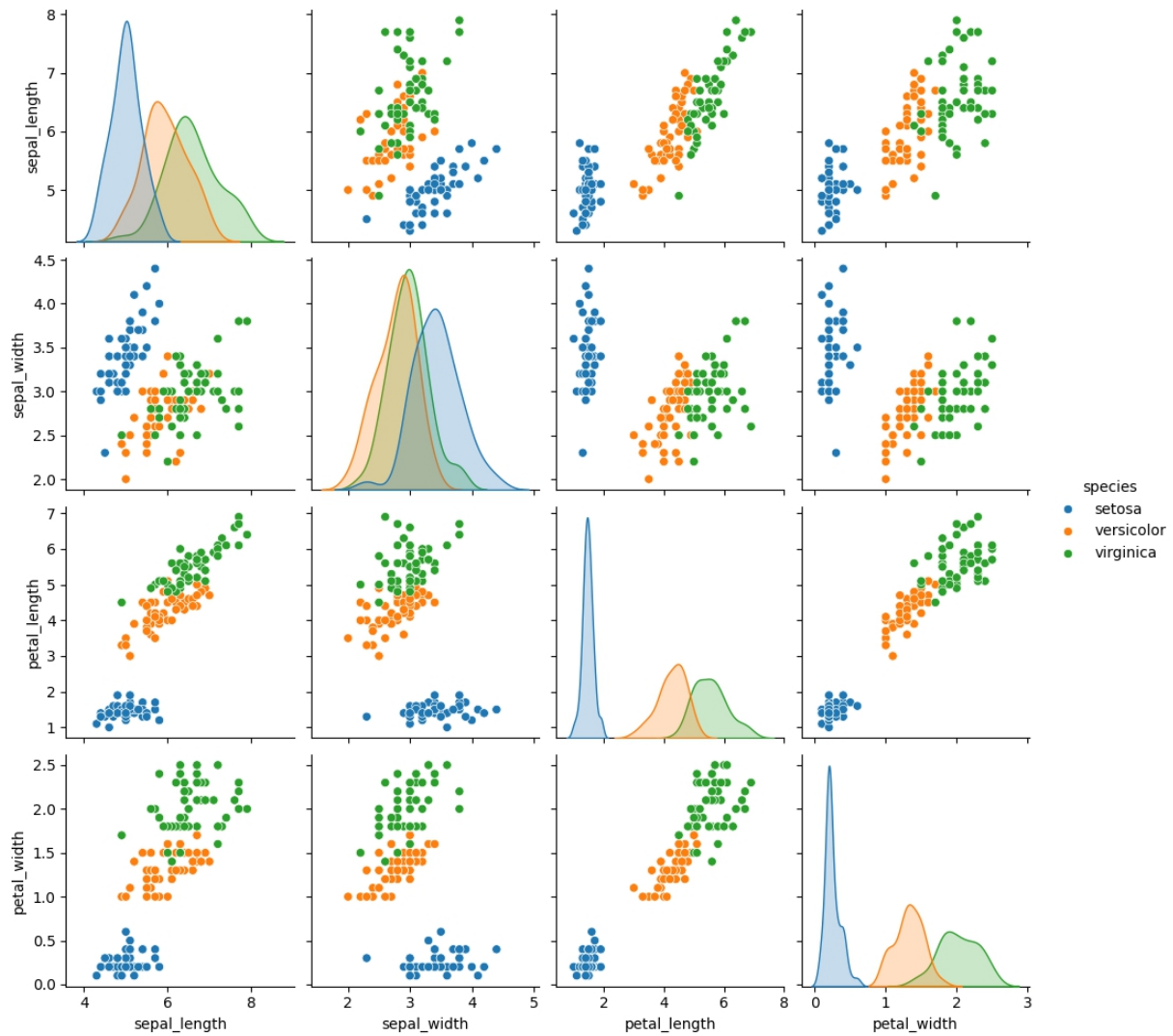
Using the Iris dataset, generate a pair plot to show the relationships between sepal length, sepal width, petal length, and petal width.

Make sure to color-code the different species in the dataset.

【Data Generation Code Example】

```
import pandas as pd
from seaborn import load_dataset
df = load_dataset('iris')
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = sns.load_dataset('iris')
sns.pairplot(df, hue='species')
plt.show()
```

First, the necessary libraries are imported: pandas for data manipulation, seaborn for the pair plot, and matplotlib for displaying the plot.

The Iris dataset is loaded directly using seaborn's `load_dataset` function, which simplifies the process.

The pair plot is created using seaborn's `pairplot` function, with the 'hue' parameter set to 'species' to color-code the different species.

Finally, `plt.show()` is called to display the plot. This function is necessary to render the plot in some environments.

The pair plot provides a matrix of scatter plots that display relationships between each pair of features in the dataset.

The diagonal of the matrix shows the distribution of each feature. Color-coding by species helps in identifying patterns and differences between the species.

【Trivia】

The Iris dataset was introduced by the British biologist and statistician Ronald Fisher in his 1936 paper, "The use of multiple measurements in taxonomic problems." It includes 150 observations of iris flowers, with four features: sepal length, sepal width, petal length, and petal width, across three species: *Iris setosa*, *Iris versicolor*, and *Iris virginica*. This dataset is widely used as a beginner's dataset for machine learning and data visualization exercises due to its simplicity and well-defined classes.

11. 3D Scatter Plot of Customer Data

Importance★★★★☆

Difficulty★★★★☆

A retail company wants to visualize their customer data in a 3D space to better understand the relationship between customer age, annual spending, and loyalty points.

Your task is to create a 3D scatter plot using Python that represents this data.

The company has provided you with the following requirements:

Generate random data for 100 customers with the following attributes:

Age: between 18 and 80 years old

Annual Spending: between \$100 and \$10,000

Loyalty Points: between 0 and 1000 points

Create a 3D scatter plot where:

X-axis represents Age

Y-axis represents Annual Spending

Z-axis represents Loyalty Points

Each point should represent a customer.

Use different colors for points based on age groups:

18-30: Red

31-50: Green

51-80: Blue

Add appropriate labels, title, and a legend to the plot.

Ensure the plot is visually appealing and easy to interpret.

Your code should include data generation and visualization in a single script.

Focus on efficient data manipulation and visualization techniques in Python.

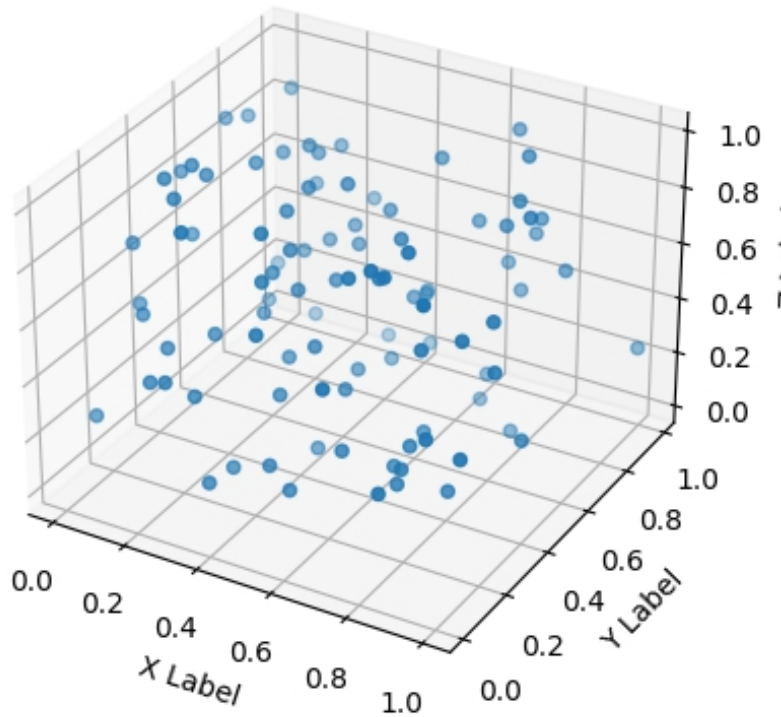
【Data Generation Code Example】



```
import numpy as np
np.random.seed(42)
age = np.random.randint(18, 81, 100)
spending = np.random.uniform(100, 10001, 100)
loyalty = np.random.randint(0, 1001, 100)
```

【Diagram Answer】

3D Scatter Plot of Random Points



【Code Answer】

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
np.random.seed(42)
age = np.random.randint(18, 81, 100)
spending = np.random.uniform(100, 10001, 100)
loyalty = np.random.randint(0, 1001, 100)
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
# Define color based on age group
```

```

colors = ['red' if a <= 30 else 'green' if a <= 50 else 'blue'
for a in age]
scatter = ax.scatter(age, spending, loyalty, c=colors, s=50,
alpha=0.6)
ax.set_xlabel('Age')
ax.set_ylabel('Annual Spending ($)')
ax.set_zlabel('Loyalty Points')
ax.set_title('3D Scatter Plot of Customer Data')
# Create legend
legend_elements = [plt.Line2D( , marker='o', color='w',
label='18-30', markerfacecolor='r', markersize=10),
plt.Line2D( , marker='o', color='w', label='31-50',
markerfacecolor='g', markersize=10),
plt.Line2D( , marker='o', color='w', label='51-80',
markerfacecolor='b', markersize=10)]
ax.legend(handles=legend_elements, title='Age Groups')
plt.tight_layout()
plt.show()

```

This code creates a 3D scatter plot to visualize customer data based on age, annual spending, and loyalty points.

Let's break down the code and explain the key concepts:

Data Generation:

We use NumPy's random functions to generate data for 100 customers.

`np.random.randint()` is used for age and loyalty points (integer values).

`np.random.uniform()` is used for annual spending (float values).

`np.random.seed(42)` ensures reproducibility of the random data.

Importing Libraries:

NumPy (`np`) for numerical operations and data generation.

Matplotlib (`plt`) for creating the plot.

`Axes3D` from `mpl_toolkits.mplot3d` for 3D plotting capabilities.

Creating the 3D Plot:

`fig = plt.figure(figsize=(10, 8))` creates a new figure with specified size.

`ax = fig.add_subplot(111, projection='3d')` adds a 3D subplot to the figure.

Color Assignment:

We use a list comprehension to assign colors based on age groups.

This creates a list of color strings ('red', 'green', or 'blue') for each customer.

Plotting the Data:

`ax.scatter()` is used to create the 3D scatter plot.

It takes the x, y, and z coordinates (age, spending, loyalty) and the color list.

`s=50` sets the size of the markers, and `alpha=0.6` makes them slightly transparent.

Setting Labels and Title:

`ax.set_xlabel()`, `ax.set_ylabel()`, and `ax.set_zlabel()` set axis labels.

`ax.set_title()` sets the plot title.

Creating a Legend:

We create custom legend elements using `plt.Line2D()`.

Each element represents an age group with its corresponding color.

`ax.legend()` adds the legend to the plot with a title.

Finalizing and Displaying the Plot:

`plt.tight_layout()` adjusts the plot layout to prevent overlapping.

`plt.show()` displays the final plot.

This code demonstrates several important concepts in Python data visualization:

3D plotting with Matplotlib

Random data generation with NumPy

Color-coding data points based on conditions

Creating custom legends

Properly labeling axes and titling plots

The resulting visualization allows for easy interpretation of the relationships between customer age, spending, and loyalty points, with color-coding providing an additional dimension of information.

【Trivia】

- ▶ 3D scatter plots are useful for visualizing relationships between three variables, but they can become cluttered with large datasets.
- ▶ Matplotlib's 3D plotting capabilities were introduced in version 1.0.0, released in 2010.
- ▶ The `mpl_toolkits.mplot3d` module provides 3D plotting tools as an extension to Matplotlib's 2D plotting capabilities.
- ▶ When working with 3D plots, it's important to consider the viewing angle, as some data points may be obscured depending on the perspective.
- ▶ The `alpha` parameter in scatter plots controls the transparency of points, which can help in visualizing overlapping data points.
- ▶ List comprehensions, as used for color assignment in this example, are a powerful and concise way to create lists in Python, often more efficient than traditional for loops.
- ▶ The `numpy.random.seed()` function is crucial for reproducibility in scientific computing and data science,

ensuring that random number generation can be replicated across different runs or by different users.

- ▶ While 3D plots can be visually appealing, they are not always the most effective way to convey information. Sometimes, multiple 2D plots or other visualization techniques might be more appropriate, depending on the data and the message you want to convey.

12. Creating a Violin Plot for Age Distribution

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst working for a company that wants to visualize the age distribution of its customers. The goal is to create a violin plot to understand the distribution better.

Use the following steps to generate a synthetic dataset of ages and then create a violin plot.

Ensure the ages range from 18 to 70, and there are 200 data points.

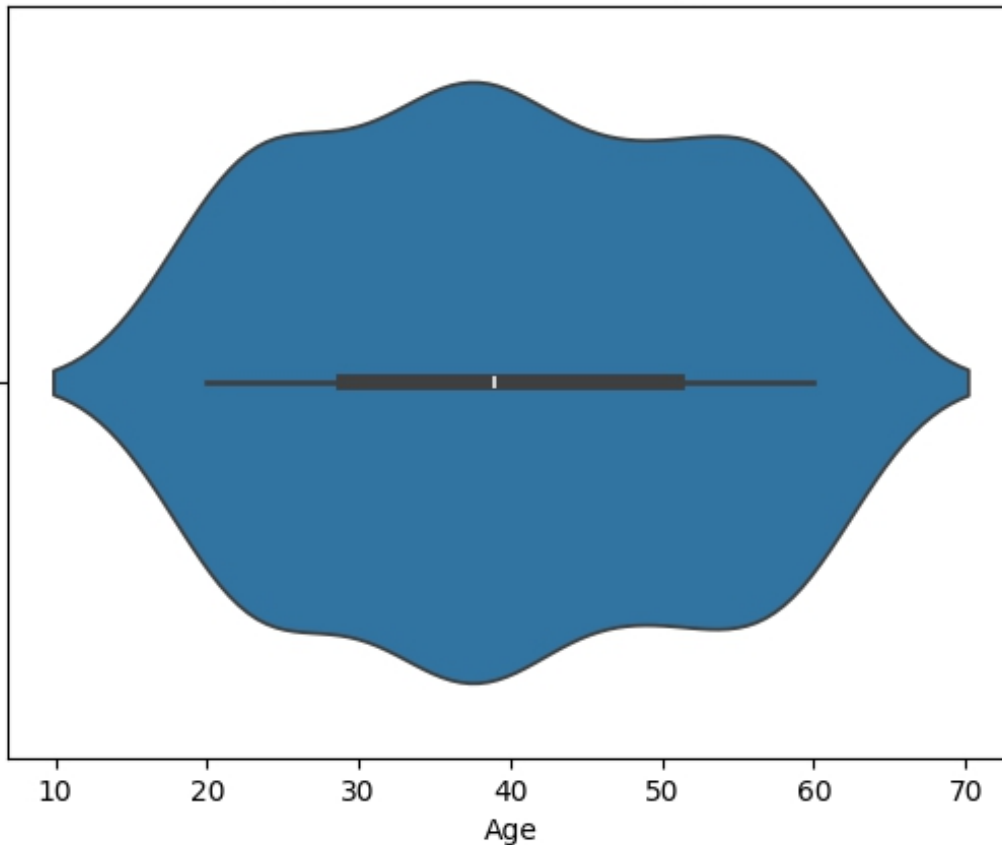
Write the necessary code to generate the data and create the plot.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
Create a synthetic dataset of ages ranging from 18 to 70
with 200 data pointsages = np.random.randint(18, 71, 200)
Convert to DataFrame for easy manipulationdata =
pd.DataFrame({'Age': ages})
data.head()
```

【Diagram Answer】

Age Distribution of Employees



【Code Answer】

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
Create a synthetic dataset of ages ranging from 18 to 70
with 200 data points
ages = np.random.randint(18, 71, 200)
Convert to DataFrame for easy manipulation
data = pd.DataFrame({'Age': ages})
Plotting the violin plot
```

```
plt.figure(figsize=(10, 6))
sns.violinplot(x=data['Age'])
plt.title('Age Distribution of Customers')
plt.xlabel('Age')
plt.show()
```

This exercise aims to teach the reader how to create a violin plot using Python.

First, synthetic data is generated to represent customer ages ranging from 18 to 70 using NumPy's `randint` function. This data is then converted into a pandas DataFrame for easier manipulation.

For visualization, the Seaborn library is used due to its simplicity and powerful plotting capabilities.

A violin plot is a combination of a box plot and a kernel density plot, showing the distribution shape of the data.

The `plt.figure` function is used to set the size of the plot.

The `sns.violinplot` function creates the violin plot, and `plt.title` and `plt.xlabel` are used to set the plot title and x-axis label, respectively.

Finally, `plt.show` displays the plot.

【Trivia】

- ▶ Violin plots are particularly useful for comparing multiple categories or distributions.
- ▶ They not only show summary statistics like a box plot but also the density of the data at different values.
- ▶ Seaborn's `violinplot` function automatically combines both the box plot and kernel density estimate, providing a comprehensive view of the data.

13. Density Plot of Random Data

Importance★★★★☆

Difficulty★★☆☆☆

You are working as a data analyst for a marketing firm. Your manager wants to visualize the distribution of customer ages in a recent survey to understand the age demographics better.

To practice, generate a random dataset of customer ages and create a density plot to visualize the distribution.

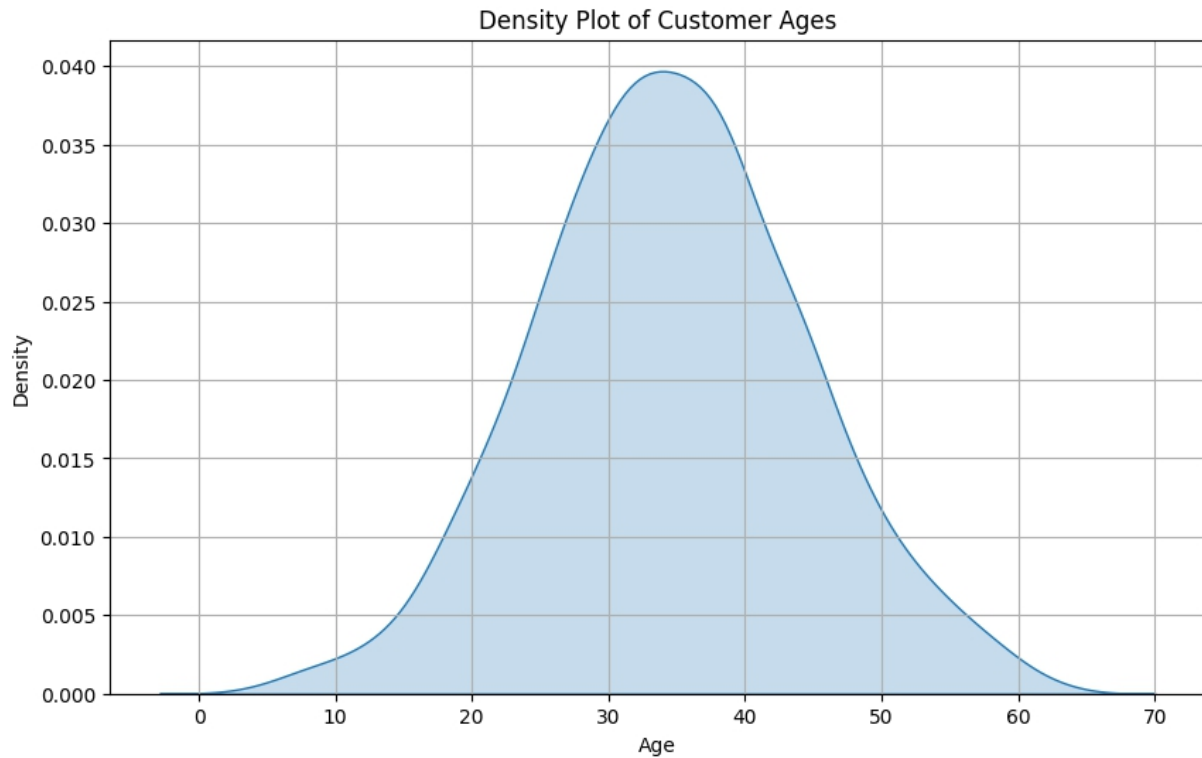
The ages should be normally distributed with a mean of 35 and a standard deviation of 10.

Provide the code to generate the random data and the density plot.

【Data Generation Code Example】

```
import numpy as np
np.random.seed(0)
ages = np.random.normal(35, 10, 1000)
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
np.random.seed(0)
ages = np.random.normal(35, 10, 1000)
plt.figure(figsize=(10, 6))
sns.kdeplot(ages, shade=True)
plt.title('Density Plot of Customer Ages')
plt.xlabel('Age')
plt.ylabel('Density')
plt.grid(True)
plt.show()
```

First, we import the necessary libraries: numpy for data generation, and matplotlib and seaborn for plotting. We set the random seed using `np.random.seed(0)` to ensure reproducibility of the random data. Next, we generate 1000 random ages following a normal distribution with a mean of 35 and a standard deviation of 10 using `np.random.normal(35, 10, 1000)`. We create a density plot using seaborn's `kdeplot` function with the `shade` parameter set to `True` to fill the area under the curve. We then set the title, x-axis label, and y-axis label for the plot using `plt.title`, `plt.xlabel`, and `plt.ylabel` respectively. Finally, we enable the grid for better readability using `plt.grid(True)` and display the plot with `plt.show()`.

【Trivia】

Density plots are useful for visualizing the distribution of data and identifying patterns or anomalies. They provide a smoothed version of a histogram, which can help in understanding the underlying distribution without the binning effects of a histogram. Seaborn is built on top of matplotlib and provides a high-level interface for drawing attractive statistical graphics.

14. Creating a Donut Chart for Budget Allocation

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst working for a small startup company. The CEO has asked you to create a visual representation of the company's budget allocation for the upcoming fiscal year.

The budget is divided into five main categories: Marketing, Research & Development, Operations, Human Resources, and IT Infrastructure.

Your task is to create a donut chart using Python that clearly shows the percentage allocation for each category.

The CEO wants to be able to quickly understand how the budget is distributed across these key areas.

Please write a Python script that:

- Creates sample data for the budget allocation

- Uses matplotlib to generate a donut chart

- Displays the percentage for each category within the chart

- Uses a color scheme that is easy to distinguish

- Includes a legend for easy reference

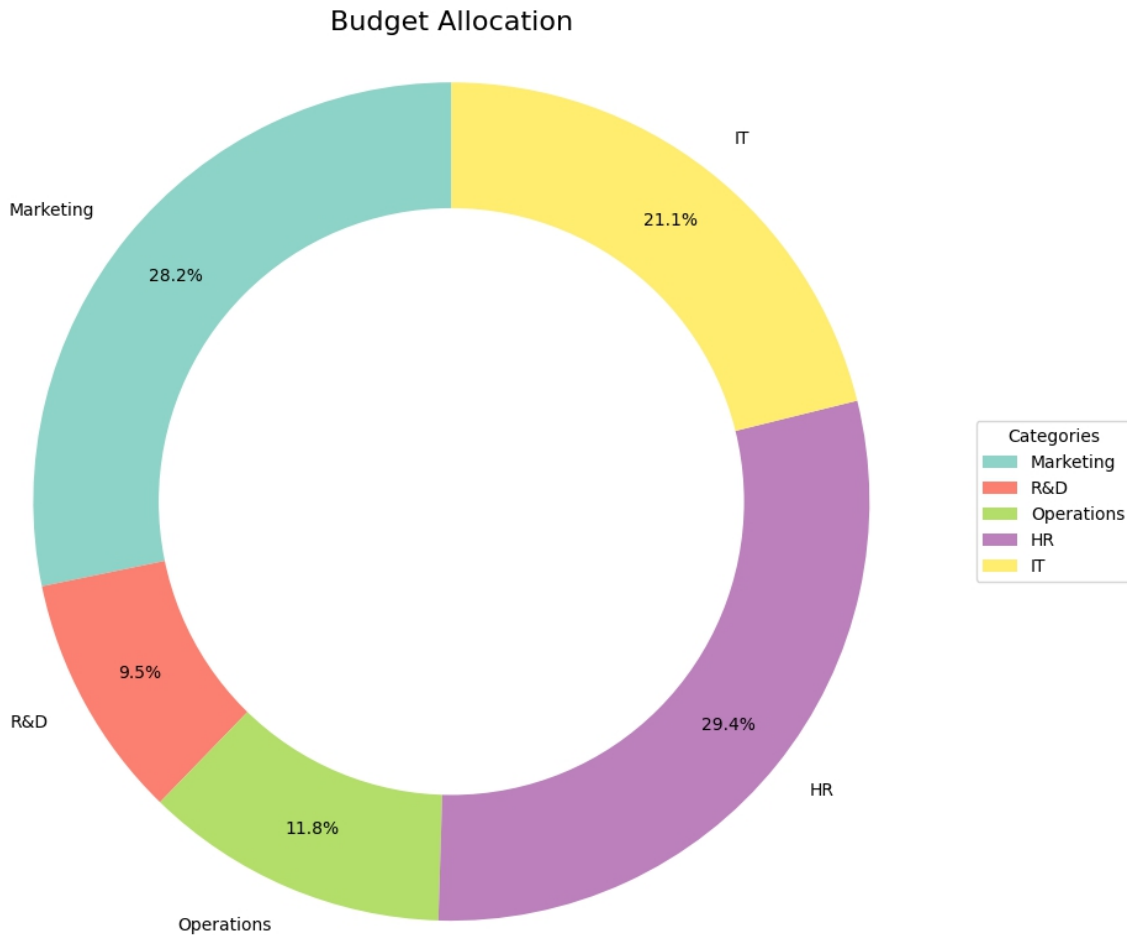
- Ensure that your code is efficient and well-commented.

The chart should be visually appealing and easy to interpret at a glance.

【Data Generation Code Example】

```
import numpy as np
# Generate sample data for budget allocation
categories = ['Marketing', 'R&D', 'Operations', 'HR', 'IT']
budget_allocation = np.random.randint(1000, 10000,
size=5)
```

【Diagram Answer】



【Code Answer】

```
import matplotlib.pyplot as plt
import numpy as np
# Generate sample data for budget allocation
categories = ['Marketing', 'R&D', 'Operations', 'HR', 'IT']
budget_allocation = np.random.randint(1000, 10000,
size=5)
# Calculate percentages
total_budget = np.sum(budget_allocation)
```

```

percentages = [round((x/total_budget)*100, 1) for x in
budget_allocation]
# Set up the plot
plt.figure(figsize=(10, 8))
# Create color palette
colors = plt.cm.Set3(np.linspace(0, 1, len(categories)))
# Create donut chart
center_circle = plt.Circle((0,0), 0.70, fc='white')
plt.pie(budget_allocation, labels=categories, colors=colors,
autopct=lambda pct: f'{pct:.1f}%', startangle=90,
pctdistance=0.85)
plt.gca().add_artist(center_circle)
# Add title and legend
plt.title('Budget Allocation', fontsize=16)
plt.legend(categories, title='Categories', loc='center left',
bbox_to_anchor=(1, 0, 0.5, 1))
# Display the chart
plt.axis('equal')
plt.tight_layout()
plt.show()

```

This Python script creates a donut chart to visualize budget allocation across different categories.

Let's break down the code and explain each part in detail:

Importing necessary libraries:

matplotlib.pyplot is imported as plt for creating the chart
numpy is imported as np for numerical operations

Data Generation:

We create a list of categories for the budget allocation

We use `np.random.randint()` to generate random integers between 1000 and 10000 for each category, simulating budget amounts

Data Processing:

We calculate the total budget by summing up all allocations
We then calculate the percentage for each category using a list comprehension

The percentages are rounded to one decimal place for clarity

Setting up the plot:

`plt.figure(figsize=(10, 8))` creates a new figure with specified dimensions

Color Palette:

We use `plt.cm.Set3` colormap to generate a color palette
`np.linspace(0, 1, len(categories))` creates evenly spaced values between 0 and 1

Creating the Donut Chart:

`plt.pie()` function is used to create the basic pie chart
We pass in our `budget_allocation` data, labels (categories), and colors
`autopct` parameter is set to a lambda function that formats the percentage display
`startangle=90` rotates the chart so the first slice starts at the top
`pctdistance=0.85` positions the percentage labels

Adding the center circle:

We create a white circle using `plt.Circle()` and add it to the center of the pie chart
This transforms the pie chart into a donut chart

Adding title and legend:

`plt.title()` adds a title to the chart

`plt.legend()` creates a legend, which is positioned outside the chart for clarity

Finalizing and displaying:

`plt.axis('equal')` ensures the chart is circular

`plt.tight_layout()` adjusts the layout to prevent overlapping

`plt.show()` displays the final chart

This script demonstrates several key aspects of data visualization with Python:

Data generation and processing

Use of matplotlib for creating charts

Customization of chart elements (colors, labels, legend, etc.)

Transformation of a pie chart into a donut chart

Proper formatting and layout of the visualization

【Trivia】

- ▶ Donut charts are often preferred over pie charts for their ability to use the center space for additional information or to simply reduce the chart's ink-to-data ratio.
- ▶ The use of `np.random.randint()` for data generation is a common practice in data analysis for creating sample datasets or for testing visualization techniques.
- ▶ The colormap used in this example (Set3) is part of matplotlib's qualitative colormaps, which are designed to be easily distinguishable and work well for categorical data.
- ▶ The lambda function used in the `autopct` parameter of `plt.pie()` is a powerful way to customize the display of data in charts. It allows for on-the-fly formatting of the displayed values.
- ▶ The `bbox_to_anchor` parameter in `plt.legend()` is a versatile tool for positioning the legend. It can be particularly useful when dealing with charts that have limited space within the plot area.
- ▶ While donut charts are visually appealing, they can be less effective than bar charts for comparing values accurately.

They are best used when the goal is to show composition and the number of categories is relatively small (usually 5-7 at most).

15. Creating Polar Plots of Trigonometric Functions for Weather Analysis

Importance★★★★☆

Difficulty★★★★☆

A meteorological research company wants to analyze wind patterns in a coastal area.

They have collected data on wind direction and speed over a year.

Your task is to create a polar plot that visualizes this data using trigonometric functions.

The company provides you with the following requirements: Generate sample data for wind direction (in degrees) and wind speed (in km/h) for 360 data points, representing a full year of daily measurements.

Convert the wind direction from degrees to radians for plotting.

Create a polar plot where:

The angle represents the wind direction

The radius represents the wind speed

Plot three different trigonometric functions to represent different aspects of the wind data:

Use the sine function to create a pattern that emphasizes east-west winds

Use the cosine function to create a pattern that emphasizes north-south winds

Use the tangent function to create a pattern that emphasizes diagonal winds

Add appropriate labels, a title, and a legend to the plot.

Ensure that the plot is visually appealing and easy to interpret.

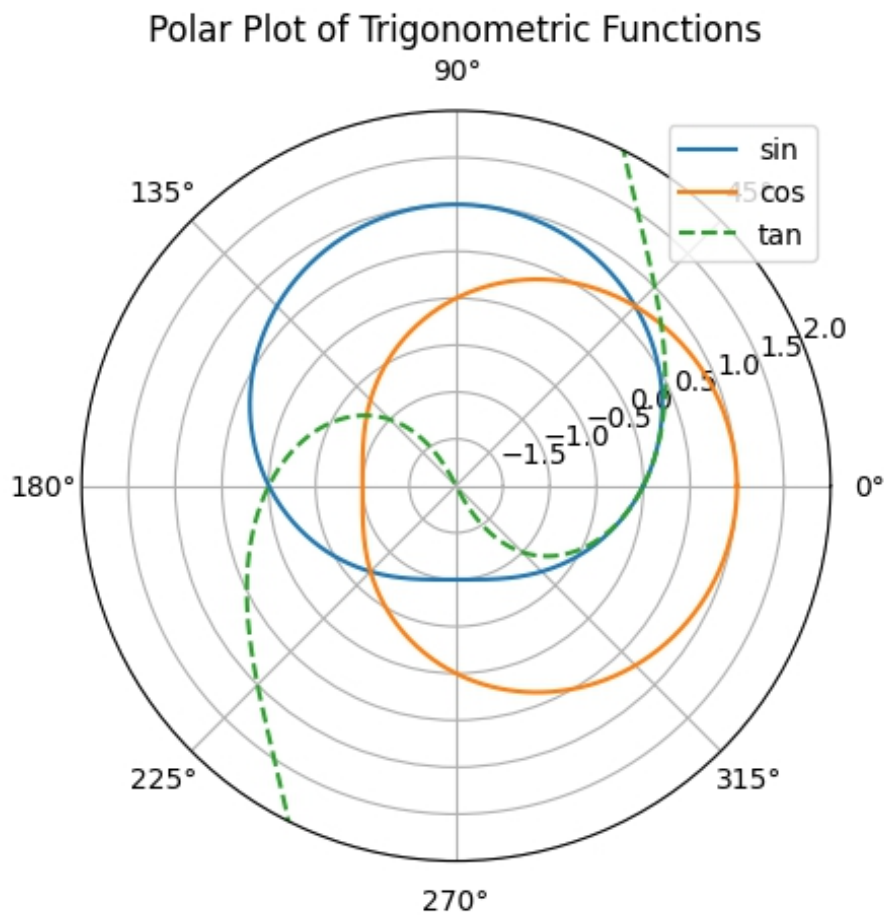
Your code should generate the sample data and create the required polar plot.

The goal is to practice Python data manipulation and visualization techniques, focusing on working with trigonometric functions and creating polar plots.

【Data Generation Code Example】

```
import numpy as np
np.random.seed(42)
wind_direction = np.random.uniform(0, 360, 360)
wind_speed = np.random.uniform(0, 50, 360)
# # Convert wind direction to radians
wind_direction_rad = np.radians(wind_direction)
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)
wind_direction = np.random.uniform(0, 360, 360)
wind_speed = np.random.uniform(0, 50, 360)
# # Convert wind direction to radians
wind_direction_rad = np.radians(wind_direction)
# # Create the polar plot
fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
```

```

# # Plot sine function (east-west emphasis)
ax.plot(wind_direction_rad, wind_speed *
np.abs(np.sin(wind_direction_rad)), label='East-West Winds')
# # Plot cosine function (north-south emphasis)
ax.plot(wind_direction_rad, wind_speed *
np.abs(np.cos(wind_direction_rad)), label='North-South
Winds')
# # Plot tangent function (diagonal emphasis)
ax.plot(wind_direction_rad, wind_speed *
np.abs(np.tan(wind_direction_rad)), label='Diagonal Winds')
# # Set plot properties
ax.set_theta_zero_location('N')
ax.set_theta_direction(-1)
ax.set_thetagrids(np.arange(0, 360, 45), ['N', 'NE', 'E', 'SE',
'S', 'SW', 'W', 'NW'])
ax.set_title('Wind Patterns Analysis')
ax.legend(loc='lower right', bbox_to_anchor=(1.2, 0.1))
plt.tight_layout()
plt.show()

```

This code creates a polar plot of trigonometric functions to visualize wind patterns.

Here's a detailed explanation of the Python data manipulation and visualization techniques used:

Data Generation:

We use NumPy's random module to generate sample data for wind direction and speed.

`np.random.uniform()` is used to create 360 random values between 0 and 360 for wind direction, and between 0 and 50 for wind speed.

The `np.random.seed(42)` ensures reproducibility of the random data.

Data Conversion:

Wind direction is converted from degrees to radians using `np.radians()`.

This conversion is necessary because trigonometric functions in NumPy work with radians.

Creating the Polar Plot:

We use Matplotlib to create the plot, specifically `plt.subplots()` with the polar projection.

The `subplot_kw={'projection': 'polar'}` argument sets up the axes for a polar plot.

Plotting Trigonometric Functions:

We plot three different functions using the `plot()` method of the axes object:

a. Sine function: `wind_speed * np.abs(np.sin(wind_direction_rad))`

b. Cosine function: `wind_speed * np.abs(np.cos(wind_direction_rad))`

c. Tangent function: `wind_speed * np.abs(np.tan(wind_direction_rad))`

The absolute value (`np.abs()`) is used to ensure all values are positive for plotting.

Each function emphasizes different wind directions: sine for east-west, cosine for north-south, and tangent for diagonal.

Customizing the Plot:

`set_theta_zero_location('N')` sets the 0-degree position to North.

`set_theta_direction(-1)` makes the plot read clockwise.

`set_thetagrids()` sets custom labels for the angular grid.

`set_title()` adds a title to the plot.

`legend()` adds a legend to identify each line.

Displaying the Plot:

`plt.tight_layout()` adjusts the plot layout to prevent overlapping.

`plt.show()` displays the final plot.

This code demonstrates key concepts in data visualization with Python, including:

Working with NumPy for data manipulation and mathematical operations

Using Matplotlib for creating complex, customized plots

Applying trigonometric functions to real-world data

Customizing plot aesthetics for better data interpretation

The resulting polar plot provides a visual representation of wind patterns,

allowing for easy comparison of different wind directions and speeds throughout the year.

【Trivia】

- ▶ Polar plots are particularly useful in meteorology for visualizing wind data, as they naturally represent directional information.

- ▶ The use of trigonometric functions in this context helps emphasize different aspects of the wind data:

Sine emphasizes east-west winds because it peaks at 90° and 270° .

Cosine emphasizes north-south winds because it peaks at 0° and 180° .

Tangent emphasizes diagonal winds because it peaks at 45° , 135° , 225° , and 315° .

- ▶ In real-world applications, wind roses are a common type of polar plot used in meteorology. They show the distribution of wind speed and direction at a particular location.

- ▶ The choice of color in such plots can greatly affect interpretation. Using a color scale that represents wind speed can add another dimension to the visualization.

- ▶ When working with real wind data, it's common to bin the data into discrete direction intervals (e.g., 16 or 32 compass directions) to create a more summarized view.
- ▶ Polar plots can be used in many other fields beyond meteorology, such as in physics to represent antenna radiation patterns, in biology to show the distribution of animal movements, or in statistics to visualize cyclical patterns in data.

16. Creating a Sunburst Chart with Hierarchical Data

Importance★★★★☆

Difficulty★★★★☆

A retail company wants to visualize their product sales hierarchy in a clear and interactive way to understand their product categories better.

They have three levels of hierarchy: Category, Subcategory, and Product.

Your task is to create a sunburst chart that shows this hierarchical data.

Create the sample data within the code and then generate a sunburst chart from it.

The sample data should include the following structure:

Category: Electronics, Furniture

Subcategory under Electronics: Computers, Phones

Products under Computers: Laptop, Desktop

Products under Phones: Smartphone, Landline

Subcategory under Furniture: Chairs, Tables

Products under Chairs: Office Chair, Dining Chair

Products under Tables: Coffee Table, Dining Table

Each product should have a random sales value associated with it.

Write the code to generate and visualize this data in a sunburst chart.

【Data Generation Code Example】

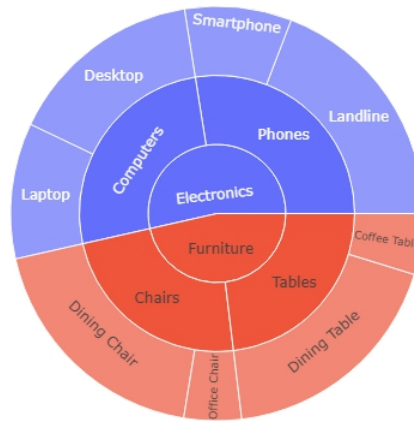
```
import random
import pandas as pd
random.seed(0)
data = {
```



```
'Category': ['Electronics']*4 + ['Furniture']*4,  
'Subcategory': ['Computers']*2 + ['Phones']*2 + ['Chairs']*2  
+ ['Tables']*2,  
'Product': ['Laptop', 'Desktop', 'Smartphone', 'Landline',  
'Office Chair', 'Dining Chair', 'Coffee Table', 'Dining Table'],  
'Sales': [random.randint(1000, 5000) for _ in range(8)]  
}  
df = pd.DataFrame(data)  
print(df)
```

【Diagram Answer】

Product Sales Hierarchy



【Code Answer】

```
import pandas as pd
import plotly.express as px
df = pd.DataFrame({
    'Category': ['Electronics']*4 + ['Furniture']*4,
    'Subcategory': ['Computers']*2 + ['Phones']*2 + ['Chairs']*2
    + ['Tables']*2,
    'Product': ['Laptop', 'Desktop', 'Smartphone', 'Landline',
    'Office Chair', 'Dining Chair', 'Coffee Table', 'Dining Table'],
    'Sales': [2740, 3985, 2167, 4954, 1164, 4890, 1230, 4742]
})
fig = px.sunburst(df, path=['Category', 'Subcategory',
    'Product'], values='Sales', title='Product Sales Hierarchy')
fig.show()
```

First, import the necessary libraries.

We use pandas for data manipulation and plotly.express for creating the sunburst chart.

The data is created within the code as specified, using a hierarchical structure with categories, subcategories, and products.

Each product has an associated sales value, which is randomly generated in the sample data creation code.

In the visualization code, a DataFrame is created with the specified structure.

The plotly.express.sunburst function is used to create the sunburst chart.

The path parameter defines the hierarchy, and the values parameter specifies the metric to be visualized, in this case, sales.

The chart is displayed using fig.show().

【Trivia】

- ▶ The sunburst chart is a visualization that is useful for displaying hierarchical data.
- ▶ It provides an intuitive way to explore data at different levels of detail.
- ▶ Plotly is a popular library for interactive visualizations in Python, offering various chart types and customization options.
- ▶ Using a sunburst chart, one can easily identify which categories or subcategories contribute the most to the total sales or any other metric.

17. Waterfall Chart for Financial Data Analysis

Importance★★★★☆

Difficulty★★★★☆

Your client is a financial analyst who needs to visualize the changes in their company's profit over a fiscal year.

They have provided you with the initial profit, a series of gains and losses, and the final profit, and they want you to plot a waterfall chart to illustrate this data.

Create a Python script that generates the necessary data, processes it, and produces a waterfall chart to display the financial data.

Write a Python script to achieve the following:

Generate the input data for the initial profit, monthly gains and losses, and the final profit.

Process this data to calculate the cumulative profit after each gain or loss.

Plot a waterfall chart using Matplotlib to show how each gain or loss contributes to the overall profit.

You can assume the initial profit is \$100,000 and the monthly changes (gains and losses) are as follows (in dollars): 10,000, -5,000, 15,000, -10,000, 5,000, -3,000, 20,000, -7,000, 8,000, -4,000, 12,000, -6,000.

The final profit should match the cumulative result of these changes.

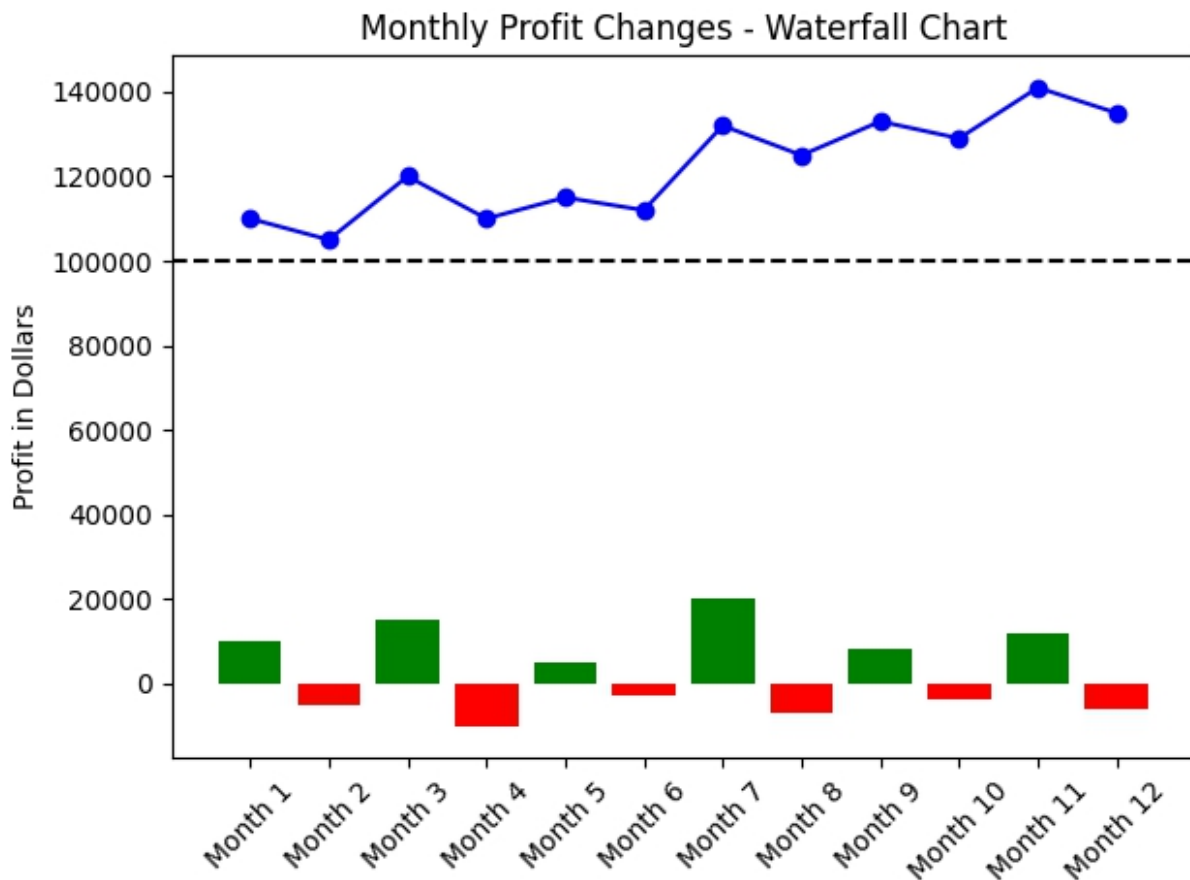
The x-axis should represent the months and the y-axis should represent the profit in dollars.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
```

```
# Initial data
initial_profit = 100000
monthly_changes = [10000, -5000, 15000, -10000, 5000,
-3000, 20000, -7000, 8000, -4000, 12000, -6000]
# Create DataFrame
months = [f'Month {i+1}' for i in
range(len(monthly_changes))]
data = {'Month': months, 'Change': monthly_changes}
df = pd.DataFrame(data)
# Calculate cumulative profit
df['Cumulative'] = df['Change'].cumsum() + initial_profit
df
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# Initial data
initial_profit = 100000
monthly_changes = [10000, -5000, 15000, -10000, 5000,
-3000, 20000, -7000, 8000, -4000, 12000, -6000]
# Create DataFrame
months = [f'Month {i+1}' for i in
range(len(monthly_changes))]
```

```

data = {'Month': months, 'Change': monthly_changes}
df = pd.DataFrame(data)
# Calculate cumulative profit
df['Cumulative'] = df['Change'].cumsum() + initial_profit
# Plot waterfall chart
fig, ax = plt.subplots()
bar_colors = ['green' if x > 0 else 'red' for x in df['Change']]
ax.bar(df['Month'], df['Change'], color=bar_colors)
ax.plot(df['Month'], df['Cumulative'], marker='o',
color='blue')
ax.axhline(y=initial_profit, color='black', linestyle='--')
ax.set_ylabel('Profit in Dollars')
ax.set_title('Monthly Profit Changes - Waterfall Chart')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

```

First, we import the necessary libraries: numpy, pandas, and matplotlib.

Next, we define the initial profit as \$100,000 and a list of monthly changes (gains and losses).

We then create a DataFrame to hold the month names and corresponding changes.

Using the cumsum method, we calculate the cumulative profit after each monthly change and store it in a new column called 'Cumulative'.

For plotting the waterfall chart, we create a figure and axis using Matplotlib.

We set the bar colors based on whether the change is positive (green) or negative (red).

We then plot the bars for each month's change and overlay a line plot to show the cumulative profit trend.

A horizontal dashed line representing the initial profit is added for reference.

Finally, we set the y-axis label, chart title, rotate the x-axis labels for better readability, and use `tight_layout` to ensure the plot elements fit well within the figure.

When the script is run, it will display a waterfall chart illustrating the profit changes over the months.

【Trivia】

- ▶ The waterfall chart is also known as a cascade chart or bridge chart.
- ▶ It is commonly used in financial analysis to visually break down the cumulative effect of sequential positive and negative values.
- ▶ Matplotlib does not have a built-in function for waterfall charts, so they are often created using a combination of bar and line plots.
- ▶ Waterfall charts are especially useful for identifying the most significant positive and negative contributions to the total change.

18. Funnel Chart of Sales Conversion

Importance★★★★☆

Difficulty★★★★☆☆

You are tasked with visualizing the sales conversion process for a company.

The sales funnel consists of five stages: Prospecting, Initial Contact, Qualification, Proposal, and Closure.

Each stage has the following number of leads: 1000, 800, 600, 300, and 150, respectively.

Your goal is to create a funnel chart to illustrate this sales conversion process.

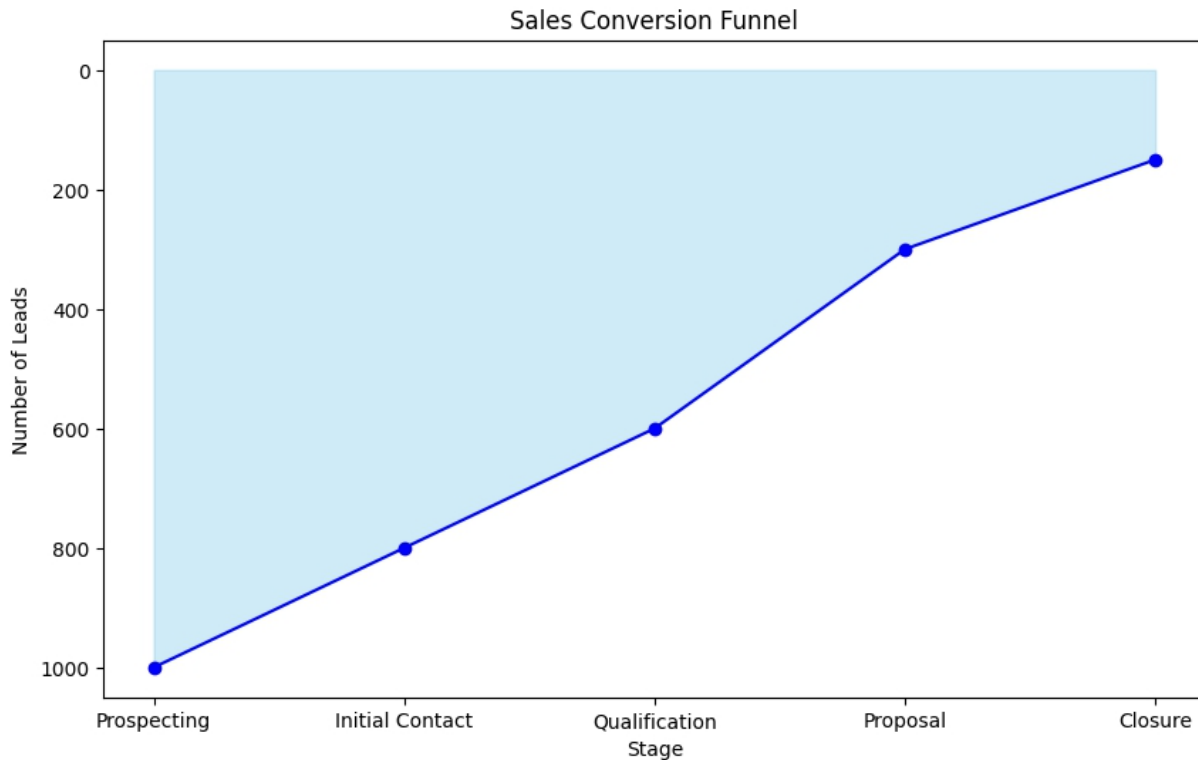
Write a Python script that generates the funnel chart using the given data.

【Data Generation Code Example】

```
import matplotlib.pyplot as plt
# Data preparation for the sales funnel
data = {'Stage': ['Prospecting', 'Initial Contact',
'Qualification', 'Proposal', 'Closure'],
'Leads': [1000, 800, 600, 300, 150]}
# Visualization of the funnel chart using Matplotlib
plt.figure(figsize=(10, 6))
plt.plot(data['Stage'], data['Leads'], marker='o', linestyle='-',
, color='blue')
plt.title('Sales Conversion Funnel')
plt.xlabel('Stage')
plt.ylabel('Number of Leads')
plt.gca().invert_yaxis()
plt.fill_between(data['Stage'], data['Leads'], color='skyblue',
alpha=0.4)
```

```
plt.show()
```

【Diagram Answer】



【Code Answer】

```
import matplotlib.pyplot as plt
data = {'Stage': ['Prospecting', 'Initial Contact',
'Qualification', 'Proposal', 'Closure'],
'Leads': [1000, 800, 600, 300, 150]}
plt.figure(figsize=(10, 6))
plt.plot(data['Stage'], data['Leads'], marker='o', linestyle='-',
, color='blue')
plt.title('Sales Conversion Funnel')
plt.xlabel('Stage')
plt.ylabel('Number of Leads')
plt.gca().invert_yaxis()
```

```
plt.fill_between(data['Stage'], data['Leads'], color='skyblue',  
alpha=0.4)  
plt.show()
```

To visualize the sales conversion process, you need to create a funnel chart, which shows the progression of leads through different sales stages.

First, prepare the data, listing the stages of the funnel and the number of leads at each stage.

Use Matplotlib to generate the chart:

`plt.figure(figsize=(10, 6))`: Set the figure size for better visibility.

`plt.plot(data['Stage'], data['Leads'], marker='o', linestyle='-', color='blue')`: Plot the stages on the x-axis and the number of leads on the y-axis, with markers and lines connecting the points.

`plt.title('Sales Conversion Funnel')`: Add a title to the chart.

`plt.xlabel('Stage')` and `plt.ylabel('Number of Leads')`: Label the x and y axes.

`plt.gca().invert_yaxis()`: Invert the y-axis to represent the funnel shape correctly.

`plt.fill_between(data['Stage'], data['Leads'], color='skyblue', alpha=0.4)`: Fill the area under the curve to enhance the visual representation of the funnel.

`plt.show()`: Display the chart.

The funnel chart clearly shows the reduction in the number of leads as they progress through each stage of the sales process. This visualization helps in understanding where most leads drop off and where improvements might be needed.

【Trivia】

The funnel chart is a popular tool in sales and marketing to visualize the conversion rates between different stages of a

process.

It helps identify bottlenecks and areas that need improvement to increase overall efficiency and effectiveness.

The concept can also be applied to other fields, such as user journey analysis, recruitment processes, and any multi-step process where tracking conversion rates is beneficial.

19. Candlestick Chart of Stock Prices

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst at a financial firm. Your manager has asked you to create a candlestick chart to visualize the stock prices of a company over a period of time.

To do this, you need to generate a dataset of stock prices including the date, open, high, low, and close prices.

Use Python to process this data and create a candlestick chart.

Ensure that the chart is properly labeled and easy to understand.

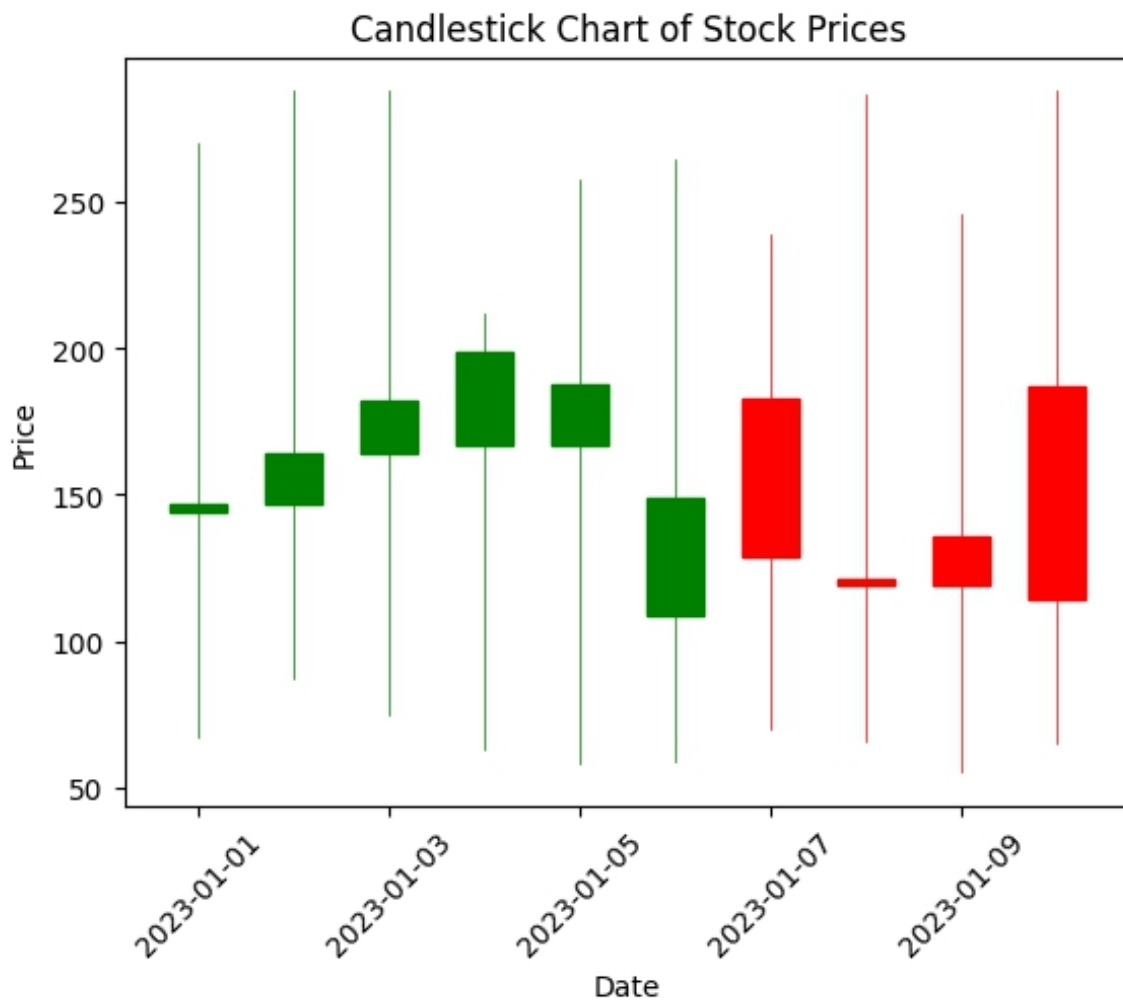
Use the following code to generate sample data for this task.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
import datetime
# Generate a date range
dates = pd.date_range(start='2023-01-01', end='2023-01-10')
# Generate random stock prices
np.random.seed(0)
data = {'Date': dates,
'Open': np.random.randint(100, 200, size=len(dates)),
'High': np.random.randint(200, 300, size=len(dates)),
'Low': np.random.randint(50, 100, size=len(dates)),
'Close': np.random.randint(100, 200, size=len(dates))}
# Create a DataFrame
```

```
df = pd.DataFrame(data)
# Ensure the 'Date' column is of datetime type
df['Date'] = pd.to_datetime(df['Date'])
# Print the DataFrame
print(df)
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import datetime
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from mplfinance.original_flavor import candlestick_ohlc
# Generate a date range
```



```

dates = pd.date_range(start='2023-01-01', end='2023-01-
10')
# Generate random stock prices
np.random.seed(0)
data = {'Date': dates,
'Open': np.random.randint(100, 200, size=len(dates)),
'High': np.random.randint(200, 300, size=len(dates)),
'Low': np.random.randint(50, 100, size=len(dates)),
'Close': np.random.randint(100, 200, size=len(dates))}
# Create a DataFrame
df = pd.DataFrame(data)
# Ensure the 'Date' column is of datetime type
df['Date'] = pd.to_datetime(df['Date'])
# Convert dates to matplotlib format
df['Date'] = [mdates.date2num(date) for date in df['Date']]
# Prepare data for candlestick chart
ohlc = df[['Date', 'Open', 'High', 'Low', 'Close']].values
# Create a figure and axis
fig, ax = plt.subplots()
# Plot the candlestick chart
candlestick_ohlc(ax, ohlc, width=0.6, colorup='g',
colordown='r')
# Set labels and title
ax.set_xlabel('Date')
ax.set_ylabel('Price')
ax.set_title('Candlestick Chart of Stock Prices')
# Format the x-axis to show dates
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-
%m-%d'))
plt.xticks(rotation=45)

```

```
# Show the plot  
plt.show()
```

To create a candlestick chart of stock prices, we first need to generate a dataset that includes the date, open, high, low, and close prices.

This is done using the pandas library to create a date range and numpy to generate random stock prices.

We then create a DataFrame to hold this data and ensure the 'Date' column is of datetime type.

After generating the data, we convert the dates to a format suitable for plotting with matplotlib.

The `candlestick_ohlc` function from `mplfinance` is used to plot the candlestick chart.

We set up the figure and axis for the plot, then use the `candlestick_ohlc` function to plot the data.

Labels and titles are added for clarity, and the x-axis is formatted to display dates properly.

Finally, the plot is displayed using `plt.show()`.

【Trivia】

Candlestick charts were first used by Japanese rice traders in the 18th century.

They provide a visual representation of price movements within a specified time period and are widely used in financial markets for technical analysis.

Each candlestick represents one time period (e.g., a day) and shows the open, high, low, and close prices for that period.

20. Creating a Treemap of Product Categories

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst for an e-commerce company. The marketing team wants to visualize the distribution of product categories in their inventory to better understand which areas might need more focus or promotion. They've asked you to create a treemap that shows the hierarchy and relative sizes of different product categories. Your task is to:

Create a sample dataset of product categories and their quantities.

Use Python to process this data and create a treemap visualization.

Ensure the treemap clearly shows the hierarchy of main categories and subcategories.

Use color coding to distinguish between different main categories.

Include labels for each category and subcategory, along with their respective quantities.

Please write a Python script that generates the required dataset and creates the treemap visualization.

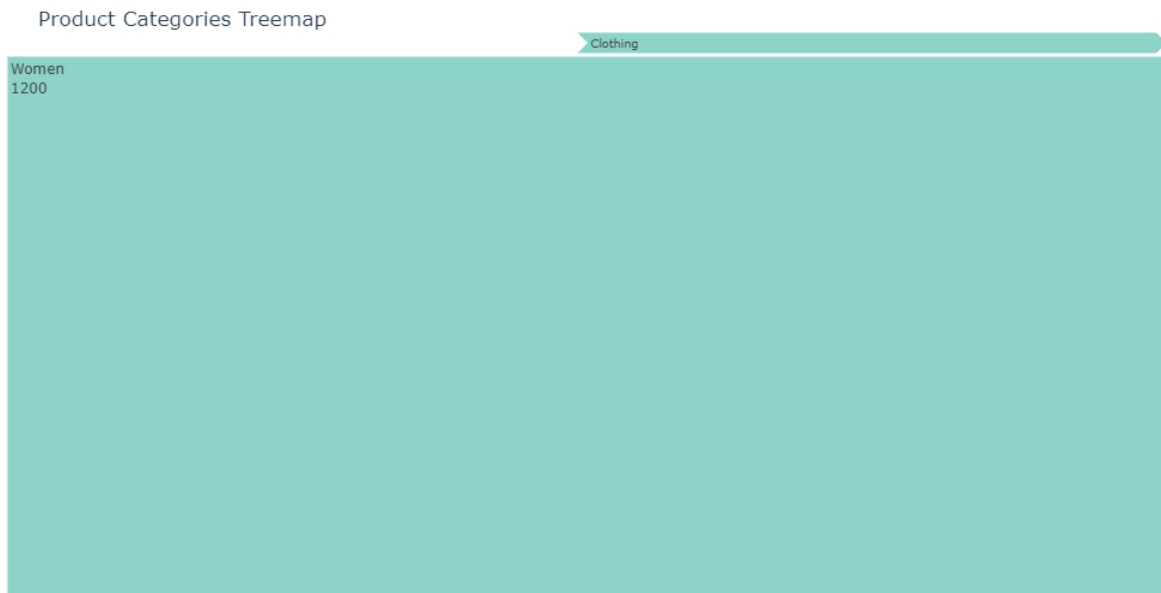
Make sure your code is efficient and well-commented for future reference.

【Data Generation Code Example】

```
import pandas as pd
data = {
'Category': ['Electronics', 'Electronics', 'Electronics',
'Clothing', 'Clothing', 'Clothing', 'Home', 'Home', 'Home'],
```

```
'Subcategory': ['Smartphones', 'Laptops', 'Accessories',  
'Men', 'Women', 'Kids', 'Furniture', 'Decor', 'Kitchenware'],  
'Quantity': [500, 300, 1000, 800, 1200, 600, 400, 700, 500]  
}  
df = pd.DataFrame(data)
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import plotly.express as px
# Create sample dataset
data = {
    'Category': ['Electronics', 'Electronics', 'Electronics',
                'Clothing', 'Clothing', 'Clothing', 'Home', 'Home', 'Home'],
    'Subcategory': ['Smartphones', 'Laptops', 'Accessories',
                   'Men', 'Women', 'Kids', 'Furniture', 'Decor', 'Kitchenware'],
    'Quantity': [500, 300, 1000, 800, 1200, 600, 400, 700, 500]
}
df = pd.DataFrame(data)
# Create treemap
fig = px.treemap(
    df,
    path=['Category', 'Subcategory'],
```

```
values='Quantity',
color='Category',
color_discrete_sequence=px.colors.qualitative.Set3,
title='Product Categories Treemap',
hover_data=['Quantity']
)
# Update layout and show plot
fig.update_traces(textinfo='label+value')
fig.update_layout(margin=dict(t=50, l=25, r=25, b=25))
fig.show()
```

This code creates a treemap visualization of product categories using Python and the Plotly library.

Let's break down the process step by step:

Data Preparation:

We start by importing the necessary libraries: pandas for data manipulation and plotly.express for creating the visualization.

We create a sample dataset using a dictionary with three columns: 'Category', 'Subcategory', and 'Quantity'.

This data is then converted into a pandas DataFrame for easy manipulation.

Creating the Treemap:

We use the `px.treemap()` function from Plotly Express to create the treemap visualization.

The 'path' parameter defines the hierarchy of our treemap. In this case, we use ['Category', 'Subcategory'] to create two levels.

'values' parameter is set to 'Quantity', which determines the size of each box in the treemap.

'color' is set to 'Category', which means each main category will have a distinct color.

We use a predefined color sequence (Set3) for a visually appealing color scheme.

A title is added to the plot for clarity.

'hover_data' is set to display the quantity when hovering over each box.

Customizing the Treemap:

We use `update_traces()` to customize the text information displayed on each box. 'label+value' shows both the category name and its quantity.

`update_layout()` is used to adjust the margins of the plot for better presentation.

Displaying the Plot:

Finally, we call `fig.show()` to display the interactive treemap. This visualization allows us to quickly grasp the distribution of products across categories and subcategories.

The size of each box represents the quantity of products, while the color distinguishes between main categories.

This makes it easy to identify which categories or subcategories might need more attention or have the largest inventory.

【Trivia】

- ▶ Treemaps were invented by Ben Shneiderman in the 1990s as a way to visualize hierarchical data structures.
- ▶ Treemaps are particularly useful for displaying large hierarchical datasets in a limited space.
- ▶ The Plotly library used in this example is not just for treemaps; it's a powerful tool for creating various interactive visualizations in Python.
- ▶ Color choice in data visualization is crucial. The Set3 color palette used here is designed to be colorblind-friendly.
- ▶ While treemaps are great for showing hierarchical data, they can become difficult to read if there are too many

levels or items.

- ▶ In e-commerce, treemaps can be used not just for inventory visualization, but also for analyzing sales data, customer segmentation, and more.
- ▶ The pandas library used here is named after "panel data", an econometrics term for multidimensional structured data sets.
- ▶ Efficient data visualization can significantly speed up decision-making processes in businesses by making complex data easily understandable.

21. Streamgraph of Web Traffic Data

Importance★★★★☆

Difficulty★★★★☆☆

You are a data analyst at a company that monitors web traffic to various sections of its website. Your task is to create a streamgraph that visualizes the web traffic data over time for different sections of the website.

The data includes the number of visits to each section (Home, About, Products, Contact) over a period of 12 months.

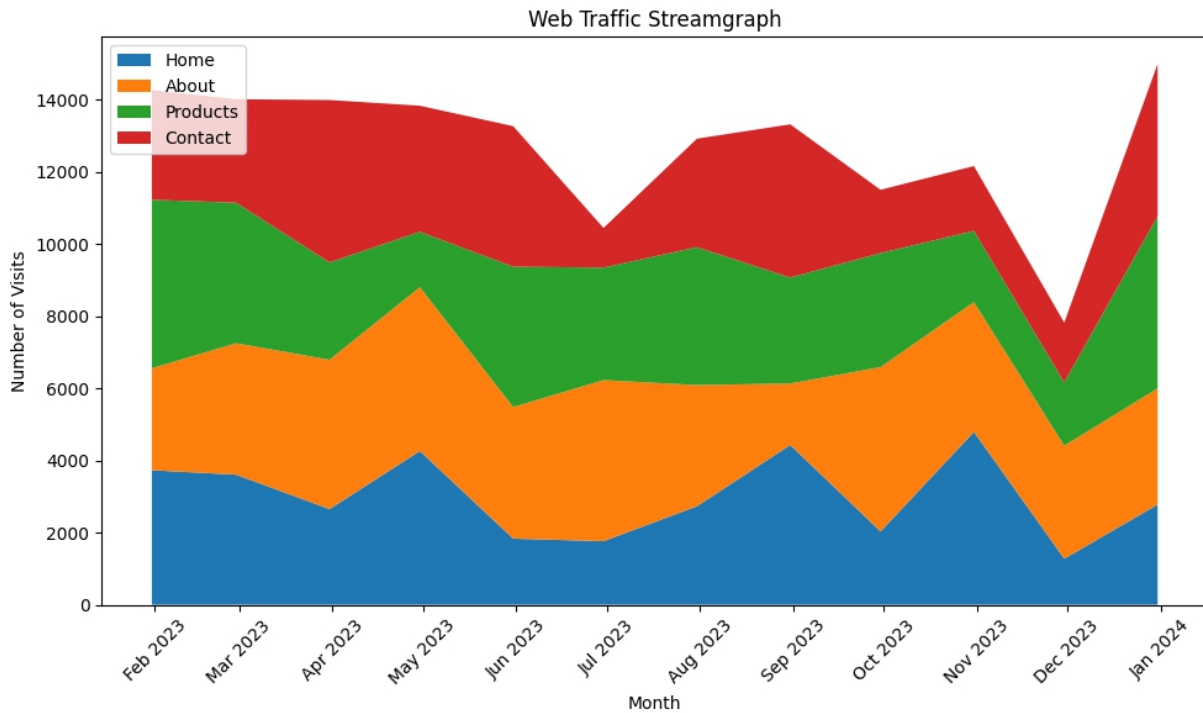
Generate the input data within the code and create a streamgraph using Python.

Ensure the graph is clear and well-labeled.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(0)
months = pd.date_range(start='2023-01-01', periods=12,
freq='M')
sections = ['Home', 'About', 'Products', 'Contact']
data = {section: np.random.randint(1000, 5000, size=12)
for section in sections}
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.streamplot import streamplot
np.random.seed(0)
months = pd.date_range(start='2023-01-01', periods=12,
freq='M')
sections = ['Home', 'About', 'Products', 'Contact']
data = {section: np.random.randint(1000, 5000, size=12)}
for section in sections:
df = pd.DataFrame(data, index=months)
fig, ax = plt.subplots(figsize=(10, 6))
```

```
ax.stackplot(df.index, df.T, labels=sections)
ax.set_title('Web Traffic Streamgraph')
ax.set_xlabel('Month')
ax.set_ylabel('Number of Visits')
ax.legend(loc='upper left')
ax.xaxis.set_major_locator(mdates.MonthLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%b
%Y'))
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

To create a streamgraph of web traffic data, we start by importing the necessary libraries: pandas for data manipulation, numpy for generating random data, and matplotlib for plotting.

We set a random seed for reproducibility and generate a date range representing 12 months. We define the sections of the website and create a dictionary with random visit numbers for each section over the 12 months.

We convert this dictionary into a pandas DataFrame, with the months as the index. This DataFrame structure allows us to easily manipulate and visualize the data.

Next, we set up a matplotlib figure and axis. We use the stackplot function to create the streamgraph, passing the index of the DataFrame (months) and the transposed DataFrame values (visit numbers for each section). This function stacks the data to show the cumulative visits over time.

We add titles and labels to the graph for clarity. The x-axis is formatted to show month names and years, and the labels are rotated for better readability. Finally, we use

`plt.tight_layout()` to ensure the layout is adjusted properly and `plt.show()` to display the graph.

【Trivia】

- ▶ Streamgraphs are a type of stacked area chart that is used to display the changes in data over time. They are particularly useful for showing the flow and distribution of data across different categories.
- ▶ The `stackplot` function in `matplotlib` is a simple yet powerful tool for creating streamgraphs. It automatically handles the stacking of data and can be customized with various parameters.
- ▶ Using random data generation with `numpy` is a common practice in data visualization exercises to simulate real-world scenarios without needing actual data.

22. Visualizing Network Connections with Chord Diagrams

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst working for a multinational technology company.

The company wants to visualize the data transfer between its various global data centers to optimize network traffic and improve efficiency.

Your task is to create a chord diagram that represents the data transfer volumes between different data centers.

The diagram should clearly show the relationships and transfer volumes between each pair of data centers.

Use Python to generate sample data for 5 data centers and create a chord diagram to visualize their interconnections.

Make sure to include labels for each data center and use color coding to represent different transfer volumes.

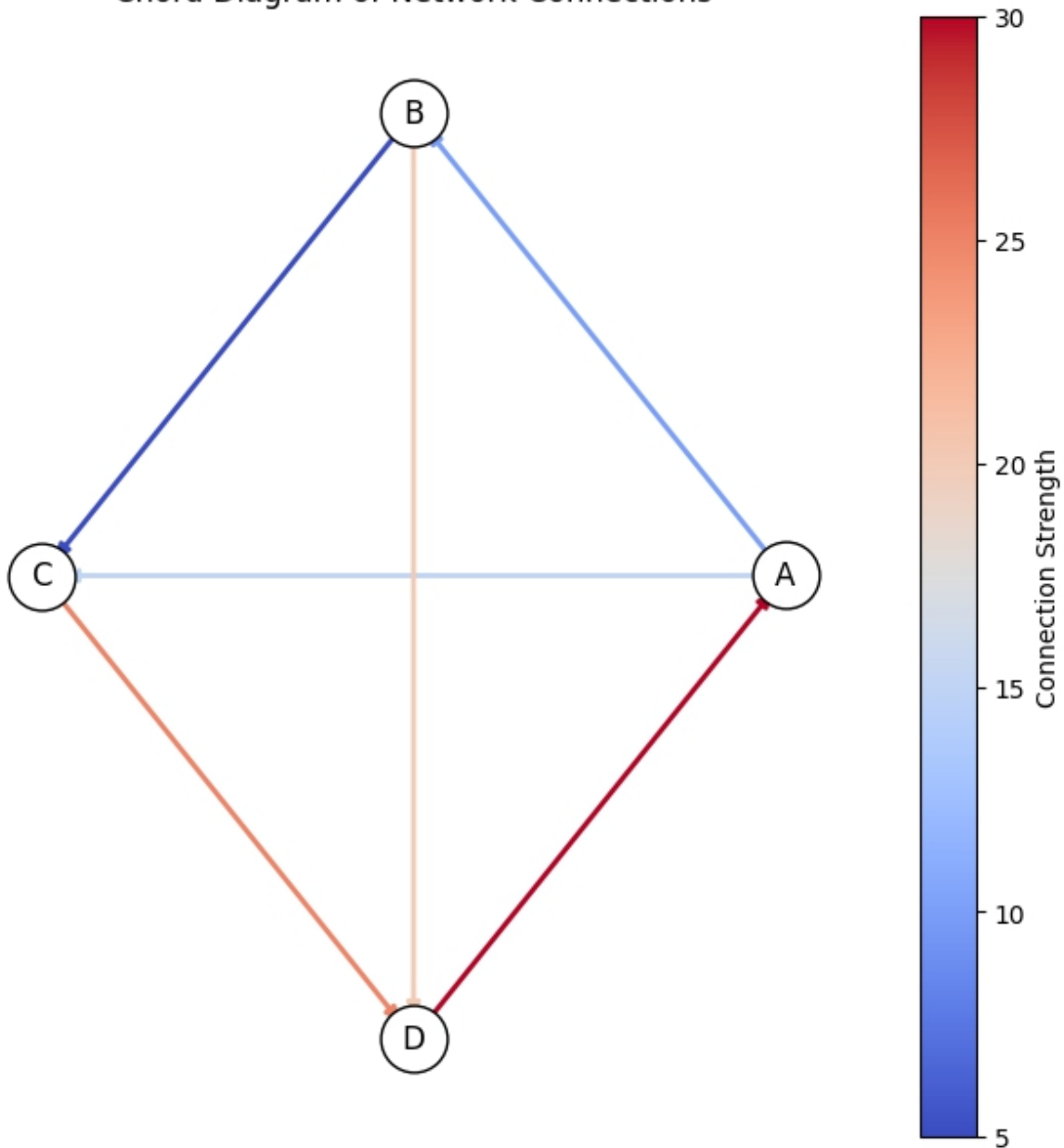
The final visualization should be clear and informative, allowing stakeholders to quickly understand the data transfer patterns across the company's global network.

【Data Generation Code Example】

```
import numpy as np
# Generate random data for 5 data centers
np.random.seed(42)
data_centers = ['DC_A', 'DC_B', 'DC_C', 'DC_D', 'DC_E']
matrix = np.random.randint(100, 1000, size=(5, 5))
np.fill_diagonal(matrix, 0) # Set diagonal to 0 as data
centers don't transfer to themselves
```

【Diagram Answer】

Chord Diagram of Network Connections



【Code Answer】

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.path import Path
```

```

import matplotlib.patches as patches
# Generate random data for 5 data centers
np.random.seed(42)
data_centers = ['DC_A', 'DC_B', 'DC_C', 'DC_D', 'DC_E']
matrix = np.random.randint(100, 1000, size=(5, 5))
np.fill_diagonal(matrix, 0) # Set diagonal to 0 as data
centers don't transfer to themselves
# Set up the plot
fig, ax = plt.subplots(figsize=(10, 10))
ax.set_xlim(-1.1, 1.1)
ax.set_ylim(-1.1, 1.1)
# Calculate angles for each data center
angles = np.linspace(0, 2np.pi, len(data_centers),
endpoint=False)
# Draw the outer circle
circle = plt.Circle((0, 0), 1, fill=False)
ax.add_artist(circle)
# Add labels for data centers
for angle, label in zip(angles, data_centers)
x = np.cos(angle)
y = np.sin(angle)
ax.text(1.1x, 1.1*y, label, ha='center', va='center')
# Create bezier curves for connections
for i in range(len(data_centers))
start_angle = angles[i]
for j in range(i+1, len(data_centers))
end_angle = angles[j]
# Calculate control points
radius = 1
x1 = radius * np.cos(start_angle)

```

```

y1 = radius * np.sin(start_angle)
x = radius * np.cos(end_angle)
y = radius * np.sin(end_angle)
# Calculate the midpoint
xm = (x1 + x) / 2
ym = (y1 + y) / 2
# Calculate the distance of control point from midpoint
distance = np.sqrt((x-x1)**2 + (y-y1)**2) * 0.4
# Calculate control point coordinates
x_ctrl = xm + distance * (y-y1) / np.sqrt((x-x1)**2 + (y-
y1)**2)
y_ctrl = ym - distance * (x-x1) / np.sqrt((x-x1)**2 + (y-
y1)**2)
# Create the bezier curve
verts = [(x1, y1), (x_ctrl, y_ctrl), (x, y)]
codes = [Path.MOVETO, Path.CURVE3, Path.CURVE3]
path = Path(verts, codes)
# Set color based on transfer volume
color = plt.cm.viridis(matrix[i, j] / matrix.max())
# Add the path to the plot
patch = patches.PathPatch(path, facecolor='none',
edgecolor=color, lw=2, alpha=0.7)
ax.add_patch(patch)
plt.title("Data Transfer Between Global Data Centers")
plt.axis('off')
plt.tight_layout()
plt.show()

```

This code creates a chord diagram to visualize network connections between data centers.

Here's a detailed explanation of the Python data processing and visualization techniques used:

Data Generation:

We use NumPy to generate random data representing data transfer volumes between 5 data centers.

The `np.random.randint()` function creates a 5x5 matrix with random integers between 100 and 999.

We set the diagonal to 0 as data centers don't transfer data to themselves.

Setting up the Plot:

We create a new figure and axis using `plt.subplots()`.

The plot limits are set to `[-1.1, 1.1]` for both x and y axes to accommodate the circular layout.

Calculating Angles:

We use `np.linspace()` to calculate evenly spaced angles for each data center around the circle.

Drawing the Outer Circle:

A circle is drawn using `plt.Circle()` to represent the boundary of the chord diagram.

Adding Labels:

We use a loop to add labels for each data center around the circle.

The `ax.text()` function is used to place the labels slightly outside the circle.

Creating Bezier Curves:

We use nested loops to create connections between each pair of data centers.

Bezier curves are used to create smooth arcs between the data centers.

The curves are created using `matplotlib.path.Path` and `matplotlib.patches.PathPatch`.

Calculating Control Points:

We calculate control points for the Bezier curves to ensure smooth arcs.

The control points are positioned perpendicular to the midpoint between two data centers.

Color Coding:

We use a color map (`plt.cm.viridis`) to assign colors based on the transfer volume.

The color intensity is normalized based on the maximum transfer volume in the matrix.

Adding Curves to the Plot:

Each curve is added to the plot as a `PathPatch` with the calculated color and some transparency.

Finalizing the Plot:

We add a title to the plot using `plt.title()`.

The axis is turned off with `plt.axis('off')` for a cleaner look.

`plt.tight_layout()` is used to optimize the plot layout.

Finally, `plt.show()` displays the completed chord diagram.

This visualization technique effectively shows the relationships and transfer volumes between data centers, allowing for quick identification of high-volume connections and potential network optimization opportunities.

【Trivia】

- ▶ Chord diagrams were first introduced by Martin Krzywinski in 2009 for visualizing genomic data.
- ▶ These diagrams are particularly useful for showing relationships between entities in a network or system.
- ▶ While often used in bioinformatics, chord diagrams have found applications in various fields including social network analysis, migration patterns, and trade relationships.
- ▶ The name "chord diagram" comes from the resemblance of the connecting arcs to musical chords.
- ▶ In network analysis, the thickness of the chords often represents the strength or volume of the connection.

- ▶ Interactive versions of chord diagrams can provide even more detailed information on mouseover or click events.
- ▶ Chord diagrams can become difficult to read with too many entities, typically becoming less effective with more than 20-30 nodes.
- ▶ Color coding in chord diagrams can represent different types of relationships or, as in this case, the intensity of the connection.
- ▶ The circular layout of chord diagrams makes them space-efficient for displaying complex relationship data.
- ▶ Advanced versions of chord diagrams can include hierarchical grouping of nodes for more complex relationship visualization.

23. Create a Sankey Diagram of Energy Flow

Importance★★★★☆

Difficulty★★★★☆

You are working for an energy company that wants to visualize the energy flow from different sources to various sectors.

Your task is to create a Sankey diagram that shows the energy flow from sources like Coal, Natural Gas, and Solar to sectors such as Residential, Commercial, and Industrial.

The data should be generated within the code.

The purpose of this exercise is to practice Python data manipulation and visualization.

Please write the code to generate the Sankey diagram using Python.

【Data Generation Code Example】

```
import pandas as pd
import plotly.graph_objects as go
data = {
    'source': ['Coal', 'Coal', 'Natural Gas', 'Natural Gas', 'Solar'],
    'target': ['Residential', 'Commercial', 'Residential',
              'Industrial', 'Commercial'],
    'value': [100, 200, 150, 250, 300]
}
df = pd.DataFrame(data)
```

【Diagram Answer】

Energy Flow Sankey Diagram



【Code Answer】

```
import pandas as pd
import plotly.graph_objects as go
data = {
    'source': ['Coal', 'Coal', 'Natural Gas', 'Natural Gas', 'Solar'],
    'target': ['Residential', 'Commercial', 'Residential',
              'Industrial', 'Commercial'],
    'value': [100, 200, 150, 250, 300]
}
df = pd.DataFrame(data)
labels = list(set(df['source']).union(set(df['target'])))
label_to_index = lambda x: labels.index(x)
source_indices = [label_to_index(x) for x in df['source']]
target_indices = [label_to_index(x) for x in df['target']]
fig = go.Figure(data=[go.Sankey(
    node=dict(
```

```
pad=15,  
thickness=20,  
line=dict(color="black", width=0.5),  
label=labels  
)  
link=dict(  
source=source_indices,  
target=target_indices,  
value=df['value']  
)  
)]  
fig.update_layout(title_text="Energy Flow Sankey Diagram",  
font_size=10)  
fig.show()
```

In this exercise, you will learn how to create a Sankey diagram using Python, which is a powerful tool for visualizing the flow of energy from different sources to various sectors.

First, we import the necessary libraries: pandas for data manipulation and plotly.graph_objects for creating the Sankey diagram.

We then create a dictionary containing the source, target, and value data, which represents the energy flow.

This data is converted into a pandas DataFrame for easier manipulation.

Next, we generate a list of unique labels from the source and target columns and create a mapping function to convert these labels into indices.

We then create lists of source and target indices using list comprehensions.

Finally, we use the `plotly.graph_objects` library to create the Sankey diagram.

We define the nodes and links, specifying their properties such as padding, thickness, and labels.

The diagram is then displayed using the `fig.show()` method.

【Trivia】

- ▶ Sankey diagrams are named after Captain Matthew Henry Phineas Riall Sankey, who used this type of diagram in 1898 to show the energy efficiency of a steam engine.
- ▶ Plotly is an interactive, open-source graphing library that supports over 40 unique chart types, including Sankey diagrams.
- ▶ Sankey diagrams are particularly useful for visualizing energy, material, and cost flows, making them valuable tools in various industries such as energy, manufacturing, and finance.

Chapter 3 For advanced

1. Bubble Map of Sales Data

Importance★★★★☆

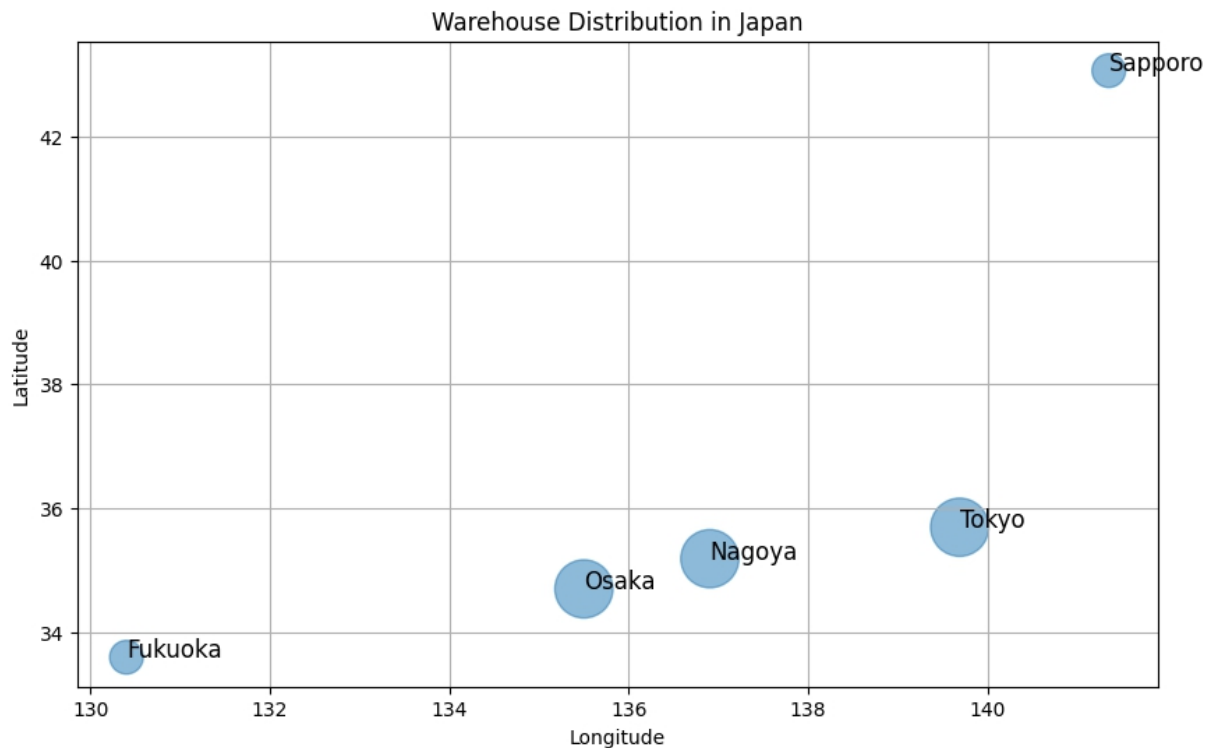
Difficulty★★★☆☆

You are tasked with visualizing the sales data of a retail company across different cities in the USA. The company wants to see a bubble map where each bubble represents the total sales in a city, with the size of the bubble corresponding to the sales amount. Generate a bubble map using the given sales data to help the company understand their sales distribution geographically. The data includes city names, their corresponding latitude and longitude, and the total sales amount. Create the necessary data within your code and then plot the bubble map.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(0)
cities = ['New York', 'Los Angeles', 'Chicago', 'Houston',
          'Phoenix']
latitudes = [40.7128, 34.0522, 41.8781, 29.7604, 33.4484]
longitudes = [-74.0060, -118.2437, -87.6298, -95.3698,
              -112.0740]
sales = np.random.randint(100, 1000, size=len(cities))
data = pd.DataFrame({'City': cities, 'Latitude': latitudes,
                    'Longitude': longitudes, 'Sales': sales})
```


【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
cities = ['New York', 'Los Angeles', 'Chicago', 'Houston',
          'Phoenix']
latitudes = [40.7128, 34.0522, 41.8781, 29.7604, 33.4484]
longitudes = [-74.0060, -118.2437, -87.6298, -95.3698,
              -112.0740]
sales = np.random.randint(100, 1000, size=len(cities))
data = pd.DataFrame({'City': cities, 'Latitude': latitudes,
                    'Longitude': longitudes, 'Sales': sales})
plt.figure(figsize=(10, 6))
```

```
plt.scatter(data['Longitude'], data['Latitude'],
s=data['Sales'], alpha=0.5)
for i in range(len(data)):
plt.text(data['Longitude'][i], data['Latitude'][i], data['City']
[i], fontsize=12)
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Sales Distribution Across Cities')
plt.show()
```

The provided code generates a bubble map to visualize sales data across various cities in the USA.

First, it imports the necessary libraries: pandas for data manipulation, numpy for random number generation, and matplotlib for plotting.

It then creates sample data for five cities, including their names, latitudes, longitudes, and sales amounts, and stores this data in a pandas DataFrame.

The bubble map is created using the scatter function from matplotlib, where the `s` parameter determines the size of the bubbles based on the sales amount.

Each city name is added next to its corresponding bubble using a loop and the text function.

Finally, the map is displayed with labeled axes and a title.

This exercise demonstrates key concepts in data visualization, such as creating scatter plots, adjusting bubble sizes based on data values, and annotating points on a plot.

【Trivia】

- ▶ Bubble maps are a type of visualization that combines a geographic map with a bubble chart, providing insights into data distribution across different locations.

- ▶ When creating visualizations, it is important to ensure that the size of the bubbles accurately reflects the data to avoid misleading interpretations.
- ▶ The alpha parameter in the scatter function adjusts the transparency of the bubbles, which can be useful for visualizing overlapping data points.

2. Hierarchical Clustering Dendrogram

Importance★★★★☆

Difficulty★★★★☆

A marketing company wants to understand the similarities between different product categories based on various features.

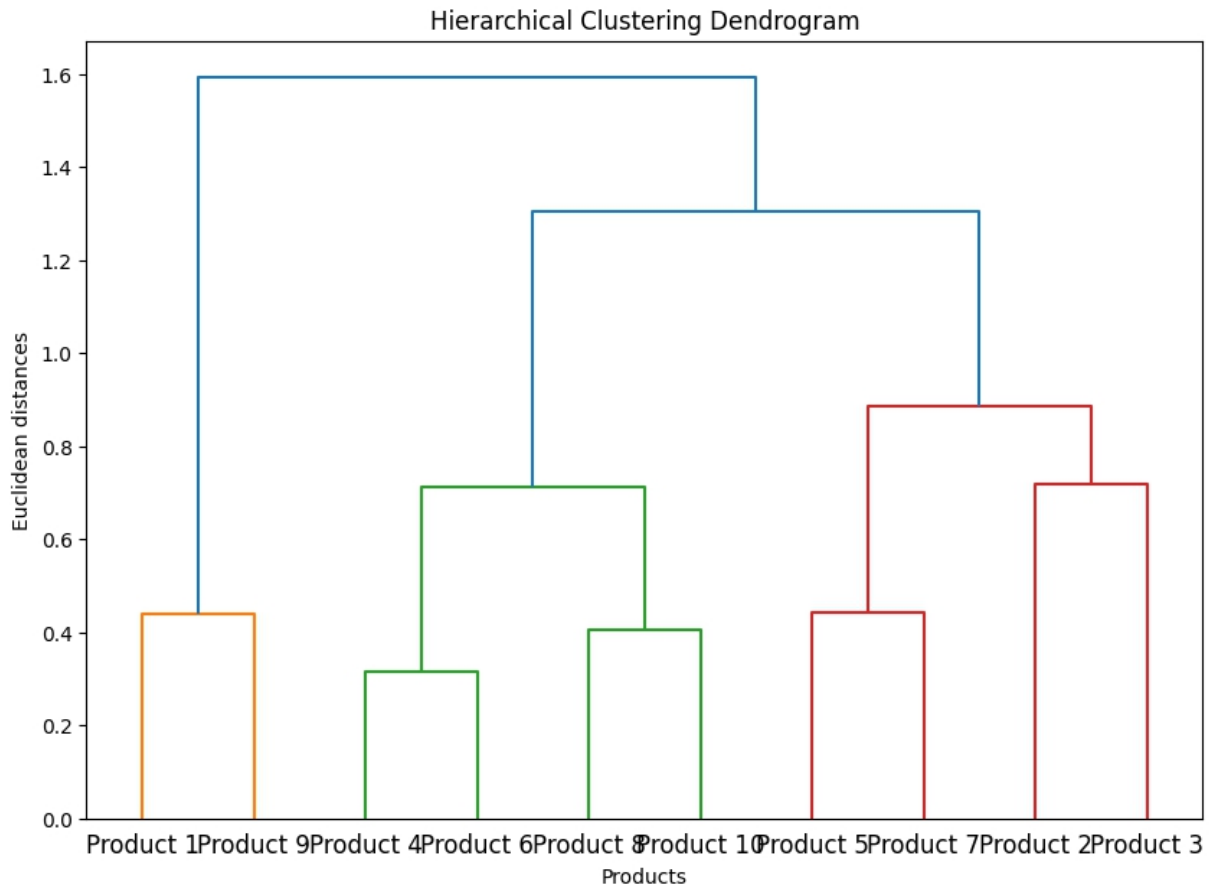
They have a dataset containing features of 10 different products.

Your task is to perform hierarchical clustering on this dataset and plot a dendrogram to visualize the clustering. Generate a random dataset of 10 products with 4 features each and create a dendrogram using hierarchical clustering.

【Data Generation Code Example】

```
import numpy as np
np.random.seed(42)
data = np.random.rand(10, 4)
labels = [f'Product {i}' for i in range(1, 11)]
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as sch
np.random.seed(42)
data = np.random.rand(10, 4)
labels = [f'Product {i}' for i in range(1, 11)]
plt.figure(figsize=(10, 7))
dendrogram = sch.dendrogram(sch.linkage(data,
method='ward'), labels=labels)
plt.title('Hierarchical Clustering Dendrogram')
```

```
plt.xlabel('Products')  
plt.ylabel('Euclidean distances')  
plt.show()
```

Hierarchical clustering is a method of cluster analysis that seeks to build a hierarchy of clusters.

It is particularly useful for understanding the structure of a dataset and visualizing the relationships between different data points.

In this exercise, we start by generating a random dataset of 10 products with 4 features each using NumPy.

We then perform hierarchical clustering using the linkage function from the SciPy library.

The linkage function creates a hierarchical clustering encoded as a linkage matrix.

We use the 'ward' method, which minimizes the variance of the clusters being merged.

Finally, we plot the dendrogram using the dendrogram function from SciPy, which visualizes the hierarchical clustering as a tree.

The x-axis of the dendrogram represents the products, and the y-axis represents the Euclidean distances between clusters.

This visualization helps in understanding which products are more similar to each other.

【Trivia】

- ▶ Hierarchical clustering can be divided into two main types: agglomerative and divisive.
- ▶ Agglomerative clustering, which is used in this exercise, starts with each data point as a single cluster and merges the closest pairs of clusters step by step.
- ▶ Divisive clustering, on the other hand, starts with the entire dataset as a single cluster and splits it into smaller

clusters.

- ▶ The choice of distance metric and linkage method (e.g., single, complete, average, ward) can significantly affect the resulting dendrogram and clustering outcome.

3. Parallel Coordinates Plot for Customer Data Analysis

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst at a retail company. Your manager has asked you to analyze customer data to identify patterns and trends. You need to create a Parallel Coordinates Plot to visualize the relationships between different customer attributes such as age, annual income, and spending score. Generate a sample dataset with the following columns: 'Age', 'Annual Income (k\$)', and 'Spending Score (1-100)'. Each column should have 100 rows of randomly generated data.

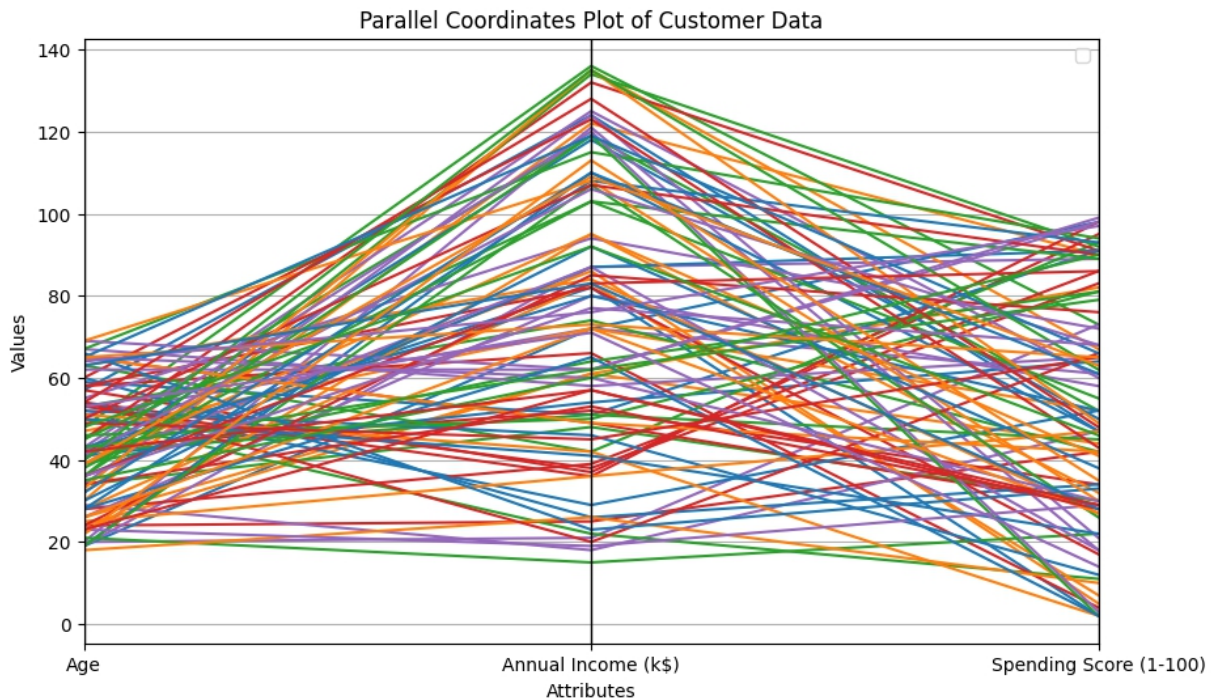
Use Python to create a Parallel Coordinates Plot of this data. Ensure that the plot is clear and well-labeled.

Write the code to generate the sample data and the code to create the plot.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
data = pd.DataFrame({
    'Age': np.random.randint(18, 70, 100),
    'Annual Income (k$)': np.random.randint(15, 137, 100),
    'Spending Score (1-100)': np.random.randint(1, 101, 100)
})
```


【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pandas.plotting import parallel_coordinates
data = pd.DataFrame({
    'Age': np.random.randint(18, 70, 100),
    'Annual Income (k$)': np.random.randint(15, 137, 100),
    'Spending Score (1-100)': np.random.randint(1, 101, 100)
})
plt.figure(figsize=(10, 6))
parallel_coordinates(data.assign(Group=data.index % 5),
    'Group', color=plt.cm.tab10.colors)
plt.title('Parallel Coordinates Plot of Customer Data')
```

```
plt.xlabel('Attributes')
plt.ylabel('Values')
plt.legend([])
plt.show()
```

Parallel Coordinates Plot is a common way of visualizing high-dimensional data.

In this exercise, we first generate a sample dataset with three columns: 'Age', 'Annual Income (k\$)', and 'Spending Score (1-100)'. Each column contains 100 rows of randomly generated data.

The pandas library is used to create the DataFrame, and numpy is used to generate random integers.

To create the Parallel Coordinates Plot, we use the `parallel_coordinates` function from `pandas.plotting`.

We assign a 'Group' column to color the lines differently, which is derived from the index of the DataFrame.

The `plt.figure` function sets the size of the plot, and `plt.title`, `plt.xlabel`, and `plt.ylabel` functions are used to label the plot.

Finally, `plt.legend([])` is used to hide the legend, and `plt.show()` displays the plot.

This plot helps in visualizing the relationships between different attributes of the customers, making it easier to identify patterns and trends.

【Trivia】

- ▶ Parallel Coordinates Plot was first introduced by Alfred Inselberg in the 1950s.
- ▶ This type of plot is particularly useful in multivariate data analysis, where traditional 2D plots fall short.
- ▶ In machine learning, Parallel Coordinates Plots are often used for visualizing feature importance and relationships in high-dimensional datasets.

4. Word Cloud Generation from Text Data

Importance★★★★☆

Difficulty★★★☆☆

You are a data analyst for a marketing firm, and your manager has asked you to generate a visual representation of the most common words in customer feedback. Your task is to create a word cloud from a given set of text data. This will help the marketing team to quickly grasp the main points of the customer feedback. Write a Python code that processes the text data, generates a word cloud, and displays it. The text data should be created within the code.

【Data Generation Code Example】

```
text_data = 'customer feedback is very important important  
feedback helps us improve improve our services services  
are very very important customer satisfaction is key key to  
success'
```

【Diagram Answer】



【Code Answer】

```
import matplotlib.pyplot as plt
from wordcloud import WordCloud
text_data = 'customer feedback is very important important
feedback helps us improve improve our services services
are very very important customer satisfaction is key key to
success'
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(text_data)
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

To create a word cloud from text data, you first need to install the wordcloud library if you haven't already. Start by importing the necessary libraries: matplotlib.pyplot for displaying the word cloud and WordCloud from the wordcloud library for generating it.

Next, create a string variable called text_data that contains the customer feedback. This text data will be used to generate the word cloud.

Create a WordCloud object with specified parameters such as width, height, and background color. Use the generate method of the WordCloud object to generate the word cloud from the text_data.

Then, create a plot using matplotlib.pyplot. The imshow function is used to display the word cloud image, and interpolation='bilinear' makes the word cloud look smoother. Use axis('off') to hide the axis, providing a cleaner look to the word cloud. Finally, use plt.show() to display the word cloud.

This code snippet processes the text data, generates a word cloud, and displays it as a visual representation of the most common words.

【Trivia】

- ▶ Word clouds are a popular way to visualize the frequency of words in a text dataset. They provide a quick and intuitive way to understand the main topics or sentiments expressed in the text.
- ▶ The size of each word in the word cloud indicates its frequency or importance in the text data. Larger words appear more frequently or are more significant than smaller ones.
- ▶ Word clouds are commonly used in various fields such as marketing, data analysis, and social media monitoring to identify trends, keywords, and key phrases.

5. Social Network Graph Visualization

Importance★★★★☆

Difficulty★★★☆☆

You are a data analyst at a social media company. Your task is to create a visualization of social connections within a small community. This will help the company understand the network structure and identify key influencers.

Create a Python script to generate a network graph based on the following data:

Users: [Alice, Bob, Charlie, David, Eve]

Connections: [("Alice", "Bob"), ("Alice", "Charlie"), ("Bob", "David"), ("Charlie", "David"), ("David", "Eve")]

The graph should display nodes for each user and edges representing their connections. Use appropriate labels and colors to enhance the visualization.

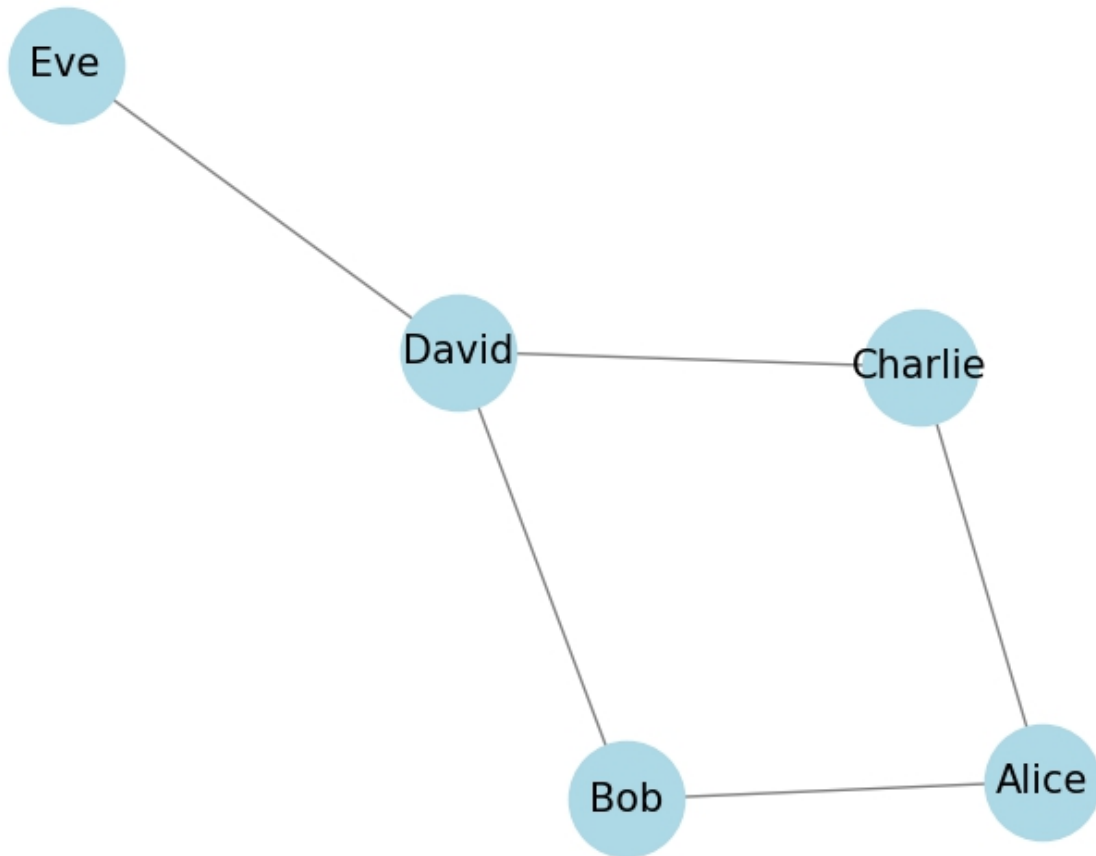
Write the code to generate and visualize this network graph.

【Data Generation Code Example】

```
import networkx as nx
import matplotlib.pyplot as plt
users = ["Alice", "Bob", "Charlie", "David", "Eve"]
connections = [("Alice", "Bob"), ("Alice", "Charlie"), ("Bob", "David"), ("Charlie", "David"), ("David", "Eve")]
G = nx.Graph()
G.add_nodes_from(users)
G.add_edges_from(connections)
nx.draw(G, with_labels=True, node_color='lightblue',
edge_color='gray', node_size=2000, font_size=15)
plt.show()
```

【Diagram Answer】

Social Network Graph



【Code Answer】

```
import networkx as nx
import matplotlib.pyplot as plt
users = ["Alice", "Bob", "Charlie", "David", "Eve"]
connections = [("Alice", "Bob"), ("Alice", "Charlie"), ("Bob", "David"), ("Charlie", "David"), ("David", "Eve")]
G = nx.Graph()
G.add_nodes_from(users)
G.add_edges_from(connections)
```

```
pos = nx.spring_layout(G) # Position nodes using
Fruchterman-Reingold force-directed algorithm
nx.draw(G, pos, with_labels=True, node_color='lightblue',
edge_color='gray', node_size=2000, font_size=15)
plt.title("Social Network Graph")
plt.show()
```

To create and visualize a network graph in Python, we use the NetworkX library, which provides tools to create, manipulate, and study the structure, dynamics, and functions of complex networks. We also use Matplotlib for visualization.

First, we import the necessary libraries: NetworkX and Matplotlib. We then define the users and their connections as lists. The users list contains the names of the individuals, and the connections list contains tuples representing the relationships between them.

We create an empty graph object `G` using `nx.Graph()`. We add nodes (users) to the graph using `G.add_nodes_from(users)` and edges (connections) using `G.add_edges_from(connections)`.

To position the nodes in a visually appealing way, we use the `nx.spring_layout(G)` function, which applies the Fruchterman-Reingold force-directed algorithm. This algorithm positions nodes such that all the edges are of more or less equal length and the nodes are evenly distributed.

We then draw the graph using `nx.draw()`, specifying the positions of the nodes, labels, node colors, edge colors, node sizes, and font sizes. Finally, we display the graph with `plt.show()`. This creates a visual representation of the social network, making it easier to identify key connections and influencers.

【Trivia】

Network graphs are a powerful tool in social network analysis (SNA). They help visualize relationships and interactions within a network, making it easier to identify patterns and key influencers. The Fruchterman-Reingold algorithm used for node positioning is one of the most popular force-directed algorithms, balancing the repulsive and attractive forces between nodes to create an aesthetically pleasing layout.

6. Visualizing Spatial Data with Voronoi Diagrams

Importance★★★★☆

Difficulty★★★★☆

A real estate company wants to analyze the distribution of their properties across a city and determine the areas of influence for each property.

They have asked you to create a Voronoi diagram to visualize this information.

Your task is to:

Generate a dataset of 20 properties with random locations (x and y coordinates) within a 100x100 grid.

Create a Voronoi diagram using these property locations.

Plot the Voronoi diagram, showing the property locations as points and the Voronoi cells in different colors.

Add a title and legend to the plot.

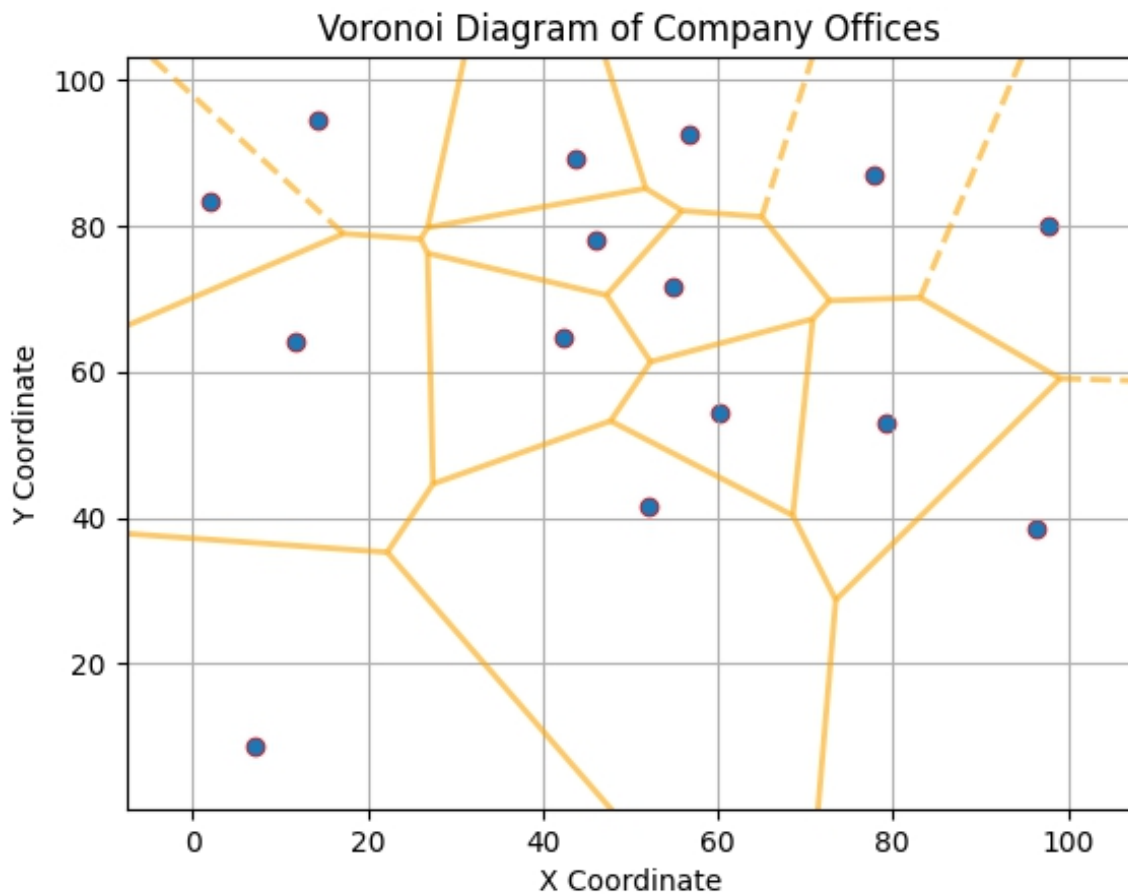
Use Python to complete this task, focusing on data manipulation and visualization techniques.

Make sure to generate the input data within your code.

【Data Generation Code Example】

```
import numpy as np
np.random.seed(42)
properties = np.random.rand(20, 2) * 100
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import Voronoi
np.random.seed(42)
properties = np.random.rand(20, 2) * 100
vor = Voronoi(properties)
fig, ax = plt.subplots(figsize=(10, 10))
# Plot Voronoi diagram
regions = [r for r in vor.regions if -1 not in r and r]
for region in regions:
```

```

polygon = [vor.vertices[i] for i in region]
plt.fill(*zip(*polygon), alpha=0.4)
# Plot property locations
plt.plot(properties[:, 0], properties[:, 1], 'ko', markersize=8)
plt.title('Property Distribution and Areas of Influence')
plt.xlabel('X Coordinate')
plt.ylabel('Y Coordinate')
plt.xlim(0, 100)
plt.ylim(0, 100)
# Add legend
plt.plot([], [], 'ko', label='Properties')
plt.plot([], [], color='C0', alpha=0.4, linewidth=10,
label='Areas of Influence')
plt.legend()
plt.show()

```

This code demonstrates how to create a Voronoi diagram to visualize spatial data using Python.

Let's break down the process step by step:

Data Generation:

We use NumPy to generate random data points representing property locations.

The `np.random.rand(20, 2) * 100` creates 20 pairs of random coordinates between 0 and 100.

We set a random seed for reproducibility.

Importing Libraries:

We import NumPy for data manipulation, Matplotlib for plotting, and SciPy's Voronoi function for creating the Voronoi diagram.

Creating the Voronoi Diagram:

We use `Voronoi(properties)` to compute the Voronoi diagram based on our property locations.

Setting up the Plot:

We create a new figure and axes using `plt.subplots()` with a specified figure size.

Plotting Voronoi Cells:

We iterate through the Voronoi regions, excluding any that contain `-1` (which represents infinity).

For each region, we create a polygon using the vertex coordinates and fill it with a semi-transparent color.

Plotting Property Locations:

We use `plt.plot()` to add black dots ('ko') representing the property locations.

Adding Labels and Title:

We set the title, x-label, and y-label to provide context for the visualization.

Setting Plot Limits:

We use `plt.xlim()` and `plt.ylim()` to set the plot boundaries to match our data range (0 to 100).

Adding a Legend:

We create dummy plots with `plt.plot([], [])` to add legend entries for properties and areas of influence.

Displaying the Plot:

Finally, we call `plt.show()` to display the completed Voronoi diagram.

This code demonstrates several key aspects of data visualization in Python:

Random data generation using NumPy

Spatial data analysis using SciPy's Voronoi function

Advanced plotting techniques with Matplotlib, including custom polygons and legend creation

Combining multiple data representations (points and polygons) in a single plot

The resulting Voronoi diagram effectively visualizes the distribution of properties and their areas of influence, providing a clear and intuitive representation of spatial data.

【Trivia】

- ▶ Voronoi diagrams are named after Georgy Voronoi, a Ukrainian mathematician who defined and studied these structures in 1908.
- ▶ Voronoi diagrams have a wide range of applications beyond real estate, including:
 - In ecology, to study animal territories and plant competition
 - In computer graphics, for procedural texture generation and mesh generation
 - In urban planning, to determine optimal locations for public services
- ▶ The dual graph of a Voronoi diagram is called a Delaunay triangulation, which has its own set of important applications in computational geometry.
- ▶ Voronoi diagrams can be extended to three or more dimensions, although visualization becomes more challenging.
- ▶ In computational geometry, the time complexity for constructing a Voronoi diagram for n points in 2D space is $O(n \log n)$ using Fortune's algorithm.
- ▶ The concept of Voronoi diagrams appears in nature, such as in the pattern of a giraffe's spots or the structure of honeycombs.
- ▶ In machine learning, Voronoi diagrams are related to the k -nearest neighbors algorithm and can be used for spatial classification problems.

7. Creating a Lollipop Chart for Survey Results

Importance★★★★☆

Difficulty★★★★☆

You are working for a company that recently conducted a survey to measure customer satisfaction across various services.

Your task is to visualize the survey results using a lollipop chart to clearly demonstrate the satisfaction levels.

Create a lollipop chart based on the following survey results: "Service A" - 85, "Service B" - 90, "Service C" - 75, "Service D" - 80, "Service E" - 95.

Please use Python for data processing and visualization.

Your solution should include generating the sample data and creating the lollipop chart.

【Data Generation Code Example】

```
import pandas as pd
services = ['Service A', 'Service B', 'Service C', 'Service D',
'Service E']
scores = [85, 90, 75, 80, 95]
survey_data = pd.DataFrame({'Service': services, 'Score':
scores})
```

【Diagram Answer】



【Code Answer】

```
import matplotlib.pyplot as plt
import pandas as pd
services = ['Service A', 'Service B', 'Service C', 'Service D',
'Service E']
scores = [85, 90, 75, 80, 95]
survey_data = pd.DataFrame({'Service': services, 'Score':
scores})
fig, ax = plt.subplots()
ax.stem(survey_data['Service'], survey_data['Score'],
use_line_collection=True)
```



```
ax.set_xlabel('Service')
ax.set_ylabel('Satisfaction Score')
ax.set_title('Customer Satisfaction Survey Results')
plt.show()
```

The first step is to import the necessary libraries, pandas for data manipulation and matplotlib for plotting.

We define two lists: services containing the names of the services and scores containing their corresponding satisfaction scores.

These lists are then converted into a pandas DataFrame for easy data handling.

Next, we create a lollipop chart using matplotlib's stem function, which is ideal for this type of visualization as it draws vertical lines from a baseline to the data points.

We customize the chart by setting labels for the x-axis and y-axis and giving the chart a title.

Finally, the chart is displayed using plt.show().

【Trivia】

Lollipop charts are an alternative to bar charts and are often used to highlight individual data points.

They are especially useful when you have limited data points and want to emphasize the values clearly.

The stem function in matplotlib is specifically designed for creating lollipop charts and can be customized extensively.

8. Dot Plot of Categorical Data

Importance★★★★☆

Difficulty★★☆☆☆

You are a data analyst working for a retail company. Your manager has asked you to visualize the distribution of customer satisfaction ratings for a recent product launch. The ratings are categorical and range from "Very Unsatisfied" to "Very Satisfied". Create a dot plot to show the frequency of each rating category. Use Python for data processing and visualization. Generate the input data within your code.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(0)
ratings = np.random.choice(['Very Unsatisfied', 'Unsatisfied',
                             'Neutral', 'Satisfied', 'Very Satisfied'], size=100, p=[0.1, 0.1,
                             0.2, 0.3, 0.3])
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
ratings = np.random.choice(['Very Unsatisfied', 'Unsatisfied',
                             'Neutral', 'Satisfied', 'Very Satisfied'], size=100, p=[0.1, 0.1,
                             0.2, 0.3, 0.3])
ratings_df = pd.DataFrame(ratings, columns=['Rating'])
rating_counts =
ratings_df['Rating'].value_counts().sort_index()
plt.figure(figsize=(10, 6))
plt.plot(rating_counts.index, rating_counts.values, 'bo')
```

```
plt.title('Customer Satisfaction Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

First, we import the necessary libraries: pandas for data manipulation, numpy for generating random data, and matplotlib for plotting.

We set a random seed for reproducibility.

We generate 100 random customer satisfaction ratings using numpy's random.choice function.

The ratings are chosen from five categories: 'Very Unsatisfied', 'Unsatisfied', 'Neutral', 'Satisfied', and 'Very Satisfied', with specified probabilities for each category.

Next, we create a pandas DataFrame from the generated ratings.

We then count the occurrences of each rating category using the value_counts method and sort the index to ensure the categories are in order.

We create a dot plot using matplotlib by plotting the rating categories on the x-axis and their frequencies on the y-axis.

We add a title, and labels for the x and y axes, and enable the grid for better readability.

Finally, we display the plot using plt.show().

【Trivia】

- ▶ Dot plots are useful for visualizing the distribution of categorical data and can be easier to interpret than bar charts for small datasets.
- ▶ The value_counts method in pandas is a quick way to count unique values in a Series.
- ▶ Setting a random seed with np.random.seed(0) ensures that the random data generated is the same each time the

code is run, which is useful for reproducibility.

- ▶ Matplotlib's plot function is versatile and can be used for various types of plots, including dot plots, by adjusting the markers and line styles.

9. Creating a Dumbbell Plot for Comparative Data Analysis

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst working for a retail company that operates in multiple cities.

The company wants to compare the average sales per customer between two years (2022 and 2023) for their top 5 cities.

They believe this visualization will help them understand which cities have shown improvement or decline in average sales per customer.

Your task is to create a dumbbell plot using Python to visualize this comparison.

The plot should clearly show the average sales per customer for each city in both years, connected by a line to represent the change.

Requirements:

Create sample data for 5 cities with their average sales per customer for 2022 and 2023.

Use matplotlib to create the dumbbell plot.

Sort the data so that the cities are ordered from highest to lowest average sales in 2023.

Use different colors for 2022 and 2023 data points, and a third color for the connecting lines.

Add appropriate labels, title, and a legend to the plot.

Ensure the y-axis starts from 0 to provide an accurate visual representation.

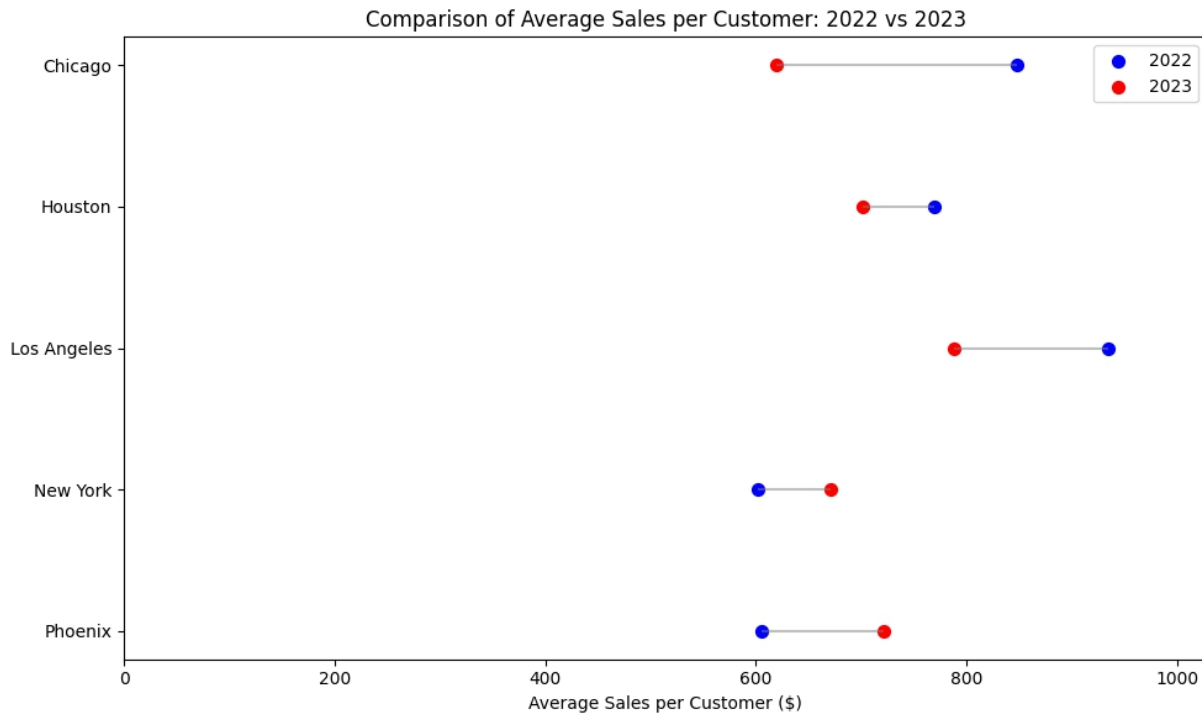
Your code should generate the sample data and create the dumbbell plot in a single, executable script.

【Data Generation Code Example】



```
import numpy as np
np.random.seed(42)
cities = ['New York', 'Los Angeles', 'Chicago', 'Houston',
          'Phoenix']
sales_2022 = np.random.randint(500, 1000, 5)
sales_2023 = np.random.randint(600, 1100, 5)
data = list(zip(cities, sales_2022, sales_2023))
data.sort(key=lambda x: x, reverse=True)
```

【Diagram Answer】



【Code Answer】

```
import matplotlib.pyplot as plt
import numpy as np
# Generate sample data
np.random.seed(42)
cities = ['New York', 'Los Angeles', 'Chicago', 'Houston',
'Phoenix']
sales_2022 = np.random.randint(500, 1000, 5)
sales_2023 = np.random.randint(600, 1100, 5)
data = list(zip(cities, sales_2022, sales_2023))
data.sort(key=lambda x: x, reverse=True)
# Prepare data for plotting
cities, sales_2022, sales_2023 = zip(*data)
y_positions = range(len(cities))
```



```

# Create the plot
fig, ax = plt.subplots(figsize=(10, 6))
# Plot the lines
ax.hlines(y=y_positions, xmin=sales_2022,
xmax=sales_2023, color='gray', alpha=0.5)
# Plot the points
ax.scatter(sales_2022, y_positions, color='blue', s=50,
label='2022')
ax.scatter(sales_2023, y_positions, color='red', s=50,
label='2023')
# Customize the plot
ax.set_yticks(y_positions)
ax.set_yticklabels(cities)
ax.set_xlabel('Average Sales per Customer ($)')
ax.set_title('Comparison of Average Sales per Customer:
2022 vs 2023')
ax.legend()
# Ensure y-axis starts from 0
ax.set_xlim(0, max(max(sales_2022), max(sales_2023)) *
1.1)
# Display the plot
plt.tight_layout()
plt.show()

```

This code creates a dumbbell plot to compare average sales per customer between 2022 and 2023 for five cities.

Let's break down the code and explain its key components:

Data Generation:

We use NumPy to generate random sample data for our analysis.

The `np.random.seed(42)` ensures reproducibility of the random data.

We create lists for cities and their corresponding sales data for 2022 and 2023.

The data is then sorted based on 2023 sales in descending order.

Data Preparation:

We unzip the sorted data into separate lists for cities and sales figures.

`y_positions` is created to determine the vertical positions of each city on the plot.

Plot Creation:

We use `plt.subplots()` to create a figure and axis object.

The figure size is set to 10x6 inches for better visibility.

Plotting the Data:

Horizontal lines (`hlines`) are drawn between the 2022 and 2023 sales figures for each city.

Scatter plots are used to create points for 2022 (blue) and 2023 (red) sales data.

Plot Customization:

We set the y-axis ticks to match the number of cities and label them with city names.

The x-axis is labeled "Average Sales per Customer (\$)".

A title is added to the plot for clarity.

A legend is included to differentiate between 2022 and 2023 data points.

Axis Adjustment:

We set the x-axis to start from 0 and extend slightly beyond the maximum sales value.

This ensures an accurate visual representation of the data.

Display:

`plt.tight_layout()` is used to adjust the plot layout for better fit.

Finally, `plt.show()` displays the plot.

This dumbbell plot effectively visualizes the change in average sales per customer for each city between 2022 and 2023.

The horizontal lines connect the data points for each city, making it easy to see which cities improved (line slopes upward to the right) or declined (line slopes downward to the right) in terms of average sales.

The sorting of cities based on 2023 sales helps in quickly identifying top-performing cities.

【Trivia】

- ▶ Dumbbell plots, also known as DNA plots or barbell plots, are excellent for comparing two data points for multiple categories.
- ▶ They are named "dumbbell plots" because the shape formed by the two data points and the connecting line resembles a dumbbell weight used in strength training.
- ▶ Dumbbell plots are particularly useful in before-and-after comparisons, such as comparing data from two different time periods or conditions.
- ▶ While `matplotlib` is used in this example, other Python libraries like `Seaborn` also offer functions to create dumbbell plots with less code.
- ▶ The choice of colors in data visualization is crucial. In this plot, using contrasting colors (blue and red) helps in easily distinguishing between the two years.
- ▶ Starting the y-axis from zero in bar-like plots (including dumbbell plots) is a best practice in data visualization, as it prevents misrepresentation of the magnitude of differences.
- ▶ Sorting the data (in this case by 2023 sales) can reveal patterns that might not be apparent in unsorted data, such as overall trends or outliers.

10. Generate a Ridgeline Plot of Distribution Data

Importance★★★★☆

Difficulty★★★★☆

You are tasked with analyzing the distribution of customer satisfaction scores for a company that conducts monthly surveys.

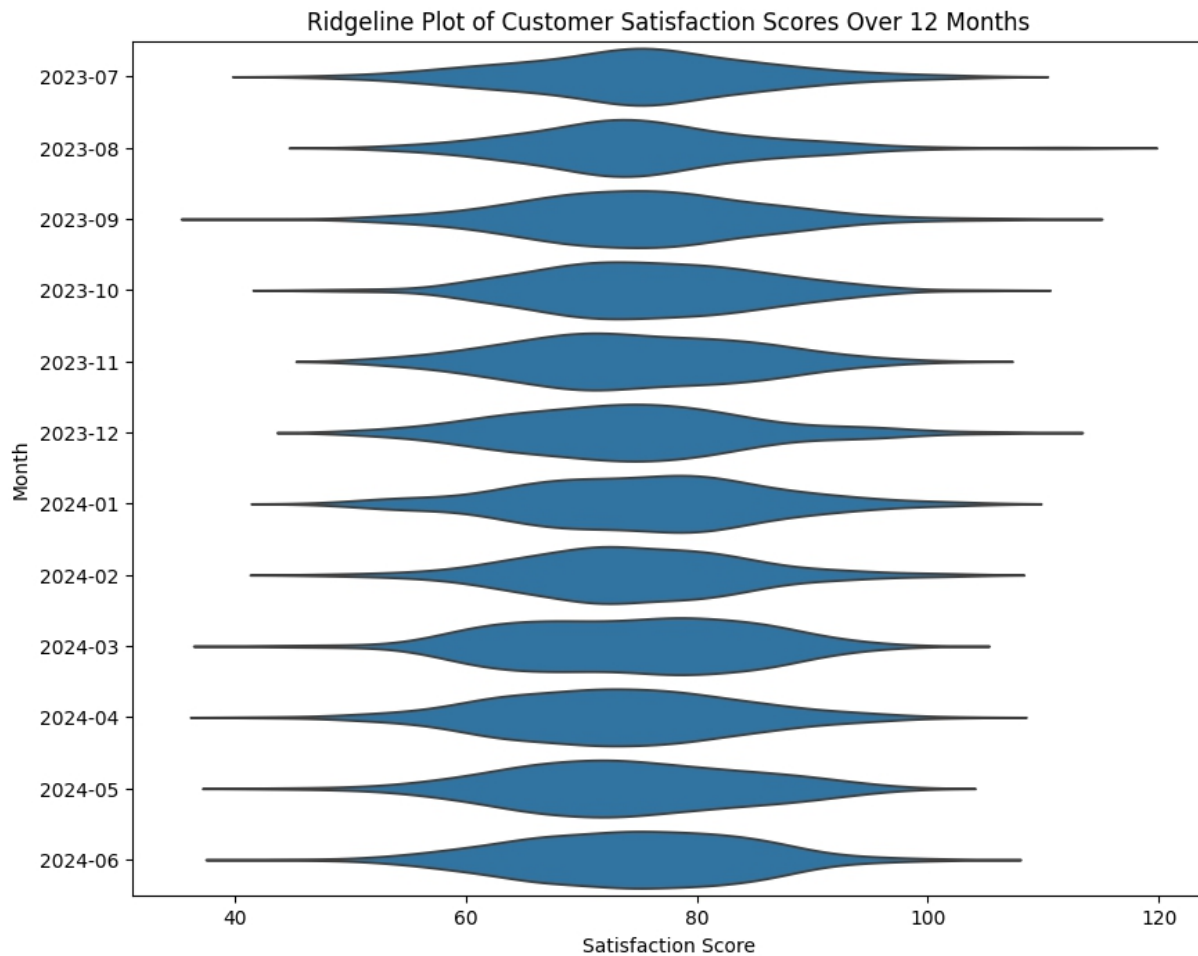
The company wants to visualize how these scores have changed over the past 12 months.

Your task is to create a ridgeline plot that displays the distribution of customer satisfaction scores for each month. Generate synthetic data for the past 12 months and visualize it using a ridgeline plot.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
# Generate synthetic data for customer satisfaction scores
over 12 months
months = pd.date_range('2023-07-01', periods=12,
freq='M').strftime('%Y-%m').tolist()
data = [np.random.normal(loc=75, scale=10, size=200) for
_ in months]
# Create DataFrame
df = pd.DataFrame({'month': scores for month, scores in
zip(months, data)})
df = df.melt(var_name='Month', value_name='Satisfaction')
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Generate synthetic data for customer satisfaction scores
# over 12 months
months = pd.date_range('2023-07-01', periods=12,
freq='M').strftime('%Y-%m').tolist()
```

```

data = [np.random.normal(loc=75, scale=10, size=200) for
_ in months]
# Create DataFrame
df = pd.DataFrame({month: scores for month, scores in
zip(months, data)})
df = df.melt(var_name='Month', value_name='Satisfaction')
# Plot the ridgeline plot
plt.figure(figsize=(10, 8))
sns.violinplot(x='Satisfaction', y='Month', data=df,
scale='width', inner=None)
plt.title('Ridgeline Plot of Customer Satisfaction Scores Over
12 Months')
plt.xlabel('Satisfaction Score')
plt.ylabel('Month')
plt.show()

```

To begin, we import the necessary libraries: numpy, pandas, matplotlib.pyplot, and seaborn. Numpy and pandas are essential for data manipulation, while matplotlib and seaborn are used for data visualization. First, we generate synthetic data for customer satisfaction scores over the past 12 months. We create a list of months using `pd.date_range` and format them as strings. Then, for each month, we generate 200 random satisfaction scores following a normal distribution with a mean of 75 and a standard deviation of 10 using `np.random.normal`. Next, we create a pandas DataFrame from the generated data. Each column in the DataFrame represents a month, and each row represents a satisfaction score.

We then use the melt function to transform the DataFrame into a long format suitable for plotting.

The melt function creates two columns: 'Month' and 'Satisfaction', which is required for the seaborn plotting function.

Finally, we use seaborn's violinplot function to create a ridgeline plot.

We set the x-axis to 'Satisfaction' and the y-axis to 'Month'. The scale='width' parameter ensures that the widths of the violins are comparable.

We remove the inner part of the violins with inner=None to make the plot look cleaner.

After setting the plot's title and axis labels, we display the plot with plt.show().

【Trivia】

Ridgeline plots are useful for visualizing the distribution of a variable over multiple categories or time periods.

They are particularly helpful in showing the evolution of distributions over time.

Seaborn's violinplot is often used to create ridgeline plots because it provides a clear representation of the data's density.

Adjusting the scale parameter in the violinplot function can help in comparing different distributions effectively.

11. Matrix Plot of Confusion Matrix

Importance★★★★☆

Difficulty★★★★☆☆

You are working as a data analyst for a retail company. Your task is to evaluate the performance of a classification model that predicts whether a customer will make a purchase based on their browsing history.

To do this, you need to create a confusion matrix and visualize it using a matrix plot.

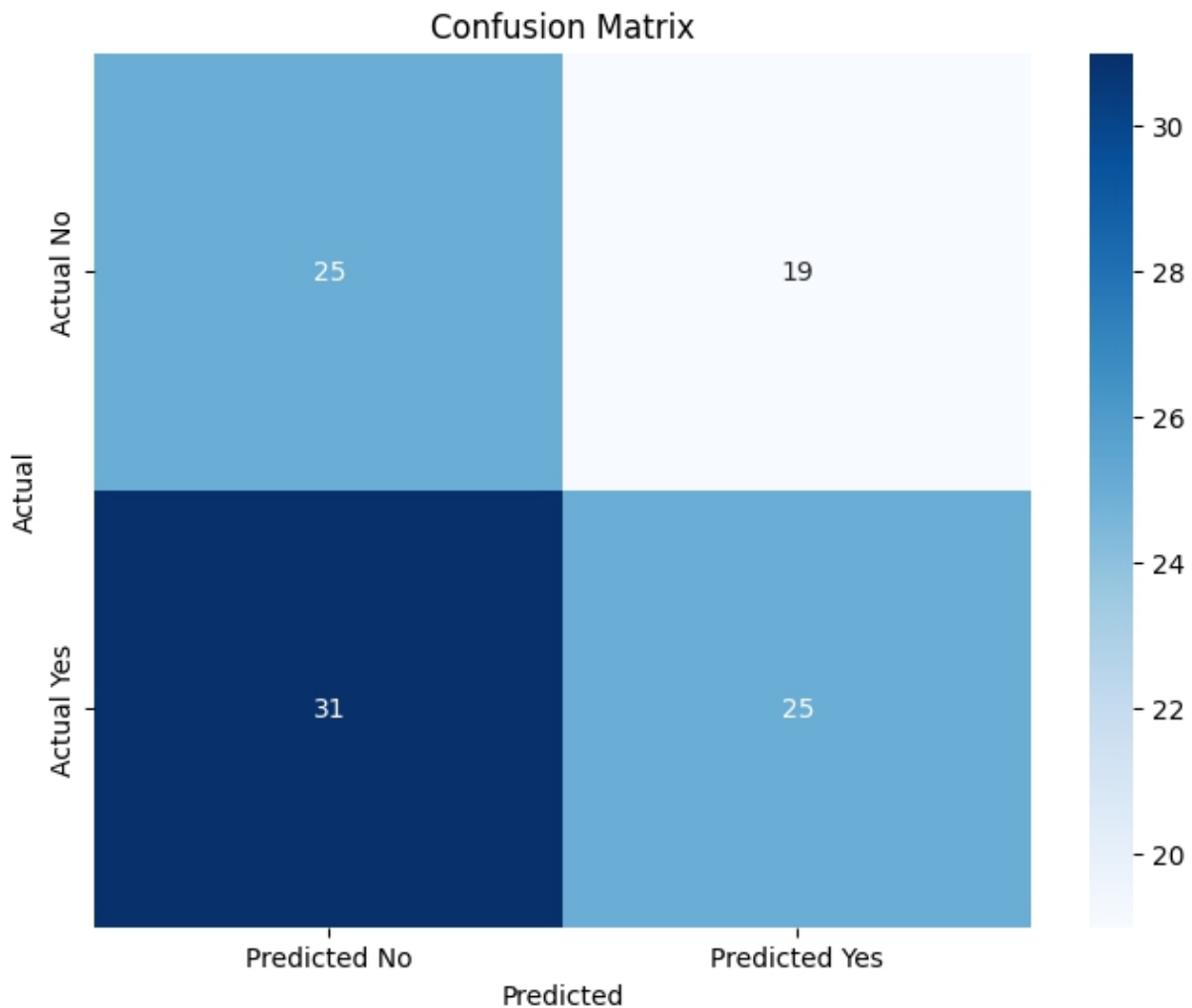
Generate a sample dataset of actual and predicted values, and then create a matrix plot of the confusion matrix.

Ensure that your code outputs the plot correctly.

【Data Generation Code Example】

```
import numpy as np
np.random.seed(42)
actual = np.random.choice([0, 1], size=100)
predicted = np.random.choice([0, 1], size=100)
```


【Diagram Answer】



【Code Answer】

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
np.random.seed(42)
actual = np.random.choice([0, 1], size=100)
predicted = np.random.choice([0, 1], size=100)
```

```
cm = confusion_matrix(actual, predicted)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['Predicted No', 'Predicted Yes'], yticklabels=
['Actual No', 'Actual Yes'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

To solve this problem, you first need to generate a sample dataset of actual and predicted values.

This can be done using NumPy's random choice function to create arrays of binary values (0 and 1).

Once you have the data, you can compute the confusion matrix using the `confusion_matrix` function from the `sklearn.metrics` module.

The confusion matrix is a table that is often used to describe the performance of a classification model.

It shows the counts of true positive, true negative, false positive, and false negative predictions.

Next, to visualize the confusion matrix, you can use the seaborn library, which provides a high-level interface for drawing attractive statistical graphics.

The heatmap function from seaborn can be used to create a matrix plot of the confusion matrix.

In the plot, you can use the `annot` parameter to display the counts in each cell, and the `cmap` parameter to choose a color map.

Labels for the x and y axes can be set using the `xticklabels` and `yticklabels` parameters.

Finally, you can add labels and a title to the plot using `plt.xlabel`, `plt.ylabel`, and `plt.title`, respectively.

The `plt.show` function is used to display the plot.

【Trivia】

- ▶ The confusion matrix is a crucial tool in evaluating the performance of classification models, especially in the case of imbalanced datasets.
- ▶ True Positive (TP) and True Negative (TN) are the cases where the model correctly predicts the positive and negative classes, respectively.
- ▶ False Positive (FP) and False Negative (FN) are the cases where the model incorrectly predicts the positive and negative classes, respectively.
- ▶ The confusion matrix can be used to calculate other important metrics such as accuracy, precision, recall, and F1-score.
- ▶ Seaborn's heatmap function provides a simple yet powerful way to visualize the confusion matrix, making it easier to interpret the results.

12. Plotting a Wind Rose Diagram

Importance★★★★☆

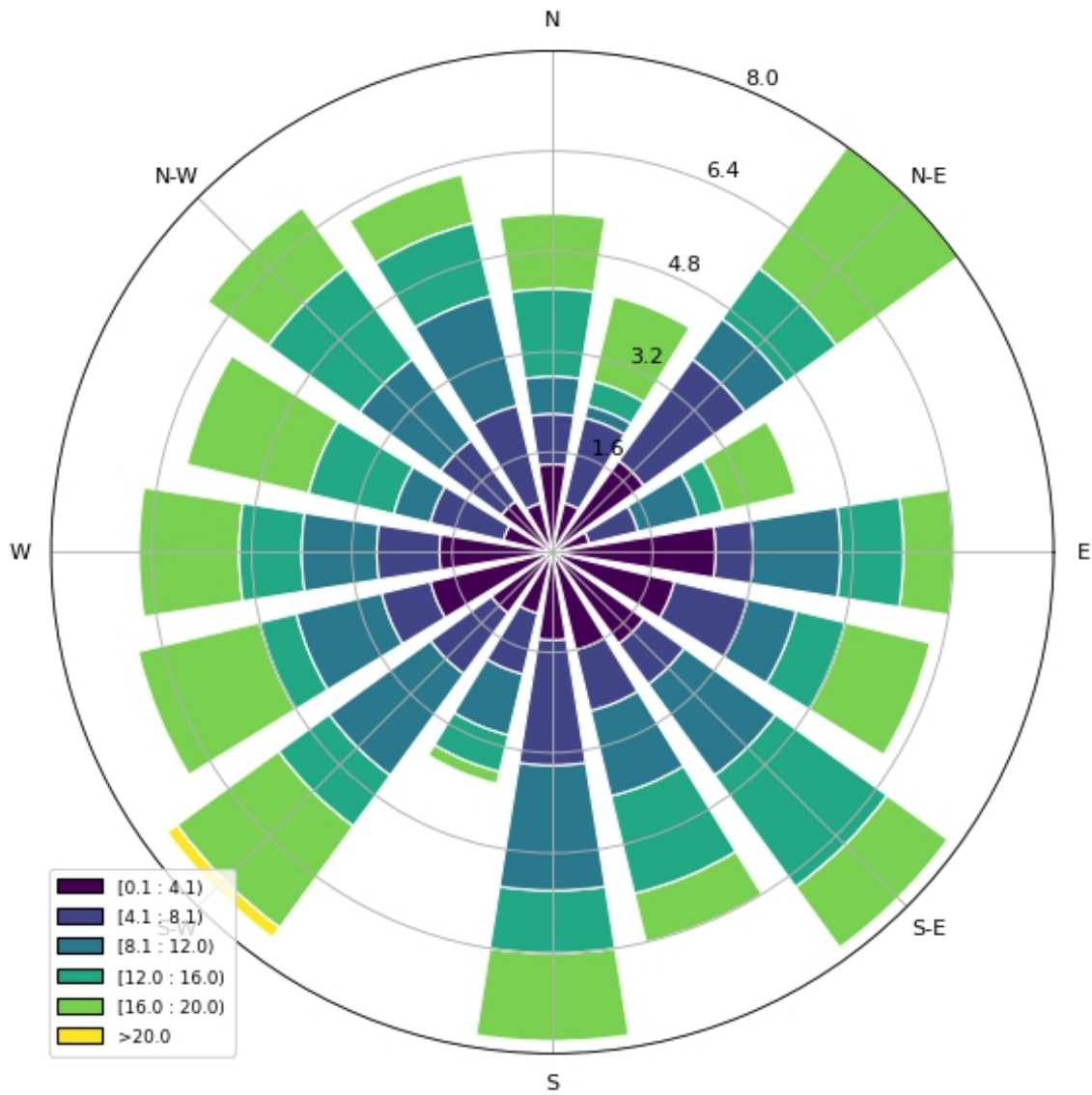
Difficulty★★★★☆☆

A local weather station in your city has collected wind speed and direction data over the past month. They need a visual representation to analyze wind patterns. Your task is to create a Wind Rose Diagram using Python to help them understand the distribution of wind speeds and directions. Use the provided sample data within your code to generate this diagram.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
np.random.seed(0)
directions = np.random.choice(['N', 'NE', 'E', 'SE', 'S', 'SW',
                               'W', 'NW'], 100)
speeds = np.random.uniform(0, 20, 100)
data = pd.DataFrame({'direction': directions, 'speed':
                     speeds})
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.projections.polar import PolarAxes
np.random.seed(0)
```

```

directions = np.random.choice(['N', 'NE', 'E', 'SE', 'S', 'SW',
                               'W', 'NW'], 100)
speeds = np.random.uniform(0, 20, 100)
data = pd.DataFrame({'direction': directions, 'speed':
                    speeds})
def wind_rose(data):
    direction_map = {'N': 0, 'NE': 45, 'E': 90, 'SE': 135, 'S': 180,
                    'SW': 225, 'W': 270, 'NW': 315}
    textdata['angle'] = data['direction'].map(direction_map)
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='polar')
    bins = np.linspace(0, 20, 5)
    for i in range(len(bins)-1):
        subset = data[(data['speed'] >= bins[i]) & (data['speed']
        < bins[i+1])]
        hist, bin_edges = np.histogram(subset['angle'],
        bins=np.arange(0, 361, 45))
        angles = np.deg2rad(np.arange(0, 360, 45))
        widths = np.deg2rad(45)
        ax.bar(angles, hist, width=widths, bottom=i,
        label=f'{bins[i]:.1f}-{bins[i+1]:.1f} m/s')
    ax.set_theta_zero_location('N')
    ax.set_theta_direction(-1)
    ax.set_xticks(angles)
    ax.set_xticklabels(['N', 'NE', 'E', 'SE', 'S', 'SW', 'W', 'NW'])
    ax.set_yticks(range(len(bins)))
    ax.set_yticklabels([f'{bins[i]:.1f}-{bins[i+1]:.1f}' for i in
    range(len(bins)-1)])
    plt.legend(loc='upper right', bbox_to_anchor=(1.1, 1.1))
    plt.title('Wind Rose Diagram')

```

```
plt.show()  
wind_rose(data)
```

To create a Wind Rose Diagram, we first need to import necessary libraries such as numpy, pandas, and matplotlib. We generate sample data for wind directions and speeds using numpy's random functions.

The data is stored in a pandas DataFrame.

We then map the wind directions to corresponding angles using a dictionary.

The wind_rose function converts wind directions to angles and plots the data on a polar plot.

We divide wind speeds into bins and plot each bin as a bar in the polar plot, with the angle representing the wind direction.

The bars' heights represent the frequency of wind speeds within each bin.

We configure the polar plot to have North at the top and angles increasing clockwise.

Finally, we add labels and a legend to the plot for clarity and display the plot using plt.show().

【Trivia】

- ▶ Wind Rose Diagrams are commonly used in meteorology to visualize wind patterns over a specific period.
- ▶ They help in understanding prevailing wind directions and speeds, which is crucial for various applications such as aviation, marine navigation, and urban planning.
- ▶ The concept of a Wind Rose dates back to ancient times when mariners used it for navigation.

13. Bullet Chart for Performance Targets

Importance★★★★☆

Difficulty★★★★☆

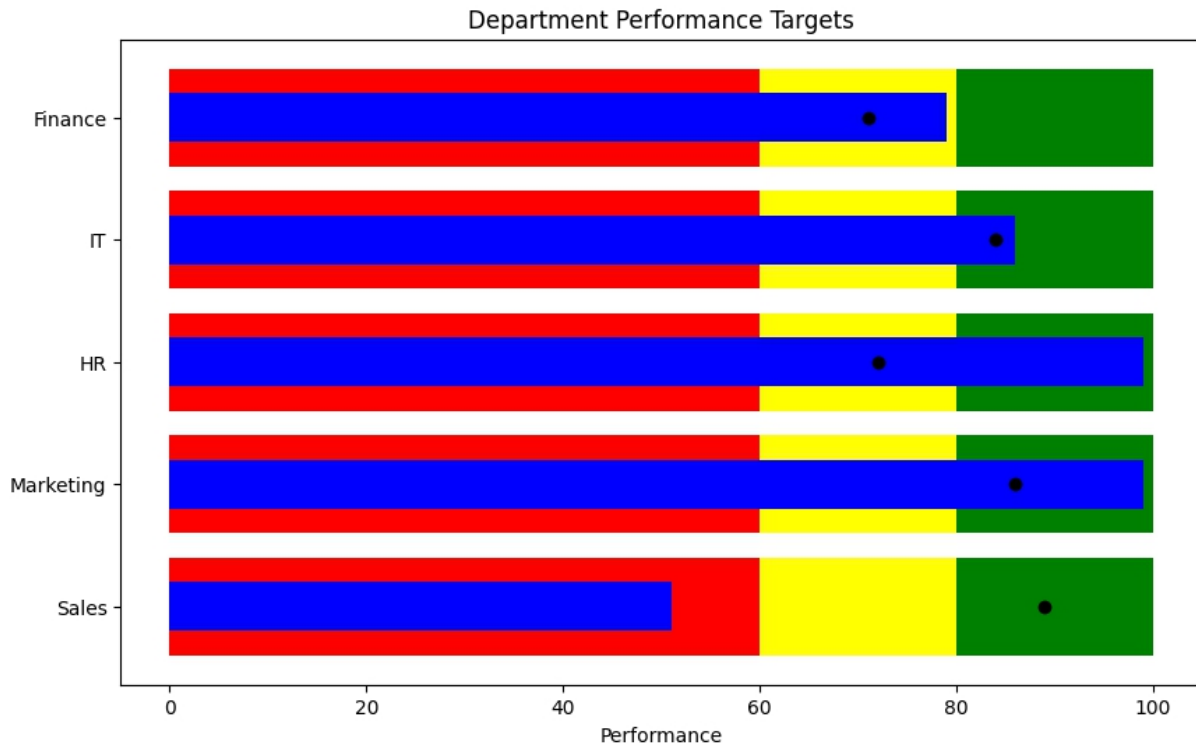
You are working as a data analyst for a retail company. Your manager has asked you to create a bullet chart to visualize the performance targets of different departments. The chart should display the actual performance, target, and ranges (poor, satisfactory, and good). The data should be generated within the code.

Create a Python script that generates the necessary data and visualizes it using a bullet chart.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
# Generate sample data
data = pd.DataFrame({
    'Department': ['Sales', 'Marketing', 'HR', 'IT', 'Finance'],
    'Actual': np.random.randint(50, 100, size=5),
    'Target': np.random.randint(70, 90, size=5),
    'Poor': [60] * 5,
    'Satisfactory': [80] * 5,
    'Good': [100] * 5
})
data
```


【Diagram Answer】



【Code Answer】

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
# Generate sample data
data = pd.DataFrame({
    'Department': ['Sales', 'Marketing', 'HR', 'IT', 'Finance'],
    'Actual': np.random.randint(50, 100, size=5),
    'Target': np.random.randint(70, 90, size=5),
    'Poor': [60] * 5,
    'Satisfactory': [80] * 5,
    'Good': [100] * 5
})
```

```

# Create a bullet chart
fig, ax = plt.subplots(figsize=(10, 6))
for idx, row in data.iterrows():
    ax.broken_barh([(0, row['Poor'])], (idx - 0.4, 0.8),
facecolors='red')
    ax.broken_barh([(row['Poor'], row['Satisfactory'] -
row['Poor'])], (idx - 0.4, 0.8), facecolors='yellow')
    ax.broken_barh([(row['Satisfactory'], row['Good'] -
row['Satisfactory'])], (idx - 0.4, 0.8), facecolors='green')
    ax.broken_barh([(0, row['Actual'])], (idx - 0.2, 0.4),
facecolors='blue')
    ax.plot(row['Target'], idx, 'o', color='black')
ax.set_yticks(range(len(data)))
ax.set_yticklabels(data['Department'])
ax.set_xlabel('Performance')
ax.set_title('Department Performance Targets')
plt.show()

```

To create a bullet chart in Python, we first generate the sample data using pandas and numpy. The data includes the department names, actual performance, target performance, and the ranges for poor, satisfactory, and good performance.

We then use matplotlib to create the bullet chart. The `broken_barh` function is used to create horizontal bars for the different performance ranges. The `facecolors` parameter is used to set the colors for the different ranges: red for poor, yellow for satisfactory, and green for good.

For the actual performance, we use a narrower bar with a blue color to distinguish it from the performance ranges. The target performance is marked with a black dot using the `plot` function.

The y-axis is labeled with the department names, and the x-axis represents the performance. The chart title is set to "Department Performance Targets".

This visualization helps to quickly compare the actual performance of each department against their targets and predefined performance ranges.

【Trivia】

Bullet charts were introduced by Stephen Few as a way to replace dashboard gauges and meters. They are more space-efficient and provide a clearer view of performance data. Bullet charts are particularly useful in business dashboards to visualize key performance indicators (KPIs).

14. Creating a Horizon Chart with Time Series Data

Importance★★★★☆

Difficulty★★★★☆

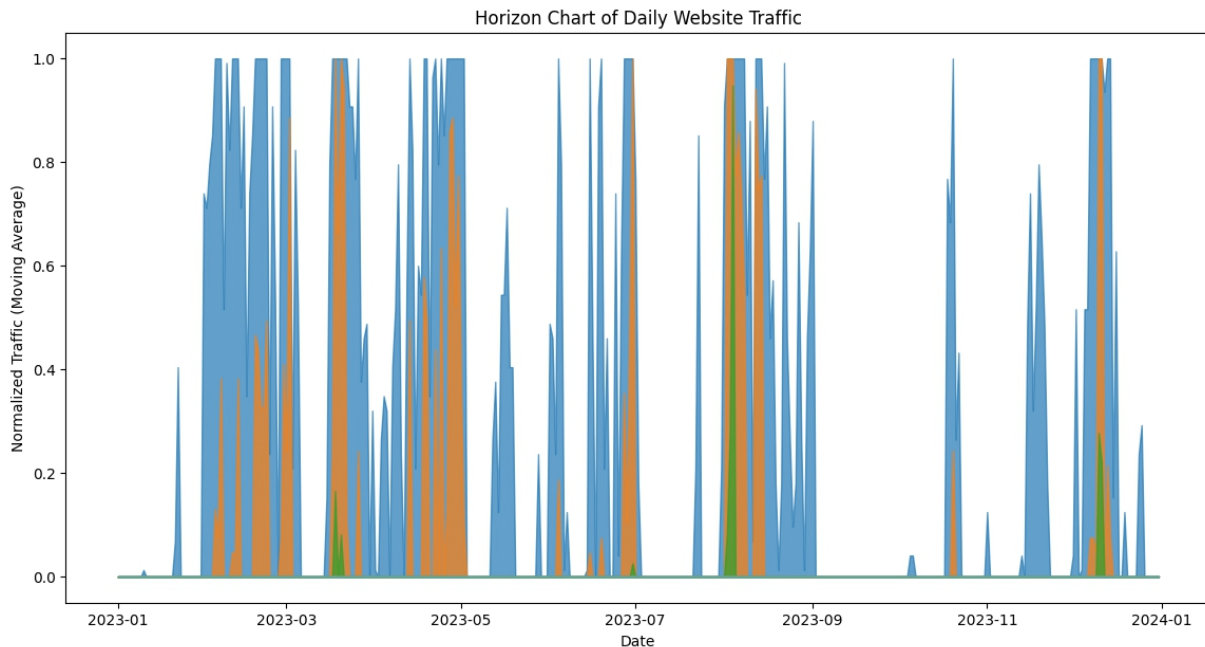
You are a data analyst at a company that monitors website traffic over time. You need to visualize this data using a horizon chart to easily identify trends and patterns. The data includes daily visits to the website over the past year. Create a horizon chart of this time series data. Generate synthetic data for the purpose of this exercise. Ensure the code efficiently processes and visualizes the data. Requirements: Generate daily website traffic data for the past year. Process the data to prepare it for visualization. Create a horizon chart to visualize the trends and patterns in the data.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
## Generate sample data
np.random.seed(42)
dates = pd.date_range(start="2023-01-01", end="2023-12-31")
traffic = np.random.poisson(lam=200, size=len(dates))
## Create DataFrame
data = pd.DataFrame({"Date": dates, "Traffic": traffic})
## Process data for horizon chart
# Calculate moving average to smooth the data
```

```
data["Traffic_MA"] =  
data["Traffic"].rolling(window=7).mean().fillna(method='bfill'  
)  
# Display the data  
data.head()
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
## Generate sample data
np.random.seed(42)
dates = pd.date_range(start="2023-01-01", end="2023-12-31")
traffic = np.random.poisson(lam=200, size=len(dates))
## Create DataFrame
data = pd.DataFrame({"Date": dates, "Traffic": traffic})
## Process data for horizon chart
data["Traffic_MA"] =
data["Traffic"].rolling(window=7).mean().fillna(method='bfill')
```

```

')
## Normalize the data for horizon chart
data["Traffic_MA_Norm"] = (data["Traffic_MA"] -
data["Traffic_MA"].mean()) / data["Traffic_MA"].std()
## Create horizon chart
fig, ax = plt.subplots(figsize=(14, 7))
## Plot different layers of the horizon chart
colors = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728",
"#9467bd", "#8c564b", "#e377c2", "#7f7f7f", "#bcbd22",
"#17becf"]
for i, color in enumerate(colors):
    layer = np.maximum(0, data["Traffic_MA_Norm"] - i) -
np.maximum(0, data["Traffic_MA_Norm"] - (i + 1))
    ax.fill_between(data["Date"], layer, color=color,
alpha=0.7)
## Set labels and title
ax.set_title("Horizon Chart of Daily Website Traffic")
ax.set_xlabel("Date")
ax.set_ylabel("Normalized Traffic (Moving Average)")
plt.show()

```

A horizon chart is an effective way to visualize time series data by layering different bands of the data on top of each other. This technique allows for the identification of trends and patterns even in large datasets. To create a horizon chart, follow these steps: Generate the Data: A random dataset of daily website traffic over a year is generated using a Poisson distribution. This is done for simplicity and to mimic the fluctuations in real website traffic data. The dates are created using `pd.date_range`, and the traffic values are generated using `np.random.poisson`. Create the

DataFrame: The generated dates and traffic data are combined into a DataFrame using `pd.DataFrame`. Process the Data: A moving average is calculated using the `.rolling(window=7).mean()` method to smooth the traffic data. This helps in reducing noise and making trends more apparent. Normalize the Data: The moving average data is normalized by subtracting the mean and dividing by the standard deviation. Normalization helps in standardizing the data range, making it suitable for a horizon chart. Create the Horizon Chart: The horizon chart is created by plotting different layers of the normalized data. Each layer represents a band of values, and different colors are used to differentiate these bands. The `fill_between` function is used to fill the area between layers with colors, providing a clear visual distinction. Labels and titles are added to the chart for clarity. By following these steps, the horizon chart effectively visualizes the trends and patterns in the website traffic data, making it easier to analyze and interpret.

【Trivia】

Horizon charts are particularly useful for large datasets where traditional line or bar charts become cluttered and difficult to interpret. They were popularized by the software company Panopticon Software (now part of Altair) and are often used in financial data analysis, climate data visualization, and any field requiring the clear representation of time series data. The concept of layering data bands in different colors allows for a compact and visually appealing representation, making it easier to spot anomalies and trends at a glance.

15. Network Flow Diagram for Traffic Data Visualization

Importance★★★★☆

Difficulty★★★★☆

You are working for a city transportation department that wants to visualize the flow of traffic between various intersections in the city.

Using Python, generate a network flow diagram to show traffic data between these intersections.

Create a random dataset representing traffic flow between different intersections, including the following columns: 'Source', 'Target', and 'Weight'.

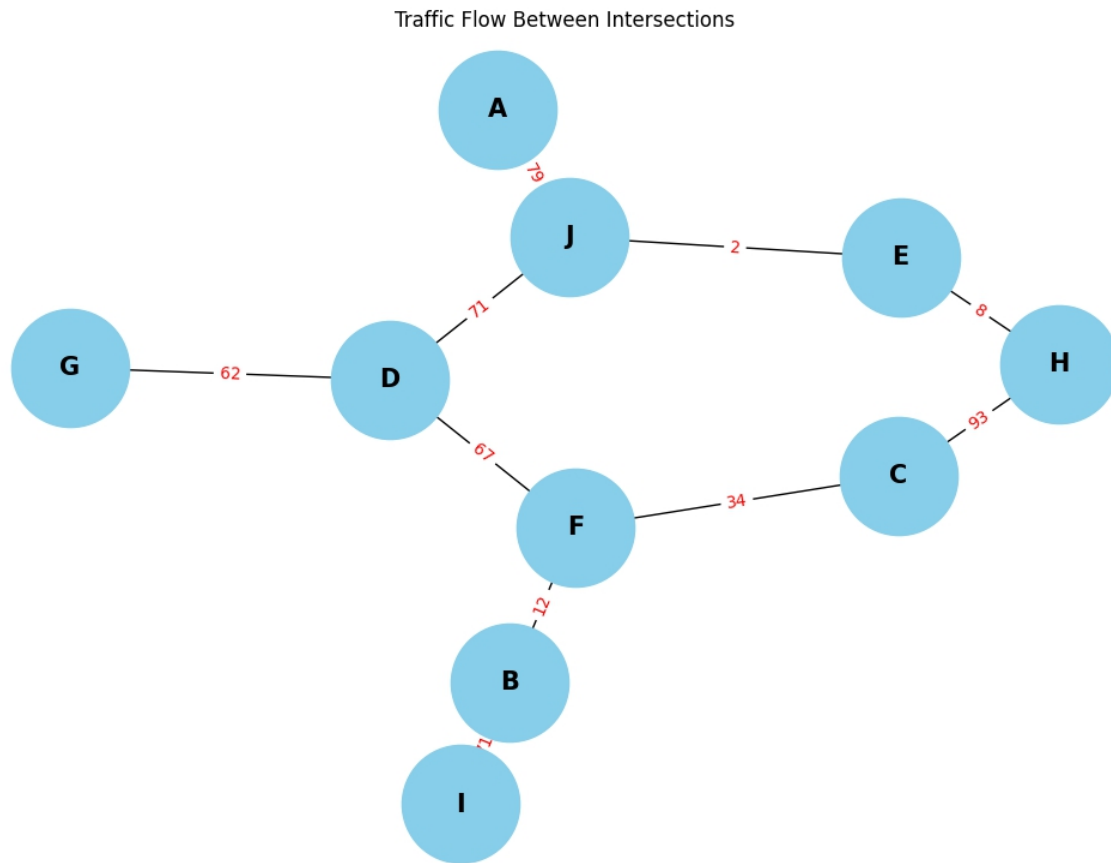
'Source' and 'Target' columns should contain intersection names, and the 'Weight' column should represent the traffic flow intensity.

Visualize this network using a suitable Python library.

【Data Generation Code Example】

```
import random
import pandas as pd
sources = ['A', 'B', 'C', 'D', 'E']
targets = ['F', 'G', 'H', 'I', 'J']
random.seed(0)
data = {'Source': [random.choice(sources) for _ in
range(15)],
'Target': [random.choice(targets) for _ in range(15)],
'Weight': [random.randint(1, 100) for _ in range(15)]}
df = pd.DataFrame(data)
df.head()
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import random
import matplotlib.pyplot as plt
import networkx as nx
sources = ['A', 'B', 'C', 'D', 'E']
targets = ['F', 'G', 'H', 'I', 'J']
random.seed(0)
data = {'Source': [random.choice(sources) for _ in
range(15)],
'Target': [random.choice(targets) for _ in range(15)],
```

```
'Weight': [random.randint(1, 100) for _ in range(15)]}
df = pd.DataFrame(data)
G = nx.from_pandas_edgelist(df, 'Source', 'Target', True)
pos = nx.spring_layout(G)
plt.figure(figsize=(10, 7))
nx.draw(G, pos, with_labels=True, node_size=5000,
node_color='skyblue', font_size=15, font_weight='bold')
edge_labels = nx.get_edge_attributes(G, 'Weight')
nx.draw_networkx_edge_labels(G, pos,
edge_labels=edge_labels, font_color='red')
plt.title('Traffic Flow Between Intersections')
plt.show()
```

To generate a network flow diagram for traffic data visualization, we start by importing the necessary libraries: pandas for data manipulation, random for generating random data, matplotlib for plotting, and networkx for creating the network graph.

We define the source and target intersections and use a random seed for reproducibility.

We then create a dataset with columns 'Source', 'Target', and 'Weight' to represent the traffic flow between intersections.

The dataset is converted into a pandas DataFrame.

Next, we use networkx to create a graph from the DataFrame using 'Source' and 'Target' as nodes and 'Weight' as edge attributes.

We define the position of nodes using the spring layout algorithm and plot the graph using matplotlib.

Nodes are drawn with labels, and edges are labeled with their corresponding weights to show traffic flow intensity.

Finally, we set the plot title and display the graph.

【Trivia】

- ▶ Network flow diagrams are widely used in various fields, including transportation, telecommunications, and logistics, to visualize and optimize the flow of goods, information, and traffic.
- ▶ NetworkX is a powerful Python library specifically designed for the creation, manipulation, and study of complex networks and their dynamics.
- ▶ Visualization of traffic data helps city planners and engineers identify congestion points and optimize traffic light timings to improve overall traffic flow efficiency.

16. Heatmap of Missing Data Visualization

Importance★★★★★

Difficulty★★★☆☆

You are a data analyst at a retail company. Your task is to analyze a dataset containing information about customer transactions.

This dataset contains missing values, and your manager wants to visualize these missing values to understand the data quality.

Create a heatmap to visualize the missing data in the dataset.

To simulate the scenario, generate a sample dataset with missing values and then plot the heatmap of the missing data.

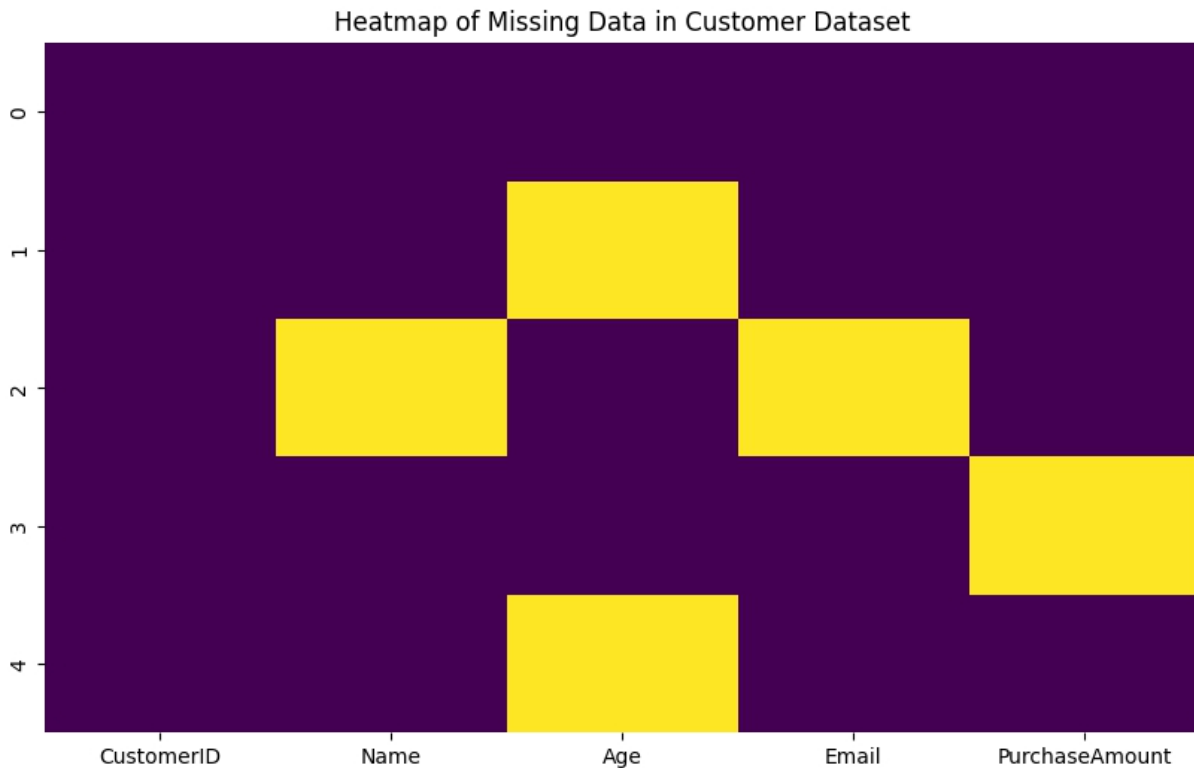
Use Python for data manipulation and visualization.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(0)
data = pd.DataFrame({
    'CustomerID': np.arange(1, 101),
    'TransactionAmount': np.random.choice([np.nan, 50, 100, 150, 200], 100, p=[0.1, 0.3, 0.3, 0.2, 0.1]),
    'ProductID': np.random.choice([np.nan, 'P1', 'P2', 'P3', 'P4'], 100, p=[0.2, 0.3, 0.2, 0.2, 0.1]),
    'PurchaseDate': pd.date_range(start='2023-01-01', periods=100, freq='D')
})
```



【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
np.random.seed(0)
data = pd.DataFrame({
    'CustomerID': np.arange(1, 101),
    'TransactionAmount': np.random.choice([np.nan, 50, 100,
    150, 200], 100, p=[0.1, 0.3, 0.3, 0.2, 0.1]),
    'ProductID': np.random.choice([np.nan, 'P1', 'P2', 'P3', 'P4'],
    100, p=[0.2, 0.3, 0.2, 0.2, 0.1]),
```

```
'PurchaseDate': pd.date_range(start='2023-01-01',
periods=100, freq='D')
})
missing_data = data.isnull()
plt.figure(figsize=(10, 6))
sns.heatmap(missing_data, cbar=False, cmap='viridis')
plt.title('Heatmap of Missing Data')
plt.xlabel('Columns')
plt.ylabel('Rows')
plt.show()
```

First, a sample dataset is created using Pandas and NumPy. The dataset consists of 100 records with four columns: 'CustomerID', 'TransactionAmount', 'ProductID', and 'PurchaseDate'.

Missing values are introduced in the 'TransactionAmount' and 'ProductID' columns using the `np.random.choice` method.

The `isnull` method of the DataFrame is then used to create a boolean DataFrame indicating the presence of missing values.

This boolean DataFrame is visualized using Seaborn's heatmap function, where missing values are represented in a distinct color.

The heatmap is customized with a title and axis labels using Matplotlib to ensure clarity.

【Trivia】

▶ Heatmaps are a powerful tool for visualizing missing data because they provide a clear and immediate visual representation of where missing values are concentrated in the dataset.

- ▶ Seaborn's heatmap function is highly customizable, allowing you to change colors, add annotations, and modify other aesthetic aspects to suit your needs.
- ▶ Handling and visualizing missing data is a critical step in data preprocessing, as it helps in deciding the best method to handle these gaps, such as imputation or removal of missing values.

17. Connected Scatter Plot for Sales Trend Analysis

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst for a retail company.

The marketing team wants to visualize the sales trend of their top-selling product over the past 12 months.

They specifically requested a connected scatter plot to show the relationship between time and sales volume.

Your task is to create a connected scatter plot using Python that displays:

Monthly sales data points

A line connecting these points in chronological order

Clear labels for each month

Appropriate axis labels and a title

Use the provided code to generate sample data for 12 months of sales.

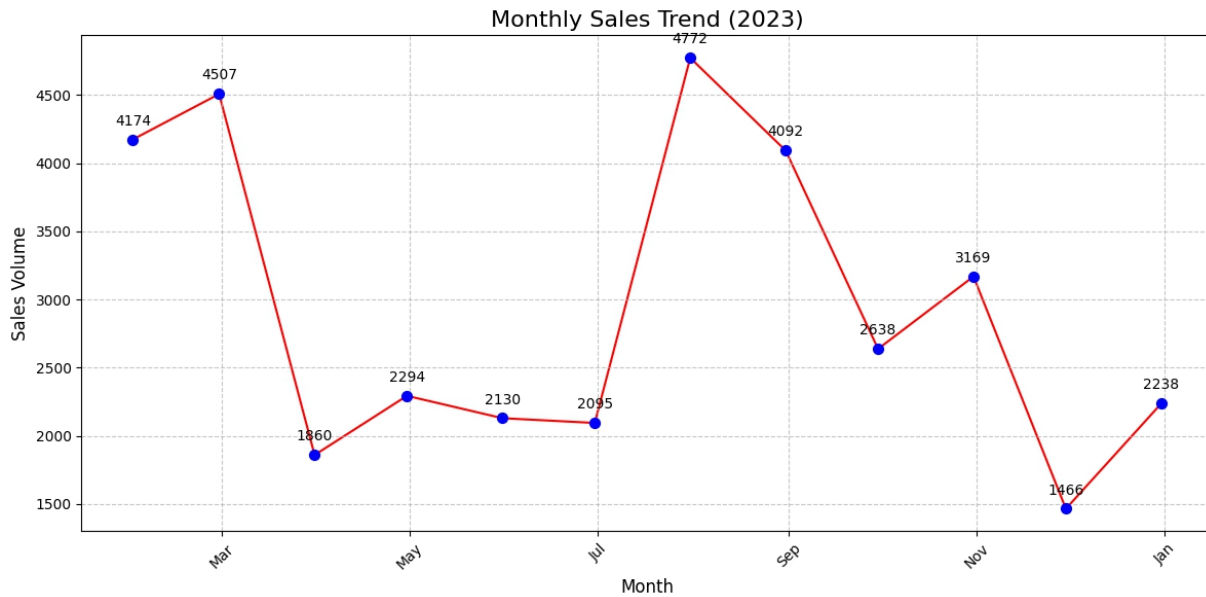
Then, create the connected scatter plot using matplotlib.

Ensure your code is efficient and follows best practices for data visualization in Python.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(42)
dates = pd.date_range(start='2023-01-01', end='2023-12-31', freq='M')
sales = np.random.randint(1000, 5000, size=12)
df = pd.DataFrame({'Date': dates, 'Sales': sales})
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)
dates = pd.date_range(start='2023-01-01', end='2023-12-31', freq='M')
sales = np.random.randint(1000, 5000, size=12)
df = pd.DataFrame({'Date': dates, 'Sales': sales})
# Set up the plot
plt.figure(figsize=(12, 6))
plt.scatter(df['Date'], df['Sales'], s=50, color='blue', zorder=2)
plt.plot(df['Date'], df['Sales'], color='red', zorder=1)
# Customize the plot
plt.title('Monthly Sales Trend (2023)', fontsize=16)
```

```

plt.xlabel('Month', fontsize=12)
plt.ylabel('Sales Volume', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
# Format x-axis to show month names
plt.gca().xaxis.set_major_formatter(plt.matplotlib.dates.Date
Formatter('%b'))
plt.xticks(rotation=45)
# Add labels for each point
[plt.annotate(f'{sale}', (date, sale), textcoords="offset
points", xytext=(0,10), ha='center') for date, sale in
zip(df['Date'], df['Sales'])]
plt.tight_layout()
plt.show()

```

This code creates a connected scatter plot to visualize monthly sales data.

Here's a detailed explanation of the Python data processing and visualization techniques used:

Data Generation:

We use pandas and numpy to create sample data.

`pd.date_range()` generates a series of dates at monthly intervals for the year 2023.

`np.random.randint()` creates random sales figures between 1000 and 5000.

The data is stored in a pandas DataFrame for easy manipulation.

Setting up the Plot:

`plt.figure(figsize=(12, 6))` creates a new figure with specified dimensions.

`plt.scatter()` plots individual data points (scatter plot).

`plt.plot()` connects these points with a line.

The 'zorder' parameter ensures the scatter points appear on top of the line.

Customizing the Plot:

`plt.title()`, `plt.xlabel()`, and `plt.ylabel()` set the title and axis labels.

`plt.grid()` adds a grid to the plot for better readability.

Formatting the X-axis:

`plt.gca().xaxis.set_major_formatter()` formats the x-axis to show month abbreviations.

`plt.xticks(rotation=45)` rotates the x-axis labels for better visibility.

Adding Data Labels:

A list comprehension with `plt.annotate()` adds labels to each data point.

'textcoords' and 'xytext' parameters position the labels above each point.

Finalizing and Displaying:

`plt.tight_layout()` adjusts the plot to fit within the figure area.

`plt.show()` displays the final plot.

This code demonstrates key concepts in Python data visualization:

Data manipulation with pandas

Creating and customizing plots with matplotlib

Combining different plot types (scatter and line)

Formatting axis labels and adding annotations

Using list comprehensions for efficient data labeling

【Trivia】

- ▶ Connected scatter plots are particularly useful for showing how a relationship between two variables evolves over time.
- ▶ This type of plot was popularized by The New York Times in their data journalism pieces.

- ▶ While matplotlib is used here, other Python libraries like Seaborn or Plotly can also create similar visualizations with different syntax.
- ▶ The 'zorder' parameter in matplotlib determines the drawing order of plot elements, which is crucial for ensuring visibility of all components.
- ▶ List comprehensions, as used for adding labels, are a powerful Python feature that can significantly reduce code complexity and improve readability.
- ▶ The DateFormatter class from matplotlib.dates is a powerful tool for customizing date representations on plot axes.
- ▶ Random seed setting (`np.random.seed()`) ensures reproducibility of random data generation, which is crucial for consistent results in data analysis and visualization.

18. Nested Pie Chart of Demographic Data

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst for a city government. Your task is to visualize the demographic breakdown of the city's population by age groups and gender using a nested pie chart. The city's population is divided into three main age groups: Children (0-14 years), Adults (15-64 years), and Seniors (65+ years). Each age group is further divided by gender (Male, Female).

Create a nested pie chart to represent this data. The data is as follows:

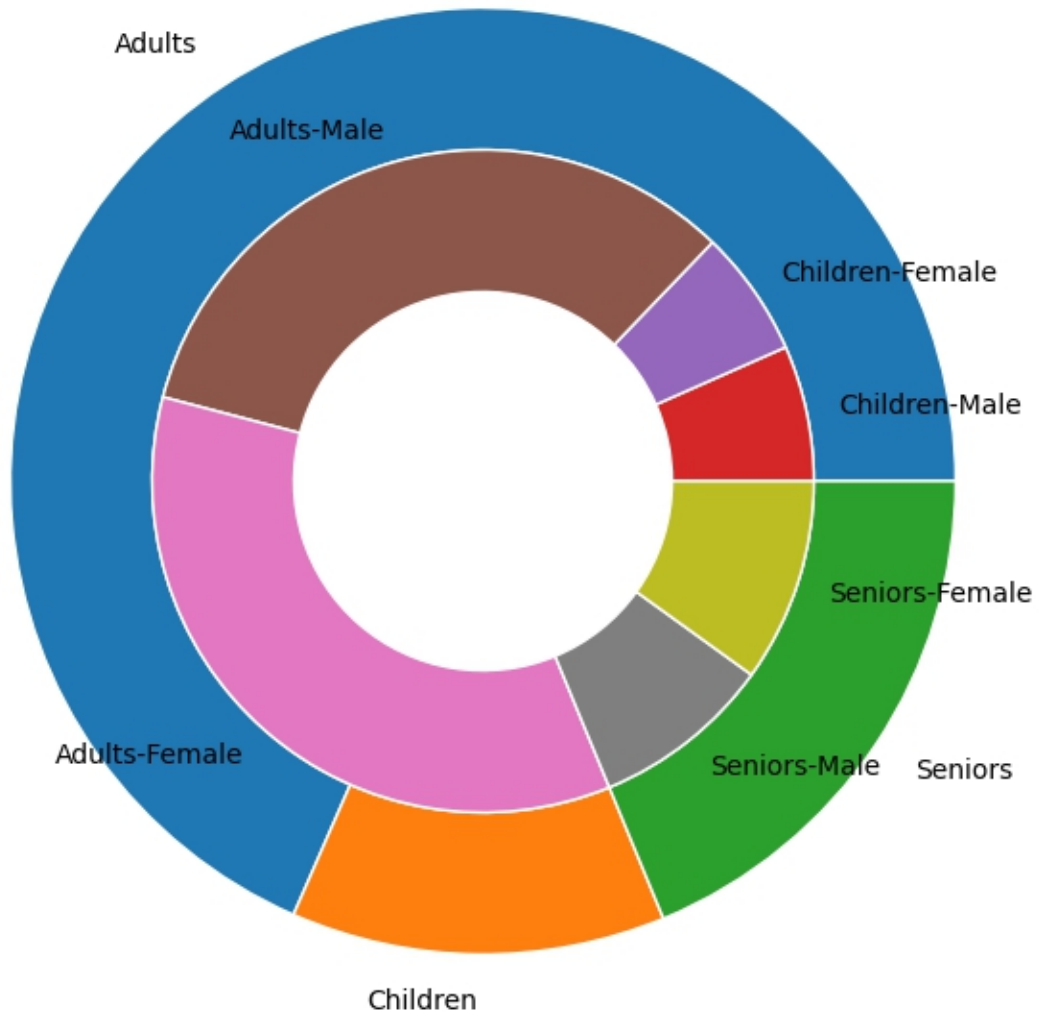
- ▶ Children: Male (3000), Female (2800)
- ▶ Adults: Male (15000), Female (16000)
- ▶ Seniors: Male (4000), Female (4500)

【Data Generation Code Example】

```
import pandas as pd
data = {'Age Group': ['Children', 'Children', 'Adults', 'Adults', 'Seniors', 'Seniors'],
        'Gender': ['Male', 'Female', 'Male', 'Female', 'Male', 'Female'],
        'Population': [3000, 2800, 15000, 16000, 4000, 4500]}
df = pd.DataFrame(data)
```

【Diagram Answer】

City Population Demographics



【Code Answer】

```
import pandas as pd
import matplotlib.pyplot as plt
```



```

data = {'Age Group': ['Children', 'Children', 'Adults', 'Adults',
'Seniors', 'Seniors'],
'Gender': ['Male', 'Female', 'Male', 'Female', 'Male', 'Female'],
'Population': [3000, 2800, 15000, 16000, 4000, 4500]}
df = pd.DataFrame(data)
groups = df.groupby('Age Group').sum().reset_index()
size1 = groups['Population'].values
size2 = df['Population'].values
labels1 = groups['Age Group'].values
labels2 = df.apply(lambda x: f"{x['Age Group']}-
{x['Gender']}", axis=1).values
plt.figure(figsize=(8, 8))
ax = plt.gca()
ax.pie(size1, labels=labels1, radius=1,
wedgeprops=dict(width=0.3, edgecolor='w'))
ax.pie(size2, labels=labels2, radius=0.7,
wedgeprops=dict(width=0.3, edgecolor='w'))
plt.title('City Population Demographics')
plt.show()

```

To solve this problem, start by importing the necessary libraries: pandas for data manipulation and matplotlib for plotting.

First, create a dictionary with the given demographic data and convert it into a DataFrame using pandas. The DataFrame should contain columns for the age group, gender, and population.

Next, group the data by the 'Age Group' column and sum the populations to get the total population for each age group. Reset the index of the grouped data to ensure it aligns correctly for plotting.

Extract the population sizes and labels for both the outer and inner rings of the pie chart. The outer ring represents the age groups, and the inner ring represents the gender distribution within each age group.

Create a nested pie chart using matplotlib's pie function. The outer pie chart uses the total population sizes for each age group, and the inner pie chart uses the population sizes for each gender within each age group. Set the radius and width of the wedges to create the nested effect. Ensure the edge colors are white for better visual separation. Finally, set the chart title and display the plot using `plt.show()`.

【Trivia】

Nested pie charts, also known as sunburst charts or multi-level pie charts, are a type of visualization that allows you to show hierarchical data through concentric circles. The outer ring represents the top level of the hierarchy, while the inner rings represent lower levels. This type of chart is particularly useful for displaying the breakdown of a population into subcategories, such as demographic data, sales by category, or organizational structures.

19. Creating a Dumbbell Dot Plot for Sales Comparison

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst for a retail company that wants to compare sales performance between two quarters. The company has provided you with sales data for various product categories in Q1 and Q2.

Your task is to create a dumbbell dot plot to visually represent the change in sales for each category between these two quarters.

The plot should clearly show the sales values for both quarters and the difference between them.

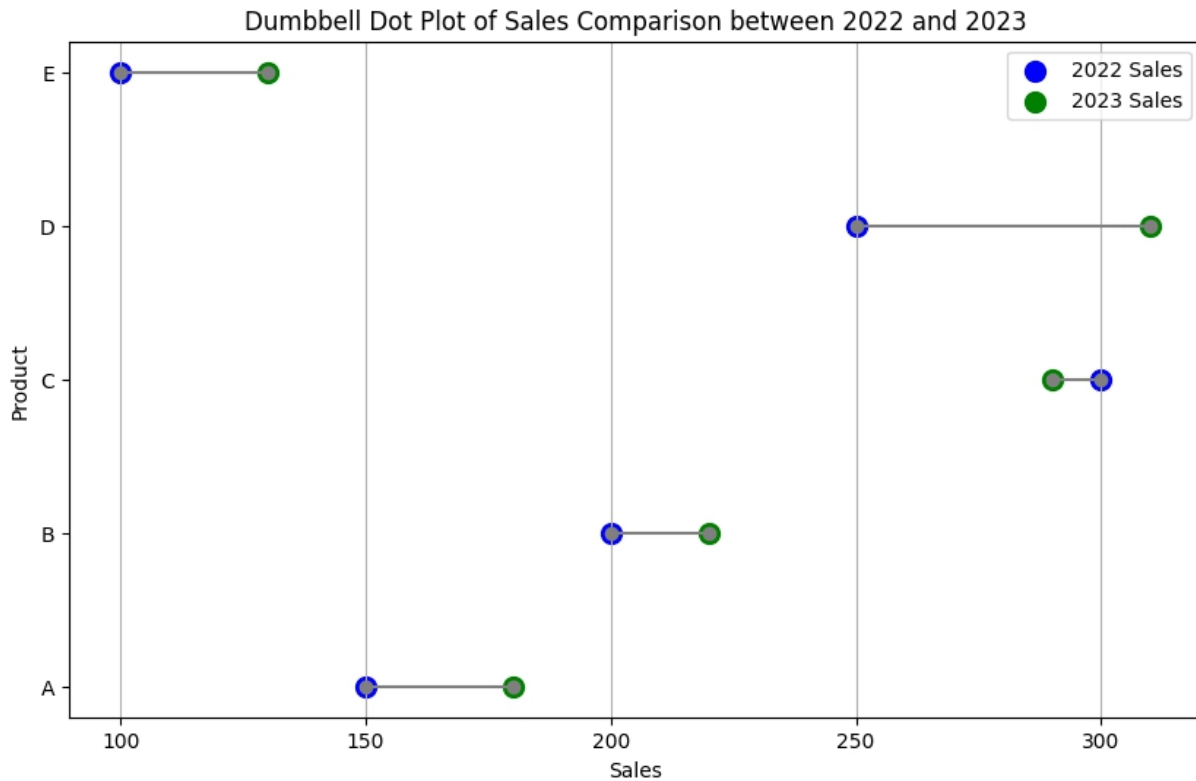
Use Python to generate sample data, process it, and create the dumbbell dot plot.

Make sure to include proper labeling and formatting to make the plot easily interpretable.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(42)
categories = ['Electronics', 'Clothing', 'Home', 'Books',
'Sports', 'Food']
q1_sales = np.random.randint(100, 1000, len(categories))
q2_sales = q1_sales + np.random.randint(-200, 400,
len(categories))
data = pd.DataFrame({'Category': categories, 'Q1_Sales':
q1_sales, 'Q2_Sales': q2_sales})
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)
categories = ['Electronics', 'Clothing', 'Home', 'Books',
'Sports', 'Food']
q1_sales = np.random.randint(100, 1000, len(categories))
q2_sales = q1_sales + np.random.randint(-200, 400,
len(categories))
data = pd.DataFrame({'Category': categories, 'Q1_Sales':
q1_sales, 'Q2_Sales': q2_sales})
fig, ax = plt.subplots(figsize=(10, 6))
```

```

y_positions = range(len(categories))
# Plot lines connecting Q1 and Q2 sales
[plt.plot([row.Q1_Sales, row.Q2_Sales], [i, i], 'o-',
color='gray', linewidth=1) for i, row in data.iterrows()]
# Plot Q1 sales (blue dots)
ax.scatter(data['Q1_Sales'], y_positions, color='blue', s=50,
label='Q1 Sales')
# Plot Q2 sales (red dots)
ax.scatter(data['Q2_Sales'], y_positions, color='red', s=50,
label='Q2 Sales')
# Set y-axis labels and ticks
ax.set_yticks(y_positions)
ax.set_yticklabels(categories)
# Set labels and title
ax.set_xlabel('Sales ($)')
ax.set_title('Sales Comparison: Q1 vs Q2')
ax.legend()
# Add grid lines
ax.grid(True, axis='x', linestyle='--', alpha=0.7)
# Adjust layout and display plot
plt.tight_layout()
plt.show()

```

This code creates a dumbbell dot plot to compare sales data between two quarters for different product categories.

Let's break down the key steps in the data processing and visualization process:

Data Generation:

We use NumPy to generate random sales data for Q1 and Q2.

The `np.random.seed(42)` ensures reproducibility of the random data.

We create a pandas DataFrame to store the category names and sales data.

Plot Setup:

We use `matplotlib` to create the plot, setting the figure size to 10x6 inches.

The `y_positions` variable is created to determine the vertical placement of each category.

Plotting the Dumbbell Lines:

We use a list comprehension with `plt.plot()` to draw lines connecting Q1 and Q2 sales for each category.

These lines are colored gray and have circular markers at each end.

Plotting the Sales Points:

We use `ax.scatter()` to plot Q1 sales as blue dots and Q2 sales as red dots.

The `s=50` parameter sets the size of the dots.

Y-axis Formatting:

We set the y-axis ticks to match our `y_positions` and label them with the category names.

Labels and Title:

We add an x-axis label for "Sales (\$)" and set a title for the plot.

A legend is added to distinguish between Q1 and Q2 sales.

Grid Lines:

We add vertical grid lines to make it easier to read the sales values.

Final Adjustments:

`plt.tight_layout()` is used to automatically adjust the plot layout.

Finally, `plt.show()` displays the plot.

This visualization technique effectively shows the change in sales between Q1 and Q2 for each product category.

The dumbbell design allows for easy comparison of the two values, with the length of the line indicating the magnitude of change.

The color coding (blue for Q1, red for Q2) further enhances the readability of the plot.

【Trivia】

- ▶ Dumbbell plots, also known as DNA plots or barbell plots, are excellent for comparing two related values across multiple categories.
- ▶ This type of plot is named after its resemblance to a dumbbell weight, with two data points connected by a line.
- ▶ Dumbbell plots are particularly useful in business contexts for before-and-after comparisons, such as sales performance across different time periods.
- ▶ The technique of using `np.random.seed()` is crucial in data science for reproducibility, allowing others to generate the same "random" data.
- ▶ Matplotlib, the library used for plotting, is highly customizable and can create a wide variety of statistical graphics beyond dumbbell plots.
- ▶ In data visualization, the principle of "small multiples" (repeating the same chart type for different categories) is effectively applied in dumbbell plots.
- ▶ The use of pandas DataFrame in this example demonstrates its versatility in handling structured data for both analysis and visualization tasks.

20. Creating a Circular Packing Plot of Hierarchical Data

Importance★★★★☆

Difficulty★★★★☆

A client in the e-commerce sector wants to visualize their product category hierarchy using a circular packing plot. They have provided you with hierarchical data representing the main categories, subcategories, and the number of products in each subcategory. Your task is to generate a circular packing plot to help them better understand the distribution and relationships within their product categories. You will need to create sample data for this exercise. The data should have the following structure: Category A Subcategory A1: 150 products Subcategory A2: 100 products Category B Subcategory B1: 200 products Subcategory B2: 50 products Subcategory B3: 75 products Category C Subcategory C1: 300 products Use Python to generate this data and create a circular packing plot.

【Data Generation Code Example】

```
import pandas as pd
# Creating hierarchical data for the circular packing plot
data = {
    'Category': ['A', 'A', 'B', 'B', 'B', 'C'],
    'Subcategory': ['A1', 'A2', 'B1', 'B2', 'B3', 'C1'],
    'Products': [150, 100, 200, 50, 75, 300]
}
# Convert to DataFrame
df = pd.DataFrame(data)
df
```


【Diagram Answer】

Circular Packing Plot of Product Categories



【Code Answer】

```
import pandas as pd
import matplotlib.pyplot as plt
import squarify
# Creating hierarchical data for the circular packing plot
data = {
    'Category': ['A', 'A', 'B', 'B', 'B', 'C'],
    'Subcategory': ['A1', 'A2', 'B1', 'B2', 'B3', 'C1'],
    'Products': [150, 100, 200, 50, 75, 300]}
```

```

}
# Convert to DataFrame
df = pd.DataFrame(data)
# Aggregate data for circular packing plot
agg_data = df.groupby(['Category',
'Subcategory']).sum().reset_index()
# Create sizes and labels for the plot
sizes = agg_data['Products']
labels = agg_data['Subcategory'] + ' (' +
agg_data['Products'].astype(str) + ')'
# Create the circular packing plot
fig, ax = plt.subplots(figsize=(10, 8))
squarify.plot(sizes=sizes, label=labels, alpha=0.7, ax=ax)
ax.axis('off')
plt.title('Circular Packing Plot of Product Categories')
plt.show()

```

In this exercise, you are required to create a circular packing plot using hierarchical data provided by a client.

First, you need to generate the sample data. We create a dictionary containing the main categories, subcategories, and the number of products in each subcategory.

This data is then converted into a pandas DataFrame for easier manipulation.

Next, we aggregate the data to prepare it for plotting.

We group the data by the 'Category' and 'Subcategory' columns, summing the 'Products' to get the total number of products for each subcategory.

For the circular packing plot, we use the squarify library, which is typically used for treemap visualizations but works well for this type of plot too.

We prepare the sizes and labels for the plot, which are derived from the aggregated data.

The sizes variable holds the number of products in each subcategory, and the labels variable is a combination of the subcategory names and their respective product counts.

We then create a plot using `squarify.plot`, passing the sizes and labels. We set the alpha parameter to 0.7 for transparency and remove the axis for better visualization.

Finally, we set the title and display the plot using `plt.show()`. This process helps visualize the distribution and relationships within the product categories, making it easier for the client to understand their data.

【Trivia】

Circular packing plots are a variation of treemaps that display hierarchical data in a circular layout. They are useful for visualizing proportions within categories in a more aesthetically pleasing and space-efficient manner.

While `squarify` is commonly used for treemaps, it can be adapted for circular packing plots with some customization, demonstrating the flexibility of visualization tools in Python.

21. Generating a Beeswarm Plot of Distribution Data

Importance★★★★☆

Difficulty★★★★☆

You are working for a market research company. Your task is to visualize the distribution of customer satisfaction scores for different product categories. The satisfaction scores range from 1 to 10, and you have data for three categories: Electronics, Clothing, and Groceries.

Generate a beeswarm plot to show the distribution of satisfaction scores for these categories.

Use the provided code to create the input data and then write the code to generate the plot.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
np.random.seed(42) # For reproducibility
categories = ['Electronics', 'Clothing', 'Groceries']
data = pd.DataFrame({
    'Category': np.random.choice(categories, 300),
    'Score': np.random.randint(1, 11, 300)
})
print(data.head())
```

【Diagram Answer】



【Code Answer】

```
import numpy as np # Import numpy for numerical
operations # Set a random seed for reproducibility
import pandas as pd # Import pandas for data manipulation
import matplotlib.pyplot as plt # Import matplotlib for
plotting
import seaborn as sns # Import seaborn for advanced
plottingnp.random.seed(42) # For reproducibilitycategories
= ['Electronics', 'Clothing', 'Groceries'] # Define product
categories
data = pd.DataFrame({ # Create a DataFrame with random
categories and scores
'Category': np.random.choice(categories, 300),
'Score': np.random.randint(1, 11, 300)
})plt.figure(figsize=(10, 6)) # Set the figure size
```

```
sns.swarmplot(x='Category', y='Score', data=data) #  
Create a beeswarm plot  
plt.title('Distribution of Customer Satisfaction Scores') # Set  
the title  
plt.xlabel('Product Category') # Set the x-axis label  
plt.ylabel('Satisfaction Score') # Set the y-axis label  
plt.show() # Display the plot
```

To create a beeswarm plot in Python, you need to use the seaborn library, which builds on top of matplotlib. First, import the necessary libraries: numpy for numerical operations, pandas for data manipulation, matplotlib for plotting, and seaborn for advanced plotting functions. Set a random seed using `np.random.seed(42)` to ensure that the random numbers generated are reproducible. Define the product categories as a list: `categories = ['Electronics', 'Clothing', 'Groceries']`. Next, create a DataFrame `data` with random categories and satisfaction scores using `np.random.choice` for the categories and `np.random.randint` for the scores. To generate the plot, set the figure size using `plt.figure(figsize=(10, 6))`. Use the `sns.swarmplot` function to create the beeswarm plot, specifying the x-axis as the 'Category' column and the y-axis as the 'Score' column of your DataFrame. Add a title and axis labels using `plt.title`, `plt.xlabel`, and `plt.ylabel` to make the plot more informative. Finally, use `plt.show()` to display the plot. This process demonstrates how to manipulate data and visualize distributions using Python, helping you understand how to create informative and visually appealing plots.

【Trivia】

- ▶ Beeswarm plots are particularly useful for displaying all data points to show the distribution and density without overlapping, unlike box plots or histograms.
- ▶ The seaborn library, built on top of matplotlib, simplifies complex visualizations and provides an aesthetically pleasing default style.
- ▶ Random seeds in numpy ensure that the random number generation is reproducible, which is crucial for consistent data analysis results.

22. Joy Plot of Distribution Data

Importance★★★★☆

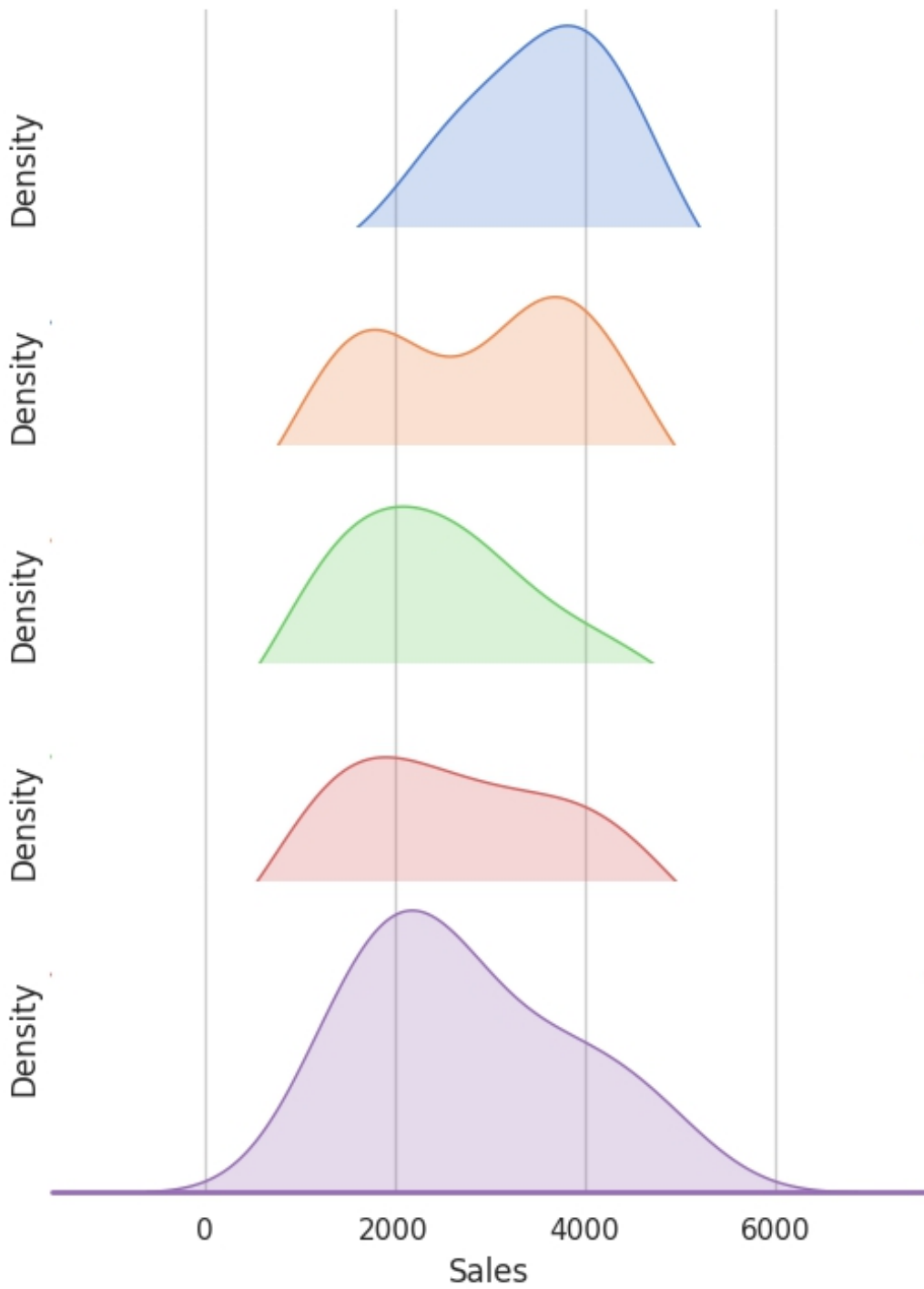
Difficulty★★★★☆

You are working as a data analyst for a retail company. Your task is to visualize the distribution of monthly sales data across different regions to identify patterns and trends. You decide to use a Joy Plot (also known as a Ridge Plot) to display the distribution of sales data for each region. Generate sample sales data for five regions over 12 months and create a Joy Plot to visualize the distributions.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
regions = ['North', 'South', 'East', 'West', 'Central']
#Create a dictionary to store sales data
data = {'Month': np.tile(np.arange(1, 13), len(regions)),
        'Region': np.repeat(regions, 12),
        'Sales': np.random.randint(1000, 5000, 12 * len(regions))}
#Convert dictionary to DataFrame
df = pd.DataFrame(data)
df
```


【Diagram Answer】



【Code Answer】



```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#Create sample data
regions = ['North', 'South', 'East', 'West', 'Central']
data = {'Month': np.tile(np.arange(1, 13), len(regions)),
        'Region': np.repeat(regions, 12),
        'Sales': np.random.randint(1000, 5000, 12 * len(regions))}
df = pd.DataFrame(data)
#Set up the plot
sns.set(style='whitegrid')
g = sns.FacetGrid(df, row='Region', hue='Region',
                  aspect=4, height=1.5, palette='muted')
#Plot each distribution
g.map(sns.kdeplot, 'Sales', fill=True)
g.map(plt.axhline, y=0, lw=2, clip_on=False)
g.fig.subplots_adjust(hspace=-0.3)
#Remove axes details that do not play well with overlap
g.set_titles("")
g.set(yticks=[])
g.despine(bottom=True, left=True)
#Show plot
plt.show()

```

The first step involves importing necessary libraries: numpy, pandas, matplotlib, and seaborn. These libraries are essential for data manipulation and visualization.

Next, sample sales data is generated using numpy to create arrays of months and random sales values for different regions.

The data is organized into a dictionary and then converted into a pandas DataFrame.

This DataFrame will be used for plotting.

The visualization is set up using seaborn, a high-level interface for drawing attractive statistical graphics.

The FacetGrid function is used to create a grid for plotting the distributions of sales data by region.

The kdeplot function from seaborn is employed to plot kernel density estimates for each region, with the fill=True parameter to fill the area under the curves.

Some aesthetic adjustments are made to the plot: hiding axis titles, removing y-axis ticks, and despine to remove the top and right spines for a cleaner look.

The subplots_adjust method is used to adjust the spacing between the plots to achieve the overlapping effect characteristic of Joy Plots.

Finally, plt.show() is called to display the plot. This step renders the Joy Plot, showing the distribution of sales data for each region across different months.

【Trivia】

Joy Plots are named after the iconic album cover of Joy Division's "Unknown Pleasures," which features a series of stacked line plots.

These plots are particularly useful for visualizing the distribution of data across multiple categories or time periods.

23. Heatmap of Correlation Matrix

Importance★★★★☆

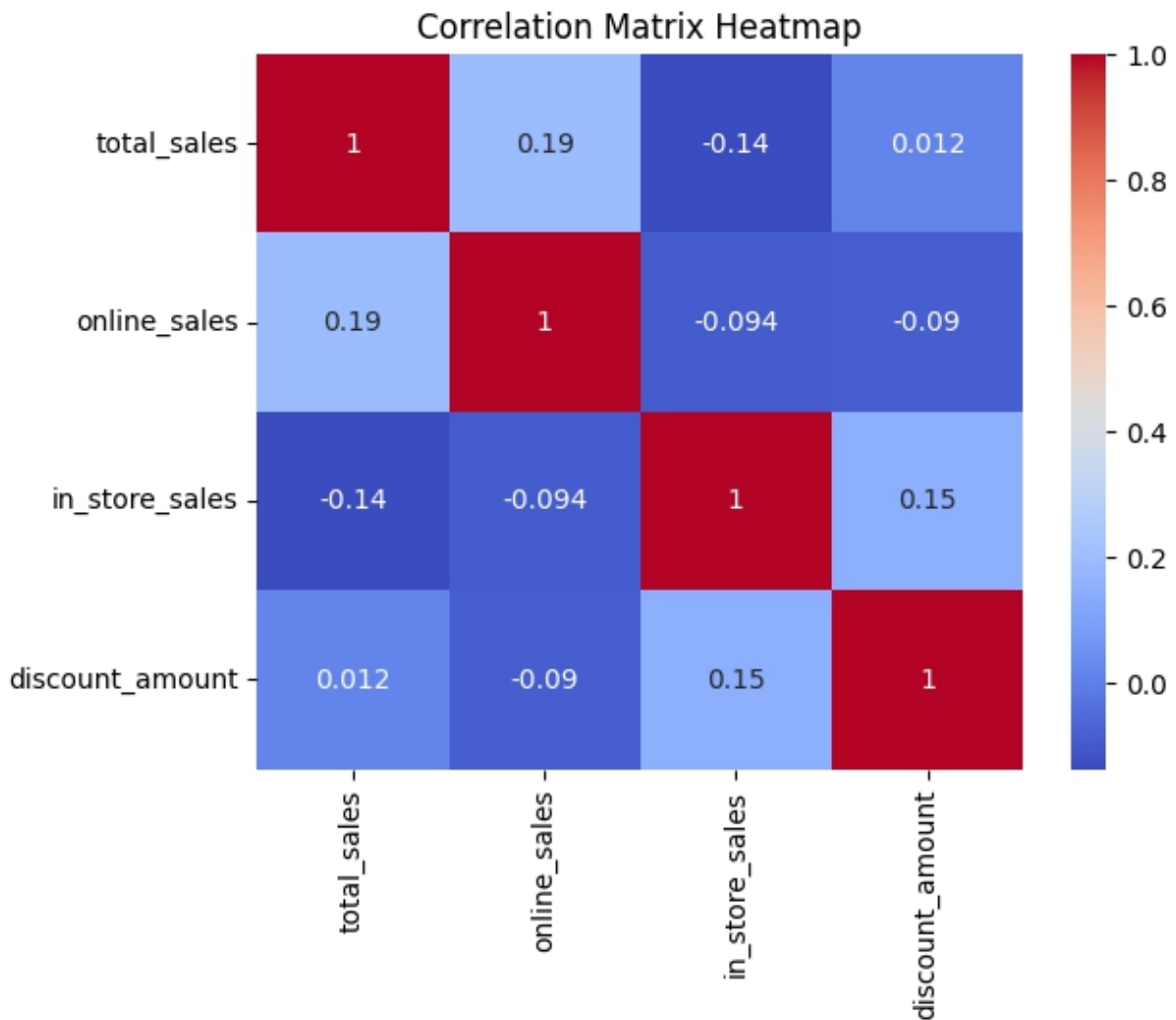
Difficulty★★☆☆☆

You are an analyst at a retail company. Your task is to understand the relationship between different sales metrics. Create a heatmap of the correlation matrix for the following sales data: total sales, online sales, in-store sales, and discount amount. The goal is to visualize the correlations to help the team understand how these metrics relate to each other. Write a Python code to generate this heatmap.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#Create a DataFrame with sales data
data = {
'total_sales': np.random.randint(1000, 5000, 100),
'online_sales': np.random.randint(200, 3000, 100),
'in_store_sales': np.random.randint(500, 4000, 100),
'discount_amount': np.random.uniform(5, 50, 100)
}
df = pd.DataFrame(data)
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#Create a DataFrame with sales data
df = pd.DataFrame({
'total_sales': np.random.randint(1000, 5000, 100),
```

```
'online_sales': np.random.randint(200, 3000, 100),
'in_store_sales': np.random.randint(500, 4000, 100),
'discount_amount': np.random.uniform(5, 50, 100)
})
#Calculate the correlation matrix
corr_matrix = df.corr()
#Create a heatmap
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```

The code begins by importing necessary libraries: pandas for data manipulation, numpy for numerical operations, seaborn for data visualization, and matplotlib for plotting. A DataFrame `df` is created with four columns representing different sales metrics: total sales, online sales, in-store sales, and discount amount. Random values are generated for each column to simulate sales data.

The correlation matrix of the DataFrame is calculated using the `.corr()` method. This matrix shows the pairwise correlation coefficients between the columns, indicating how strongly they are related.

Seaborn's heatmap function is used to create a heatmap of the correlation matrix. The `annot=True` parameter adds the correlation values to the heatmap cells, and `cmap='coolwarm'` specifies the color scheme. The plot is titled 'Correlation Matrix Heatmap' using `plt.title()`, and `plt.show()` displays the heatmap.

【Trivia】

A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. The value is in the

range of -1 to 1. A value closer to 1 implies a strong positive correlation, while a value closer to -1 implies a strong negative correlation. A value around 0 implies no correlation. Heatmaps are useful for visualizing the strength and direction of correlations in large datasets.

24. Generating Pair Grid Plot for Customer Satisfaction Analysis

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst working for a retail company. The company has collected customer satisfaction data across multiple dimensions for their new product line. Your task is to create a pair grid plot to visualize the relationships between different satisfaction metrics. The dataset includes the following variables:
Overall Satisfaction (1-10 scale)
Product Quality (1-10 scale)
Customer Service (1-10 scale)
Price Satisfaction (1-10 scale)
Likelihood to Recommend (1-10 scale)
Create a pair grid plot using seaborn to visualize the relationships between these variables. The plot should include scatter plots for each pair of variables and histograms for each individual variable. Use different colors for each variable to enhance readability. Your code should generate the sample data within the script and produce the pair grid plot without requiring any external data files.

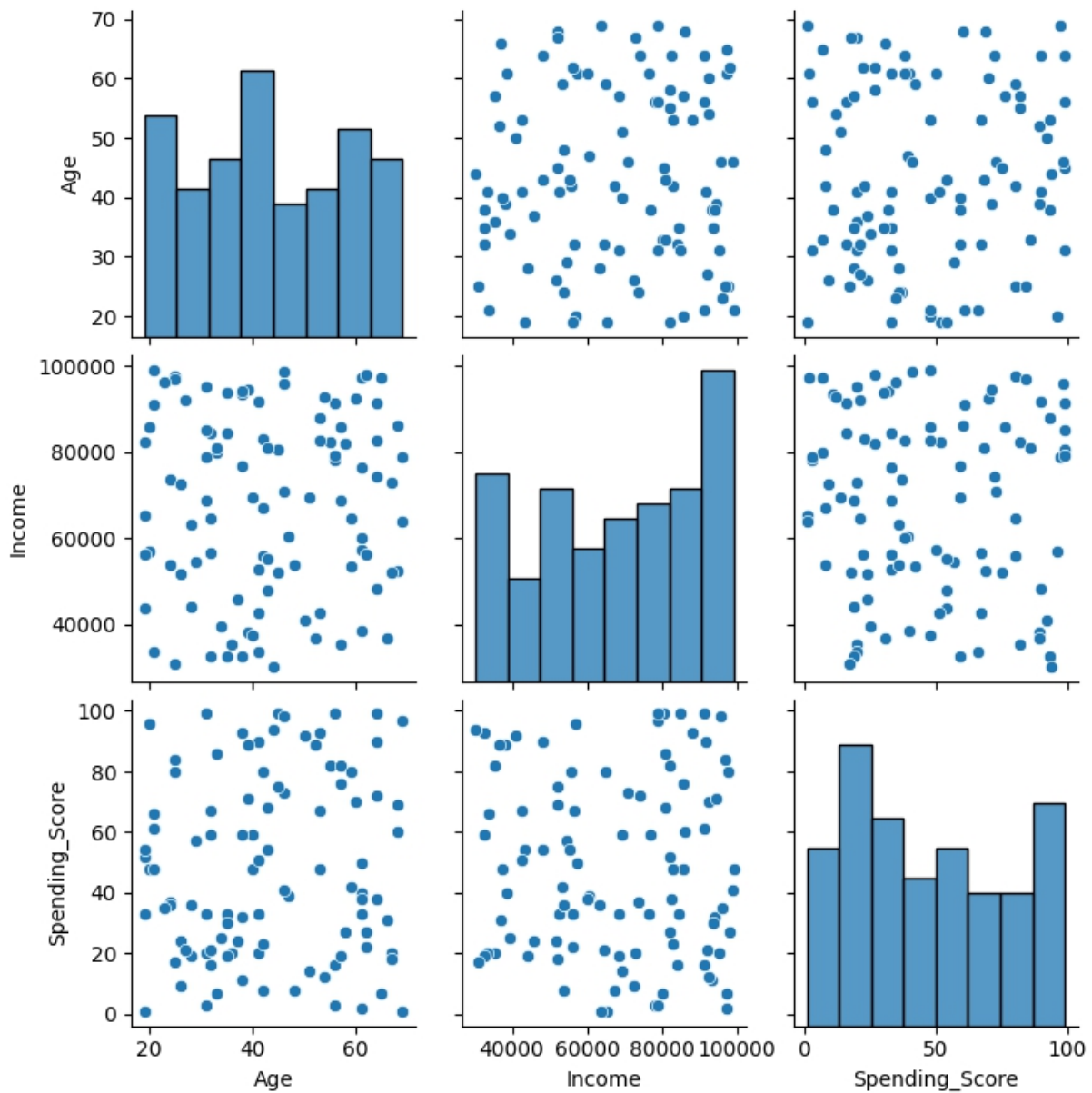
【Data Generation Code Example】

```
import numpy as np
np.random.seed(42)
n_samples = 200
overall_satisfaction = np.random.randint(1, 11, n_samples)
product_quality = np.random.randint(1, 11, n_samples)
customer_service = np.random.randint(1, 11, n_samples)
```



```
price_satisfaction = np.random.randint(1, 11, n_samples)
likelihood_to_recommend = np.random.randint(1, 11,
n_samples)
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
np.random.seed(42)
n_samples = 200
data = pd.DataFrame({
    'Overall Satisfaction': np.random.randint(1, 11, n_samples),
    'Product Quality': np.random.randint(1, 11, n_samples),
    'Customer Service': np.random.randint(1, 11, n_samples),
    'Price Satisfaction': np.random.randint(1, 11, n_samples),
    'Likelihood to Recommend': np.random.randint(1, 11,
n_samples)
})
sns.set(style="ticks", color_codes=True)
g = sns.PairGrid(data)
g.map_diag(plt.hist)
g.map_offdiag(plt.scatter)
g.add_legend()
plt.tight_layout()
plt.show()
```

This code generates a pair grid plot to visualize relationships between different customer satisfaction metrics.

Let's break down the code and explain its key components:

Data Generation:

We use NumPy to generate random data for our satisfaction metrics.

We create a pandas DataFrame with 200 samples and 5 variables.

Each variable is randomly generated with values between 1 and 10.

Data Visualization:

We use seaborn, a statistical data visualization library built on top of matplotlib.

The `sns.PairGrid(data)` function creates a grid of axes for plotting pairwise relationships in the dataset.

Plotting:

`g.map_diag(plt.hist)` plots histograms on the diagonal axes, showing the distribution of each variable.

`g.map_offdiag(plt.scatter)` creates scatter plots for each pair of variables on the off-diagonal axes.

This allows us to see both the distribution of individual variables and the relationships between pairs of variables.

Customization:

`sns.set(style="ticks", color_codes=True)` sets the visual style of the plot.

`g.add_legend()` adds a legend to the plot for better interpretation.

`plt.tight_layout()` adjusts the plot layout to prevent overlapping.

Display:

`plt.show()` displays the final plot.

This pair grid plot is particularly useful for exploring multivariate data.

It allows us to quickly identify patterns, correlations, and distributions across multiple variables simultaneously.

For example, we can easily spot if there's a strong correlation between overall satisfaction and likelihood to recommend, or if product quality has a uniform or skewed distribution.

The use of scatter plots for pairwise comparisons helps in identifying linear or non-linear relationships between variables, while the histograms on the diagonal provide insight into the distribution of each individual metric.

This visualization technique is extremely valuable in customer satisfaction analysis as it provides a

comprehensive overview of how different aspects of customer experience relate to each other, helping businesses identify areas for improvement and understand the factors that most strongly influence overall customer satisfaction.

【Trivia】

- ▶ Pair grid plots, also known as scatterplot matrices or SPLOM, were introduced by John W. Tukey and Paul A. Tukey in 1981.
- ▶ Seaborn, the library used for creating this plot, is named after a character from the novel "The Voyage of the Dawn Treader" by C.S. Lewis.
- ▶ The concept of using a matrix of scatterplots to visualize multivariate data relationships dates back to 1920s, but it became more popular with the advent of computer graphics in the 1970s and 1980s.
- ▶ Pair grid plots are particularly useful in exploratory data analysis (EDA), a concept popularized by John Tukey in his 1977 book "Exploratory Data Analysis."
- ▶ The diagonal elements in a pair grid plot can be customized to show various types of univariate plots, not just histograms. Common alternatives include kernel density estimates or box plots.
- ▶ In large datasets with many variables, pair grid plots can become cluttered. Advanced techniques like hierarchical clustering can be used to order the variables and reveal patterns more effectively.
- ▶ The seaborn library used in this example is built on top of matplotlib, which in turn was inspired by MATLAB's plotting capabilities.
- ▶ Pair grid plots are not limited to continuous variables. They can also be used with categorical data, where the off-diagonal plots might show box plots or violin plots instead of scatter plots.

25. Facet Grid Plot of Categorical Data

Importance★★★★☆

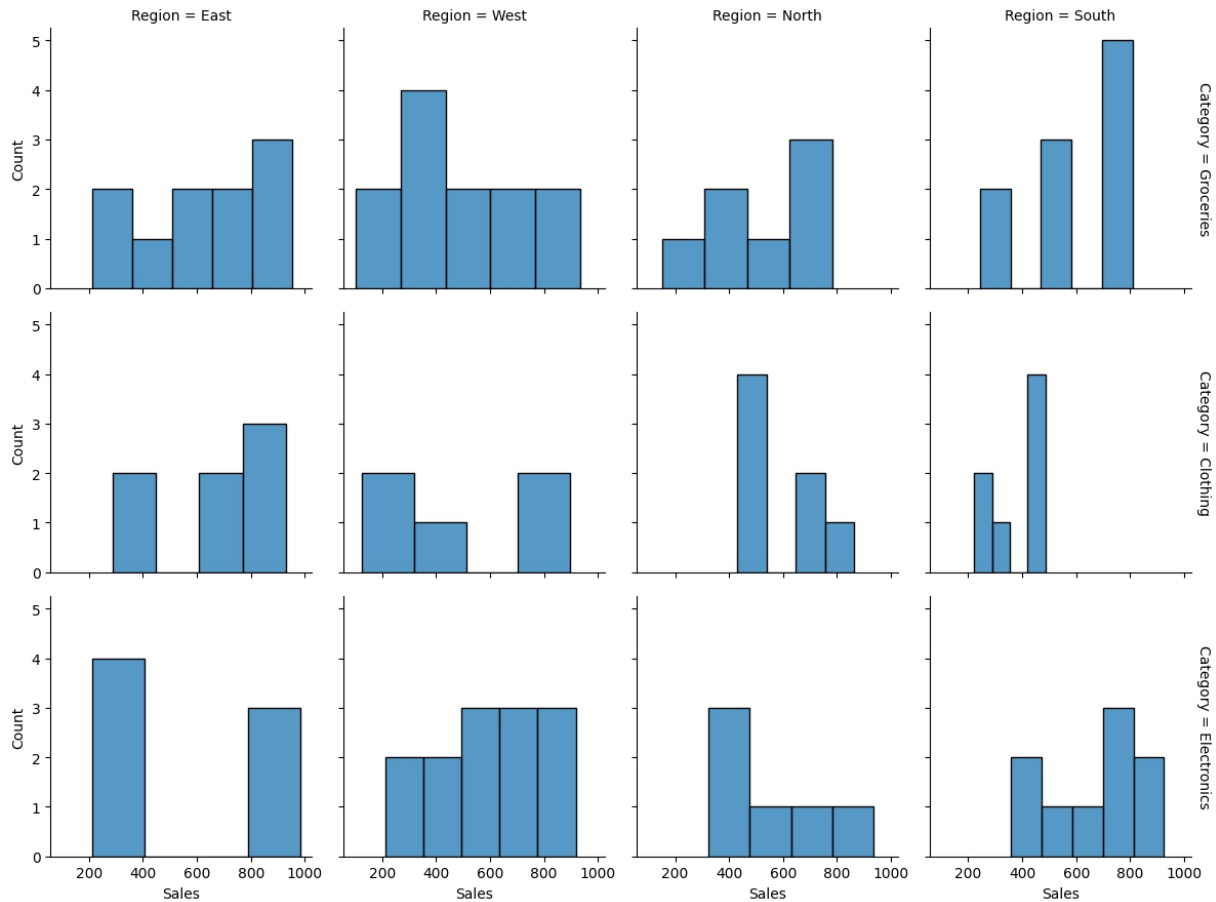
Difficulty★★★★☆

A retail company wants to analyze the sales performance across different regions and product categories. They have data on sales amounts, regions, and product categories. Your task is to create a Facet Grid plot that visualizes the distribution of sales amounts across these categories and regions. Use the provided sample data to generate the plot. Ensure the plot is clear and informative.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
# Generate sample data
np.random.seed(42)
regions = ['North', 'South', 'East', 'West']
categories = ['Electronics', 'Clothing', 'Groceries']
data = {
    'Region': [regions[np.random.randint(0, 4)] for _ in
range(100)],
    'Category': [categories[np.random.randint(0, 3)] for _ in
range(100)],
    'Sales': np.random.randint(100, 1000, 100)
}
df = pd.DataFrame(data)
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
# Generate sample data
np.random.seed(42)
regions = ['North', 'South', 'East', 'West']
categories = ['Electronics', 'Clothing', 'Groceries']
data = {
```

```

'Region': [regions[np.random.randint(0, 4)] for _ in
range(100)],
'Category': [categories[np.random.randint(0, 3)] for _ in
range(100)],
'Sales': np.random.randint(100, 1000, 100)
}
df = pd.DataFrame(data)
# Create the FacetGrid plot
g = sns.FacetGrid(df, col="Region", row="Category",
margin_titles=True)
g.map(sns.histplot, "Sales")
plt.show()

```

To create a Facet Grid plot of categorical data, you first need to generate a sample dataset.

In this example, the dataset consists of sales data categorized by regions and product categories.

The data is generated using NumPy's random functions to ensure a diverse and random distribution of values.

The pandas library is used to create a DataFrame from the generated data.

The core of the visualization process involves using Seaborn's FacetGrid class.

A FacetGrid allows you to create a grid of plots based on the values of categorical variables.

In this case, the grid is defined by the 'Region' and 'Category' columns in the dataset.

Each cell in the grid represents a combination of a region and a category, and the sales data is visualized within each cell.

The map method is used to apply a specific plotting function (in this case, sns.histplot) to each cell in the grid.

Finally, `plt.show()` is called to display the plot. This approach provides a clear and organized way to visualize how sales amounts are distributed across different regions and product categories, making it easier to identify patterns and trends in the data.

【Trivia】

- ▶ Seaborn's `FacetGrid` is particularly useful for visualizing complex relationships in multi-dimensional categorical data.
- ▶ The `FacetGrid` can be customized extensively, including adjusting the size of the grid, adding titles, and modifying the appearance of the plots.
- ▶ Histograms, scatter plots, and other types of plots can be used within the `FacetGrid` to represent the data in various ways.
- ▶ The `margin_titles` parameter in `FacetGrid` helps in displaying the titles of the rows and columns on the margins, making the grid more readable.

26. Plotting a Linear Regression

Importance★★★★☆

Difficulty★★★★☆☆

You are working as a data analyst for a retail company. The company wants to understand the relationship between advertising spending and sales.

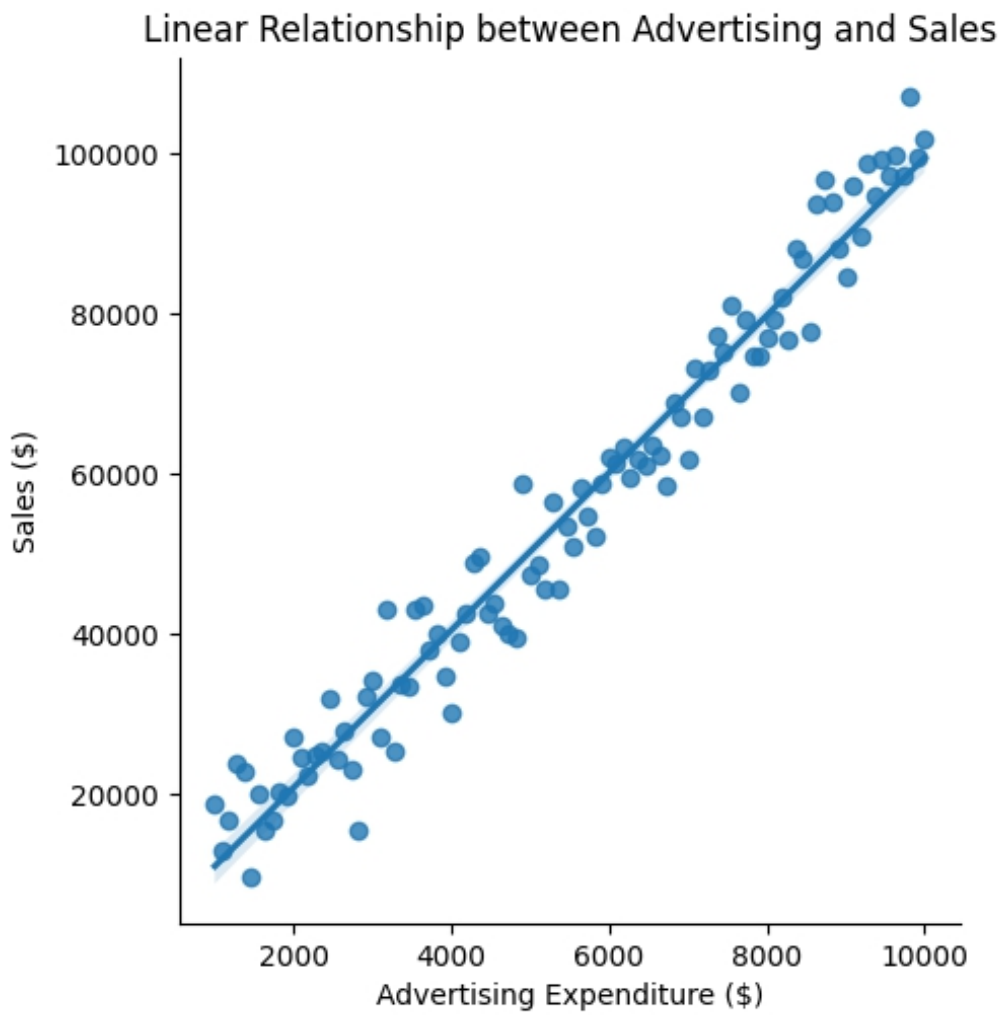
Your task is to create a scatter plot of the given data and fit a linear regression line to visualize this relationship.

Generate the data within the code, create the plot, and display it. Use Python's data processing and visualization libraries to achieve this.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
Generate sample data np.random.seed(0)
advertising_spend = np.random.uniform(1000, 5000, 100)
sales = 5 * advertising_spend + np.random.normal(0, 1000, 100)
Create DataFrame data = pd.DataFrame({'Advertising Spend': advertising_spend, 'Sales': sales})
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
Generate sample data np.random.seed(0)
advertising_spend = np.random.uniform(1000, 5000, 100)
```

```

sales = 5 * advertising_spend + np.random.normal(0, 1000,
100)
Create DataFrame data = pd.DataFrame({'Advertising
Spend': advertising_spend, 'Sales': sales})
Fit linear regression model model = LinearRegression()
model.fit(data[['Advertising Spend']], data['Sales'])
sales_pred = model.predict(data[['Advertising Spend']])
Plot data and regression line plt.figure(figsize=(10, 6))
sns.scatterplot(x='Advertising Spend', y='Sales',
data=data)
plt.plot(data['Advertising Spend'], sales_pred, color='red',
label='Linear Regression Line')
plt.xlabel('Advertising Spend')
plt.ylabel('Sales')
plt.title('Linear Relationship between Advertising Spend and
Sales')
plt.legend()
plt.show()

```

First, we import the necessary libraries: numpy, pandas, matplotlib, seaborn, and sklearn.

numpy is used for numerical operations, pandas for data manipulation, matplotlib and seaborn for plotting, and sklearn for the linear regression model.

We generate random data for advertising spending using numpy's uniform function, which creates 100 data points between 1000 and 5000.

To simulate sales data, we assume a linear relationship where sales are five times the advertising spend plus some random noise.

This noise is added using numpy's normal function to make the data more realistic.

We store the generated data in a pandas DataFrame for easier manipulation and plotting.

Next, we fit a linear regression model using sklearn's LinearRegression class.

We train the model with the advertising spend as the predictor variable and sales as the target variable.

After fitting the model, we use it to predict sales based on the advertising spend.

For visualization, we create a scatter plot of the advertising spend against sales using seaborn's scatterplot function.

We also plot the regression line using matplotlib's plot function, setting the x-values as the advertising spend and the y-values as the predicted sales.

Labels and a title are added for clarity, and we use the legend to differentiate between the data points and the regression line.

Finally, we display the plot using `plt.show()`.

【Trivia】

Linear regression is one of the simplest and most commonly used machine learning algorithms.

It assumes a linear relationship between the input variables (independent variables) and the single output variable (dependent variable).

The goal is to find the line that best fits the data, minimizing the sum of the squared differences between the observed and predicted values.

This method is highly interpretable, making it valuable for understanding relationships between variables in various fields such as economics, biology, and social sciences.

27. Creating a Residual Plot for Regression Analysis

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst working for a real estate company. The company wants to understand the relationship between house prices and their square footage.

They have collected data on recent house sales, including the price and square footage of each house.

Your task is to create a linear regression model to predict house prices based on square footage, and then create a residual plot to assess the model's performance.

Specifically, you need to:

Generate sample data for house prices and square footage.

Perform a linear regression analysis.

Create a residual plot to visualize the differences between the actual prices and the predicted prices.

Include appropriate labels and a title for the plot.

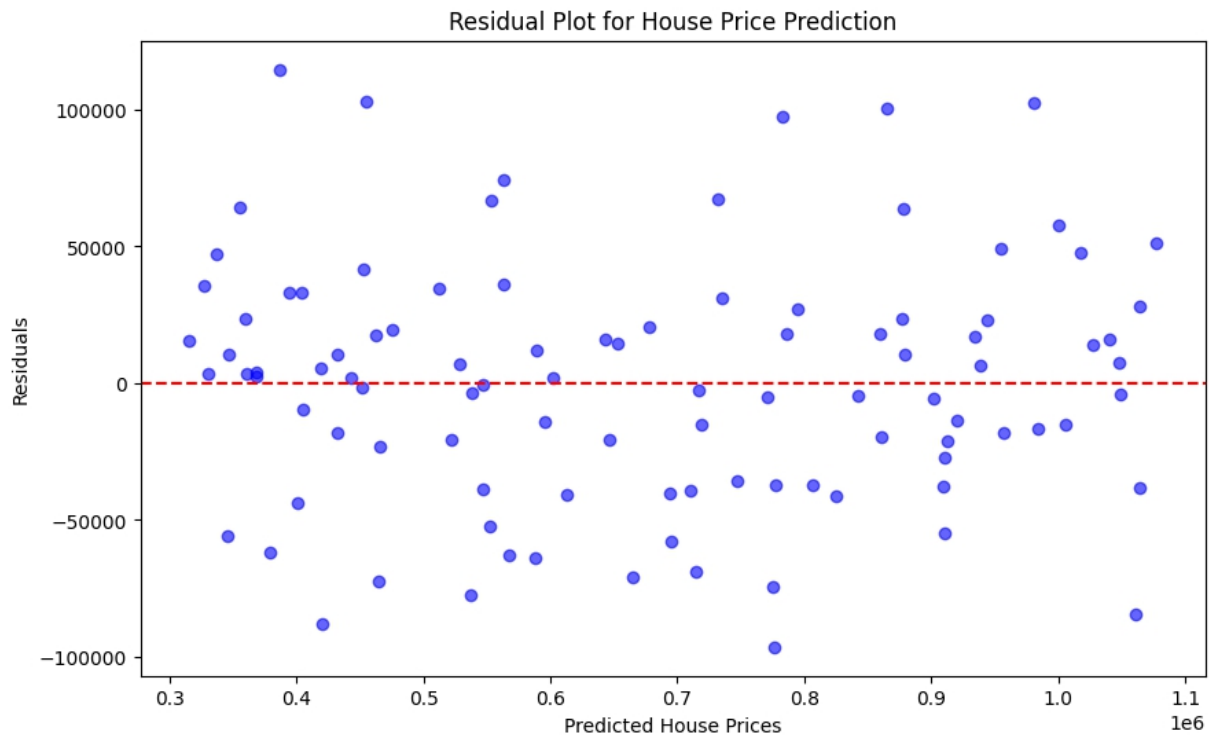
Write a Python script that accomplishes these tasks using libraries such as NumPy, Pandas, Matplotlib, and Scikit-learn.

Make sure to create the sample data within your code.

【Data Generation Code Example】

```
import numpy as np
np.random.seed(42)
square_footage = np.random.uniform(1000, 5000, 100)
price = 100000 + 200 * square_footage +
np.random.normal(0, 50000, 100)
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
np.random.seed(42)
square_footage = np.random.uniform(1000, 5000, 100)
price = 100000 + 200 * square_footage +
np.random.normal(0, 50000, 100)
df = pd.DataFrame({'Square Footage': square_footage,
'Price': price})
X = df[['Square Footage']]
y = df['Price']
```

```
model = LinearRegression()
model.fit(X, y)
predictions = model.predict(X)
residuals = y - predictions
plt.figure(figsize=(10, 6))
plt.scatter(predictions, residuals, color='blue', alpha=0.6)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted House Prices')
plt.ylabel('Residuals')
plt.title('Residual Plot for House Price Prediction')
plt.show()
```

This code demonstrates how to create a residual plot for a regression analysis using Python.

Let's break down the process step by step:

Data Generation:

We start by generating sample data for our analysis.

We create two arrays: 'square_footage' and 'price'.

The 'square_footage' array contains random values between 1000 and 5000, representing the square footage of houses.

The 'price' array is calculated based on a linear relationship with square footage, plus some random noise to simulate real-world variability.

Data Preparation:

We create a pandas DataFrame from our generated data.

This step organizes our data into a structured format, making it easier to work with.

Linear Regression:

We use scikit-learn's LinearRegression class to create and fit our model.

We separate our data into features (X) and target (y).
In this case, 'Square Footage' is our feature, and 'Price' is our target.

The 'fit' method trains the model on our data.

Making Predictions:

We use the trained model to make predictions on our input data.

These predictions represent the estimated house prices based on square footage.

Calculating Residuals:

Residuals are the differences between the actual values (y) and the predicted values.

We calculate these by subtracting the predictions from the actual prices.

Creating the Residual Plot:

We use matplotlib to create the residual plot.

Here's what each part of the plotting code does:

`plt.figure(figsize=(10, 6))`: Sets the size of the plot.

`plt.scatter(predictions, residuals, color='blue', alpha=0.6)`:
Creates a scatter plot of predicted prices vs. residuals.

The alpha parameter sets the transparency of the points.

`plt.axhline(y=0, color='red', linestyle='--')`: Adds a horizontal line at $y=0$ to help visualize the distribution of residuals above and below zero.

`plt.xlabel()` and `plt.ylabel()`: Add labels to the x and y axes.

`plt.title()`: Adds a title to the plot.

`plt.show()`: Displays the plot.

The resulting residual plot allows us to visually assess the performance of our regression model.

In an ideal scenario, the residuals should be randomly scattered around the horizontal line at $y=0$.

Any patterns in the residual plot could indicate issues with the model, such as non-linearity or heteroscedasticity.

This exercise demonstrates key skills in Python data manipulation and visualization, including working with NumPy for numerical operations, Pandas for data structuring, scikit-learn for machine learning tasks, and matplotlib for creating informative visualizations.

【Trivia】

- ▶ Residual plots are crucial tools in regression analysis for checking the assumptions of linear regression.
- ▶ A perfect residual plot would show a random scatter of points with no discernible pattern.
- ▶ Patterns in residual plots can indicate issues like non-linearity, heteroscedasticity, or the need for additional predictor variables.
- ▶ The scikit-learn library used in this example is one of the most popular machine learning libraries in Python.
- ▶ In real estate analysis, factors beyond square footage (like location, age of the house, etc.) would typically be included for more accurate price predictions.
- ▶ The `numpy.random.seed()` function is used to ensure reproducibility of random number generation, which is important for consistent results in data science projects.
- ▶ The `alpha` parameter in `plt.scatter()` controls the transparency of points, which can be useful for visualizing overlapping data points.

28. Categorical Plot of Survey Data

Importance★★★★☆

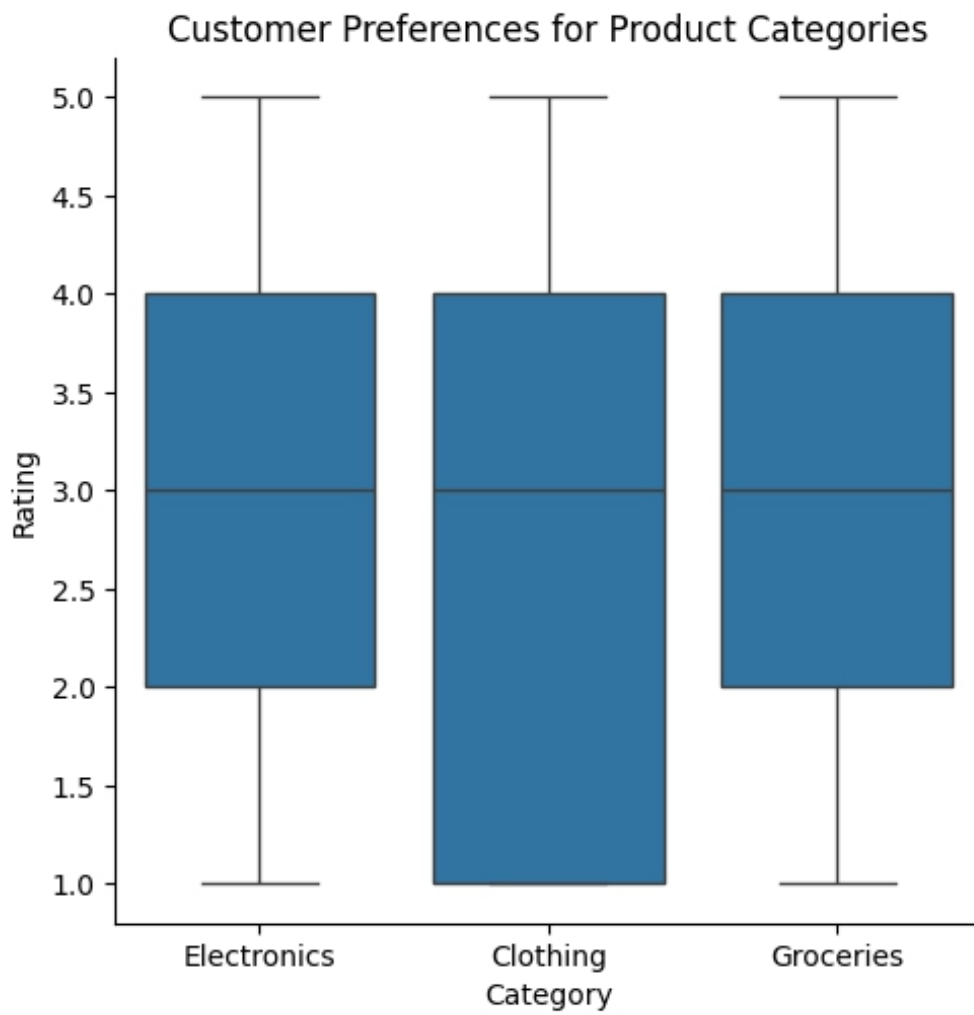
Difficulty★★★☆☆

You are working as a data analyst for a company that recently conducted a survey to understand customer preferences for different product categories. The survey results include responses from 100 customers, with each customer rating their preference for three product categories: 'Electronics', 'Clothing', and 'Groceries'. Your task is to generate a categorical plot to visualize the survey data. Create the input data within your code and ensure the plot is displayed.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(0)
data = pd.DataFrame({
    'CustomerID': range(1, 101),
    'Electronics': np.random.randint(1, 6, 100),
    'Clothing': np.random.randint(1, 6, 100),
    'Groceries': np.random.randint(1, 6, 100)
})
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
np.random.seed(0)
data = pd.DataFrame({
    'CustomerID': range(1, 101),
    'Electronics': np.random.randint(1, 6, 100),
```

```
'Clothing': np.random.randint(1, 6, 100),  
'Groceries': np.random.randint(1, 6, 100)  
)  
data_melted = pd.melt(data, id_vars=['CustomerID'],  
value_vars=['Electronics', 'Clothing', 'Groceries'],  
var_name='Category', value_name='Rating')  
sns.catplot(data=data_melted, kind='box', x='Category',  
y='Rating')  
plt.title('Customer Preferences for Product Categories')  
plt.show()
```

First, we import the necessary libraries: pandas for data manipulation, numpy for generating random data, seaborn for plotting, and matplotlib for displaying the plot.

We set a random seed using `np.random.seed(0)` to ensure reproducibility of the random data.

Next, we create a DataFrame named `data` with 100 rows, where each row represents a customer.

The DataFrame includes columns for 'CustomerID', 'Electronics', 'Clothing', and 'Groceries', with random integer values between 1 and 5 representing customer ratings.

We then use the `pd.melt` function to transform the DataFrame from wide format to long format.

This function takes the original DataFrame and creates a new DataFrame where each row represents a single rating for a specific category.

The `id_vars` parameter specifies the columns to keep as identifiers, and the `value_vars` parameter specifies the columns to unpivot.

The resulting DataFrame has three columns: 'CustomerID', 'Category', and 'Rating'.

Next, we use seaborn's `catplot` function to create a categorical plot.

We specify the data source as `data_melted`, the plot type as `'box'`, and the x and y axes as `'Category'` and `'Rating'`, respectively.

Finally, we set the plot title using `plt.title` and display the plot using `plt.show`.

【Trivia】

- ▶ Seaborn is built on top of matplotlib and provides a high-level interface for drawing attractive and informative statistical graphics.
- ▶ The `pd.melt` function is particularly useful for transforming data into a format suitable for plotting and analysis.
- ▶ Box plots are useful for visualizing the distribution of data and identifying outliers. They show the median, quartiles, and potential outliers in the data.
- ▶ Setting a random seed ensures that the random numbers generated are reproducible, which is crucial for debugging and sharing results.

29. Creating a Strip Plot of Categorical Data

Importance★★★★☆

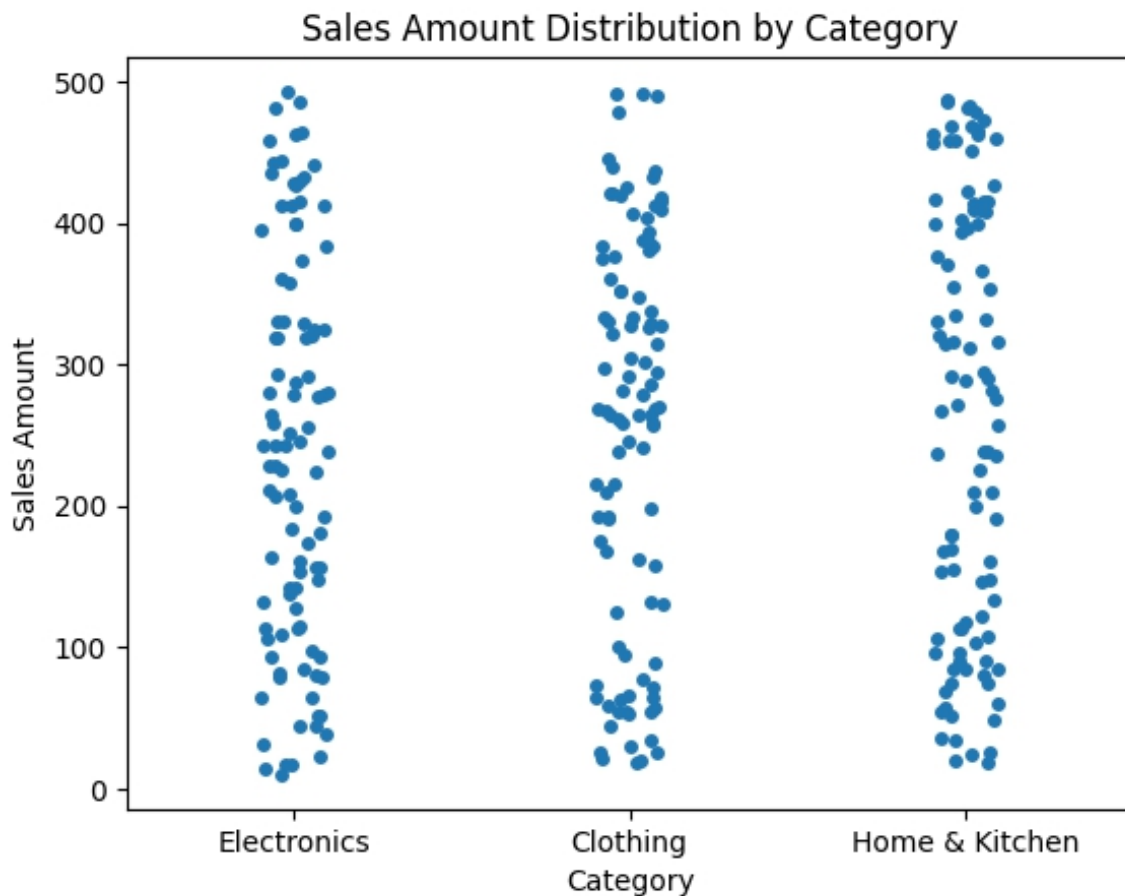
Difficulty★★★★☆

You are a data analyst at a retail company. Your manager has asked you to visualize the distribution of sales amounts across different product categories to identify any patterns or anomalies. Create a strip plot to display this information. Generate a sample dataset with the following structure:
Categories: 'Electronics', 'Clothing', 'Home & Kitchen'
Sales Amounts: Random values between 10 and 500
Use Python to create this visualization.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
categories = ['Electronics', 'Clothing', 'Home & Kitchen']
data = pd.DataFrame({
    'Category': [categories[i % 3] for i in range(300)],
    'Sales Amount': np.random.randint(10, 501, 300)
})
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
categories = ['Electronics', 'Clothing', 'Home & Kitchen']
data = pd.DataFrame({
    'Category': [categories[i % 3] for i in range(300)],
    'Sales Amount': np.random.randint(10, 501, 300)
})
sns.stripplot(x='Category', y='Sales Amount', data=data)
```



```
plt.title('Sales Amount Distribution by Category')
plt.xlabel('Category')
plt.ylabel('Sales Amount')
plt.show()
```

To create a strip plot of categorical data, we first need to generate a sample dataset.

We use the pandas library to create a DataFrame and numpy to generate random sales amounts.

The 'Category' column contains repeated values for 'Electronics', 'Clothing', and 'Home & Kitchen'.

The 'Sales Amount' column contains random integers between 10 and 500.

Next, we use the seaborn library to create the strip plot.

The `sns.stripplot` function takes the `x` and `y` parameters to define the categorical and numerical data, respectively.

We pass the DataFrame to the `data` parameter.

Finally, we use matplotlib's `plt` functions to add a title and labels to the plot and display it with `plt.show()`.

【Trivia】

- ▶ Strip plots are useful for visualizing the distribution of data points across different categories, especially when dealing with small datasets.
- ▶ Seaborn's `stripplot` function can be customized with various parameters, such as `jitter`, to improve the readability of overlapping points.
- ▶ Combining strip plots with other plots like box plots can provide more insights into the data distribution.

30. Swarm Plot of Distribution Data

Importance★★★★☆

Difficulty★★☆☆☆

You are a data analyst working for a company that wants to visualize the distribution of a particular dataset.

Your task is to create a swarm plot that shows the distribution of this data.

First, you need to generate a sample dataset with values following a normal distribution.

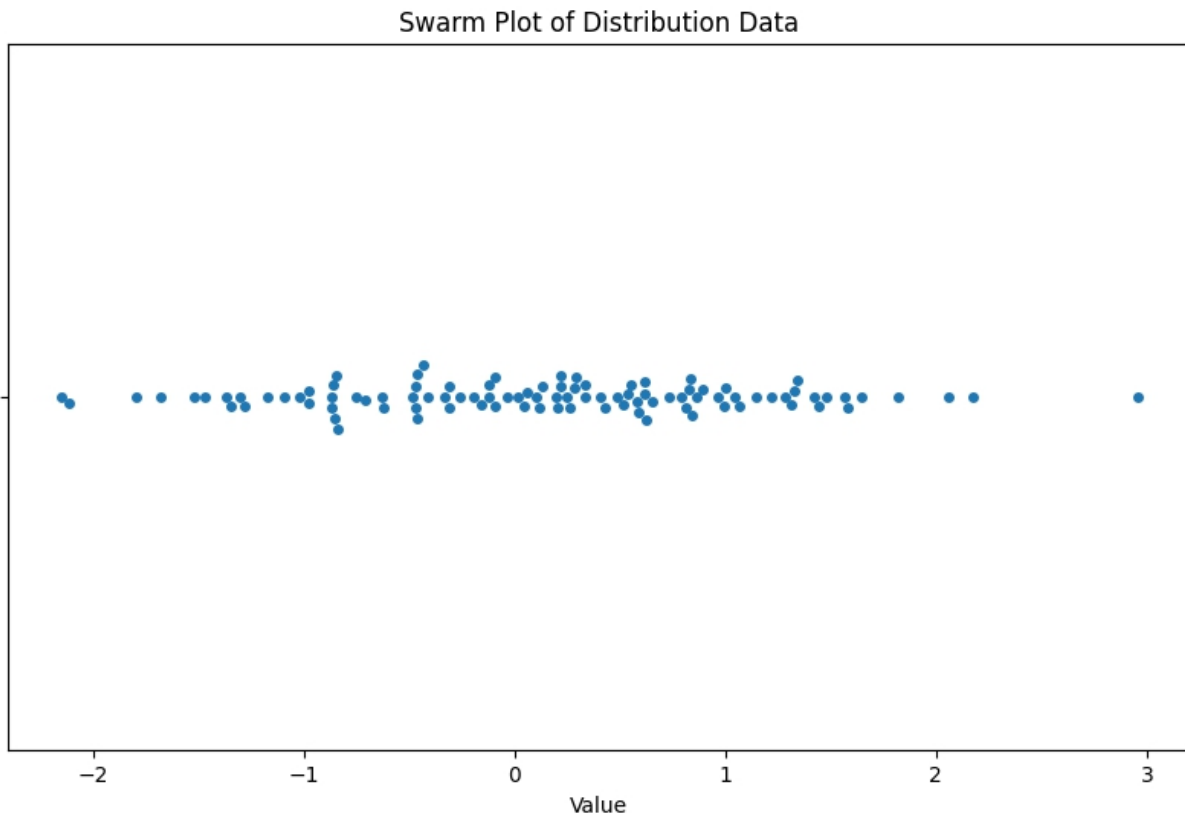
Then, using this dataset, create a swarm plot.

The plot should help in understanding the distribution of the data points.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
data = pd.DataFrame({ 'value': np.random.normal(0, 1,
100) })
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.DataFrame({ 'value': np.random.normal(0, 1,
100) })
plt.figure(figsize=(10, 6))
sns.swarmplot(x=data['value'])
plt.title('Swarm Plot of Distribution Data')
plt.xlabel('Value')
plt.show()
```

To create a swarm plot, we start by generating the data.

Here, we use the numpy library to create a normal distribution of 100 data points with a mean of 0 and a standard deviation of 1.

This data is stored in a pandas DataFrame.

Next, we import the necessary libraries for plotting: matplotlib.pyplot and seaborn.

Matplotlib is a fundamental library for creating static, animated, and interactive visualizations in Python.

Seaborn, built on top of matplotlib, provides a high-level interface for drawing attractive and informative statistical graphics.

We then use seaborn's swarmplot function to create the swarm plot.

The 'x' parameter is set to the 'value' column of our DataFrame.

This function positions each data point individually along the x-axis, ensuring they do not overlap, thus showing the distribution of values clearly.

Finally, we add a title and label to the x-axis for better understanding of the plot, and use plt.show() to display the plot.

【Trivia】

- ▶ Seaborn's swarm plot is particularly useful when you need to visualize all individual observations along with their distribution.
- ▶ Unlike strip plots, swarm plots adjust the positions of data points to avoid overlap, providing a better view of the data density.
- ▶ Seaborn also provides other similar plots like box plots and violin plots that can be used to show data distributions with additional statistical information.
- ▶ Combining swarm plots with other plot types can give a comprehensive view of data distributions and outliers.

31. Factor Plot of Categorical Data

Importance★★★★☆

Difficulty★★★★☆☆

A retail company wants to analyze customer satisfaction across different store locations.

The company conducted a survey, gathering satisfaction ratings (1 to 5) from customers at three different stores (Store A, Store B, Store C).

Your task is to create a factor plot to visualize the distribution of customer satisfaction ratings across these stores.

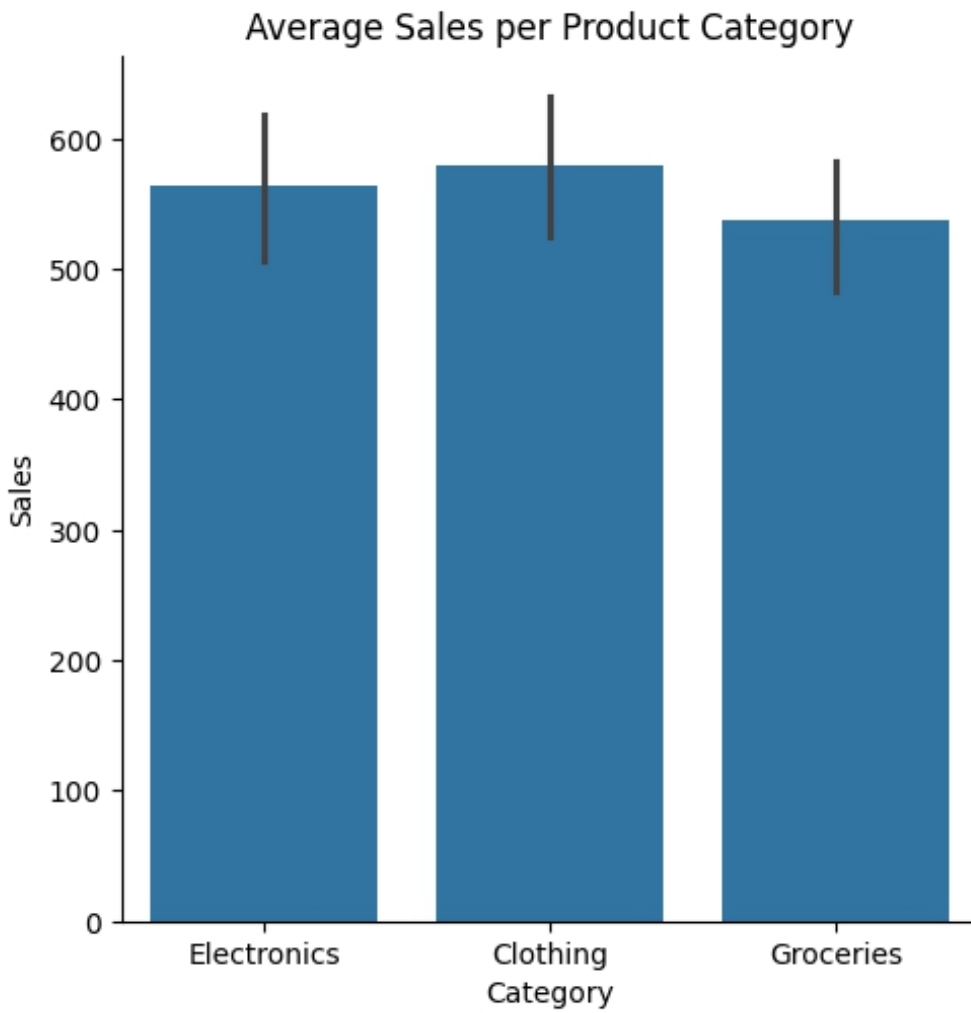
This will help the company understand if there are significant differences in customer satisfaction between the stores.

Generate the input data within your code.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(0)
data = {
    'Store': ['Store A']*100 + ['Store B']*100 + ['Store C']*100,
    'Satisfaction': np.random.choice([1, 2, 3, 4, 5], 300, p=[0.1,
0.2, 0.4, 0.2, 0.1])
}
df = pd.DataFrame(data)
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
np.random.seed(0)
data = {
    'Store': ['Store A']*100 + ['Store B']*100 + ['Store C']*100,
```

```
'Satisfaction': np.random.choice([1, 2, 3, 4, 5], 300, p=[0.1,
0.2, 0.4, 0.2, 0.1])
}
df = pd.DataFrame(data)
sns.catplot(x='Store', y='Satisfaction', kind='box', data=df)
plt.title('Customer Satisfaction Ratings by Store')
plt.xlabel('Store')
plt.ylabel('Satisfaction Rating')
plt.show()
```

To solve this problem, we first need to generate the input data.

We use NumPy and pandas to create a DataFrame that simulates survey results for customer satisfaction ratings across three stores.

We seed the random number generator to ensure reproducibility.

The satisfaction ratings are generated randomly based on specified probabilities.

The pandas DataFrame stores this data with two columns: 'Store' and 'Satisfaction'.

Next, we use the seaborn library to create a factor plot. Seaborn is a powerful visualization library built on top of matplotlib, providing high-level functions for statistical plots. Here, we use `sns.catplot()` with `kind='box'` to create a box plot, which visualizes the distribution of satisfaction ratings for each store.

Box plots display the median, quartiles, and potential outliers, giving a clear summary of the data distribution.

We set the title and labels for the x and y axes to make the plot informative.

Finally, `plt.show()` is called to display the plot.

This visualization helps to compare customer satisfaction across the stores, revealing any significant differences.

【Trivia】

- ▶ Box plots are useful for detecting outliers and understanding the spread of data.
- ▶ Seaborn provides many other categorical plots like bar plots, count plots, and violin plots, each useful for different types of data analysis.
- ▶ Using a seed with random number generators ensures that the results are reproducible, which is crucial for debugging and comparing results in data analysis.

32. Comparative Point Plot Generation

Importance★★★★☆

Difficulty★★★★☆

A marketing manager at a multinational company wants to compare the sales performance of three product lines across different regions.

They have collected data on sales figures for each product line in various countries.

Your task is to create a point plot that visually represents this comparative data.

The manager wants to see:

- ▶ Sales figures for each product line
- ▶ Data points for different countries
- ▶ Clear differentiation between product lines

Create a Python script that generates sample data and produces a point plot using seaborn.

The plot should clearly show the sales performance of each product line across different countries, allowing for easy comparison.

Make sure to include appropriate labels, a title, and a legend in your visualization.

Your code should:

Generate sample data for three product lines and multiple countries

Create a point plot using seaborn

Customize the plot for clarity and readability

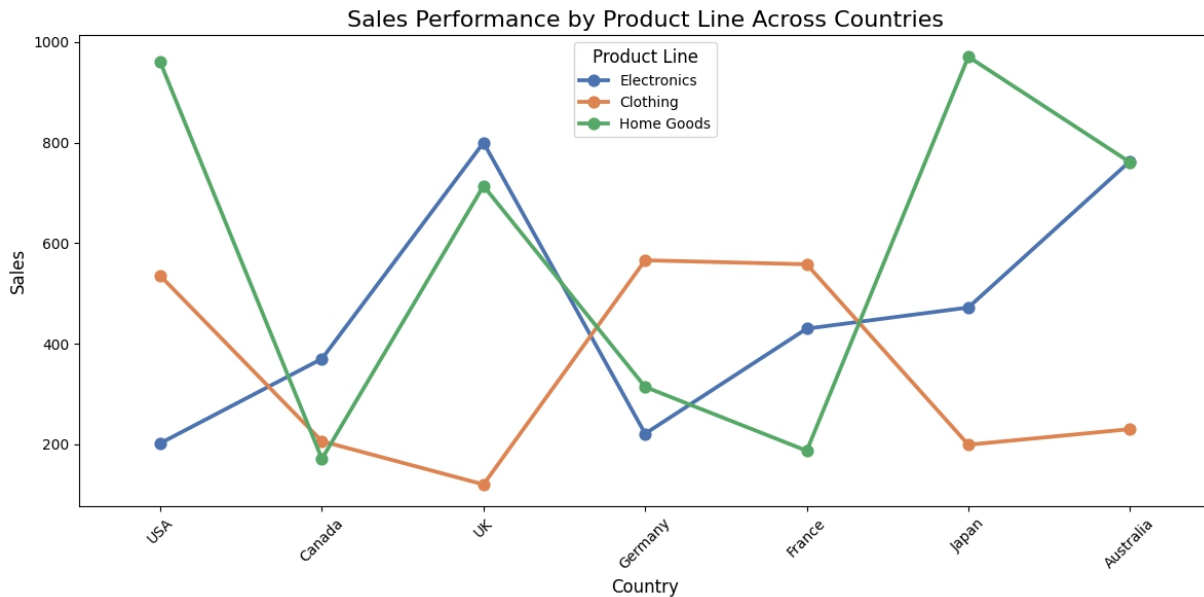
Display the final visualization

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
```

```
np.random.seed(42)
countries = ['USA', 'Canada', 'UK', 'Germany', 'France',
            'Japan', 'Australia']
product_lines = ['Electronics', 'Clothing', 'Home Goods']
data = [{'Country': country, 'Product Line': product, 'Sales':
np.random.randint(100, 1000)} for country in countries for
product in product_lines]
df = pd.DataFrame(data)
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
np.random.seed(42)
countries = ['USA', 'Canada', 'UK', 'Germany', 'France',
            'Japan', 'Australia']
product_lines = ['Electronics', 'Clothing', 'Home Goods']
data = [{ 'Country': country, 'Product Line': product, 'Sales':
np.random.randint(100, 1000)} for country in countries for
product in product_lines]
df = pd.DataFrame(data)
plt.figure(figsize=(12, 6))
sns.pointplot(x='Country', y='Sales', hue='Product Line',
data=df, palette='deep')
```

```
plt.title('Sales Performance by Product Line Across  
Countries', fontsize=16)  
plt.xlabel('Country', fontsize=12)  
plt.ylabel('Sales', fontsize=12)  
plt.xticks(rotation=45)  
plt.legend(title='Product Line', title_fontsize='12',  
fontsize='10')  
plt.tight_layout()  
plt.show()
```

This code creates a point plot to compare sales performance across different product lines and countries.

Let's break down the key components of the data processing and visualization:

Data Generation:

We use NumPy's random number generator to create sample sales data.

The data is structured as a list of dictionaries, each containing information about the country, product line, and sales figure.

This list is then converted into a pandas DataFrame for easy manipulation.

Data Visualization:

Setting up the plot:

We use matplotlib.pyplot to create a figure with a specified size (12x6 inches).

Creating the point plot:

The seaborn.pointplot function is used to create the visualization.

It takes the following main arguments:

x='Country': Countries are plotted on the x-axis
y='Sales': Sales figures are plotted on the y-axis
hue='Product Line': This differentiates the product lines by color
data=df: The DataFrame containing our data
palette='deep': A color palette for distinguishing product lines

Customizing the plot:

We set a title for the plot using `plt.title()`

X and Y axis labels are added with `plt.xlabel()` and `plt.ylabel()`

X-axis labels (country names) are rotated 45 degrees for better readability using `plt.xticks(rotation=45)`

A legend is added and customized using `plt.legend()`

Finalizing and displaying:

`plt.tight_layout()` is used to automatically adjust the plot layout

`plt.show()` displays the final visualization

This code demonstrates several important aspects of data visualization in Python:

Using pandas for data manipulation

Leveraging seaborn for statistical data visualization

Customizing plots with matplotlib for improved clarity and aesthetics

The resulting point plot effectively shows:

The sales performance of each product line across different countries

How sales vary between countries for each product line

Which product lines perform better or worse in specific countries

This type of visualization is particularly useful for comparing categorical data across multiple categories and subgroups,

making it ideal for the marketing manager's needs in this scenario.

【Trivia】

- ▶ Seaborn is built on top of matplotlib and provides a high-level interface for drawing attractive statistical graphics.
- ▶ Point plots are excellent for showing the central tendency (like mean or median) of numeric variables for different levels of categorical variables.
- ▶ In a point plot, the points represent the mean of the y variable for each x category, while the error bars often represent the 95% confidence interval.
- ▶ The 'deep' color palette in seaborn is designed to be colorblind-friendly, making your visualizations more accessible.
- ▶ Rotating x-axis labels is a common technique to prevent overlapping when dealing with long category names or many categories.
- ▶ The `tight_layout()` function in matplotlib automatically adjusts subplot parameters to give specified padding. This is particularly useful when you have titles, labels, or other elements that might otherwise overlap.
- ▶ Seaborn integrates closely with pandas data structures, making it easy to plot data directly from DataFrames.
- ▶ When working with real-world data, it's often necessary to handle missing values or outliers before creating visualizations. Pandas provides various methods for this, such as `dropna()` for removing rows with missing data or `fillna()` for filling in missing values.

33. Creating a Bar Plot with Categorical Data

Importance★★★★☆

Difficulty★★☆☆☆

A retail store wants to visualize the distribution of sales across different product categories to identify the most popular ones.

Your task is to create a bar plot that shows the total sales for each product category.

Use the provided sample data to generate this plot.

The data consists of two columns: "Category" and "Sales".

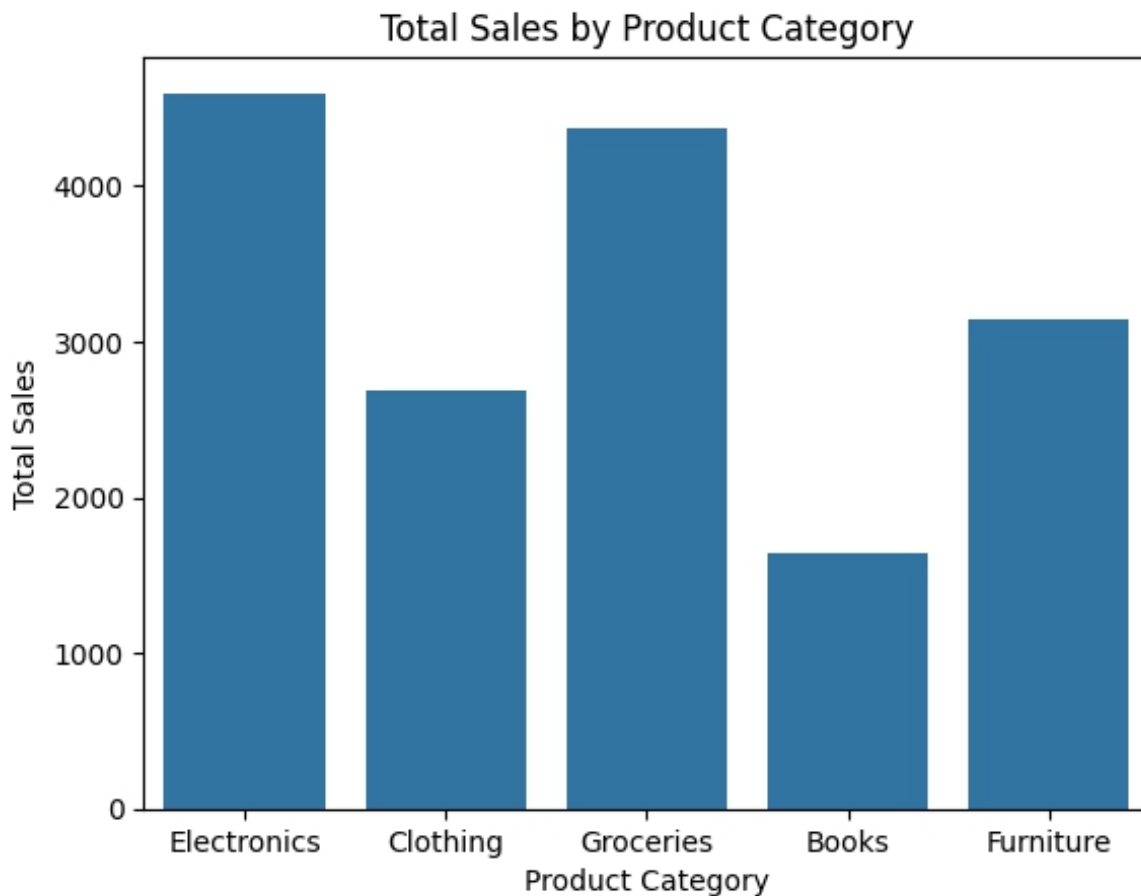
The "Category" column contains the product categories, and the "Sales" column contains the sales amounts for each transaction.

Generate the necessary sample data and create the bar plot.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
# Generating sample datacategories = ['Electronics',
'Clothing', 'Groceries', 'Home & Garden', 'Sports', 'Toys']
np.random.seed(0) # For reproducibility
data = pd.DataFrame({
'Category': np.random.choice(categories, 100),
'Sales': np.random.randint(10, 1000, 100)
})
data.head()
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Generating sample datacategories = ['Electronics',
'Clothing', 'Groceries', 'Home & Garden', 'Sports', 'Toys']
np.random.seed(0) # For reproducibility
data = pd.DataFrame({
'Category': np.random.choice(categories, 100),
'Sales': np.random.randint(10, 1000, 100)
```



```
}  
# Aggregating sales data by category  
sales_by_category =  
data.groupby('Category').sum().reset_index()  
# Creating the bar plot  
plt.figure(figsize=(10, 6))  
sns.barplot(x='Category', y='Sales',  
data=sales_by_category)  
plt.title('Total Sales by Category')  
plt.xlabel('Category')  
plt.ylabel('Total Sales')  
plt.show()
```

To begin, we generate the sample data using numpy and pandas.

We create a DataFrame with two columns: "Category" and "Sales". The "Category" column is populated with random choices from a predefined list of product categories, and the "Sales" column contains random integers representing sales amounts.

By using `np.random.seed(0)`, we ensure that the random data is reproducible.

After generating the sample data, we aggregate the sales amounts by category using the `groupby` method in pandas. This groups the data by the "Category" column and sums the "Sales" values for each category.

The result is a new DataFrame where each row represents a product category and its corresponding total sales.

We then create a bar plot to visualize the total sales for each product category.

We use seaborn's `barplot` function, specifying the x-axis as "Category" and the y-axis as "Sales".

We also customize the plot by setting the figure size, adding a title, and labeling the axes.

When the code is executed, it displays a bar plot showing the total sales for each product category, allowing the retail store to identify which categories are the most popular.

【Trivia】

Seaborn is built on top of matplotlib and provides a high-level interface for drawing attractive statistical graphics. It simplifies the process of creating complex visualizations with fewer lines of code.

Additionally, seaborn integrates well with pandas DataFrames, making it easy to visualize data directly from DataFrame objects.

34. Count Plot of Categorical Data

Importance★★★★☆

Difficulty★★★★☆

A retail company wants to analyze the distribution of their product categories to better understand their inventory. They have provided you with a sample dataset of product categories and the number of products in each category. Your task is to create a count plot to visualize the frequency of each product category.

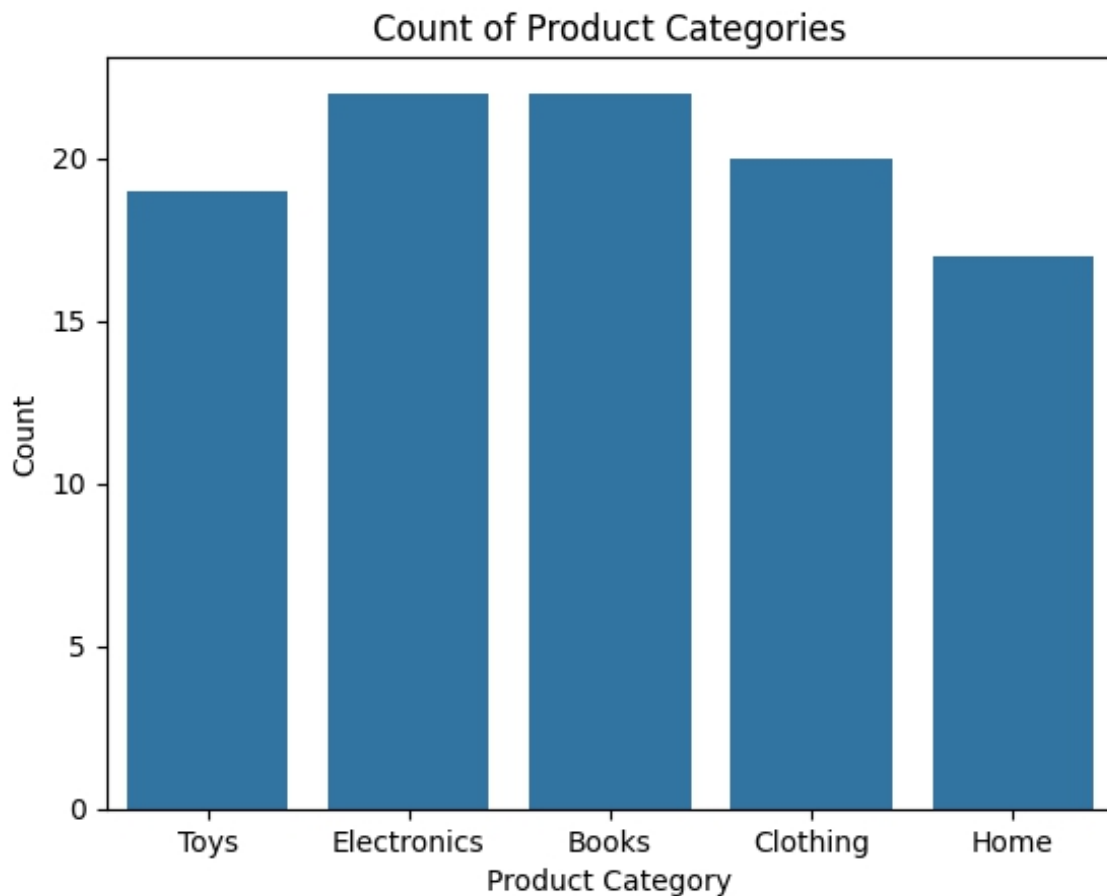
Write a Python code to generate a count plot of the product categories using the provided sample data.

Use the following data for your analysis: Category A: 30 products
Category B: 45 products
Category C: 15 products
Category D: 25 products
Category E: 10 products

【Data Generation Code Example】

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
data = {'Category': ['A', 'B', 'C', 'D', 'E'], 'Count': [30, 45, 15, 25, 10]}
df = pd.DataFrame(data)
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
data = {'Category': ['A', 'B', 'C', 'D', 'E'], 'Count': [30, 45, 15, 25, 10]}
df = pd.DataFrame(data)
sns.countplot(data=df, x='Category', y='Count')
plt.xlabel('Product Category')
plt.ylabel('Number of Products')
plt.title('Count of Products by Category')
```

```
plt.show()
```

To solve this problem, we first need to create a dataset containing the product categories and their respective counts.

This is done using a dictionary where the keys are 'Category' and 'Count', and the values are lists of category labels and their corresponding product counts.

We then convert this dictionary into a pandas DataFrame for easier manipulation and plotting.

For visualization, we use the seaborn library, which provides a high-level interface for drawing attractive and informative statistical graphics.

The `sns.countplot` function is used to create a count plot.

In this case, we set the data parameter to our DataFrame `df`, `x` to 'Category', and `y` to 'Count' to specify the categorical and numerical data for the plot.

After specifying the data for the plot, we use `plt.xlabel`, `plt.ylabel`, and `plt.title` to set the labels for the x-axis, y-axis, and the title of the plot, respectively.

Finally, we call `plt.show()` to display the plot.

This code will generate a count plot showing the number of products in each category, helping the retail company understand their inventory distribution.

【Trivia】

Count plots are particularly useful for visualizing the distribution of categorical data.

They are often used in exploratory data analysis to understand the frequency of different categories in a dataset.

Seaborn's `countplot` function is a powerful tool because it can easily handle pandas DataFrames, allowing for quick and effective data visualization.

Using count plots can help businesses identify trends, make informed decisions about inventory management, and spot categories that may need attention.

35. KDE Plot of Distribution Data

Importance★★★★☆

Difficulty★★★☆☆

You are working as a data analyst for a company that wants to visualize the distribution of their sales data using a Kernel Density Estimate (KDE) plot.

Your task is to create a KDE plot based on a sample sales data.

This will help the company understand the distribution and density of sales over a certain period.

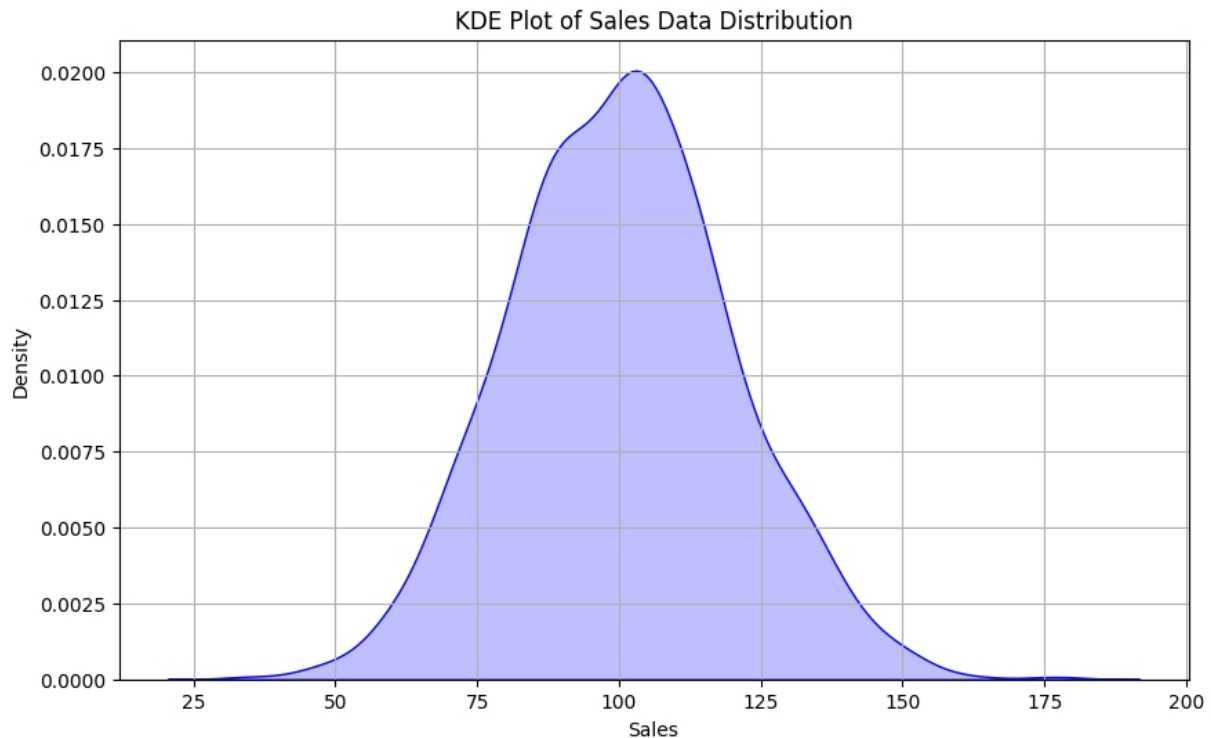
Generate the sample sales data within your code and produce the KDE plot.

Use appropriate libraries to accomplish this task.

【Data Generation Code Example】


```
import numpy as np
np.random.seed(42)
sales_data = np.random.normal(loc=100, scale=20,
size=1000)
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
np.random.seed(42)
sales_data = np.random.normal(loc=100, scale=20,
size=1000)
plt.figure(figsize=(10,6))
sns.kdeplot(sales_data, shade=True, color="blue")
plt.title("KDE Plot of Sales Data Distribution")
plt.xlabel("Sales")
plt.ylabel("Density")
plt.grid(True)
plt.show()
```

To create a Kernel Density Estimate (KDE) plot, we first need to generate sample data.

We use NumPy to create random sales data with a normal distribution, setting a seed for reproducibility.

The KDE plot will be created using Seaborn and Matplotlib libraries.

We start by importing the necessary libraries: NumPy for data generation, Matplotlib for plotting, and Seaborn for the KDE plot.

We set the seed for NumPy's random number generator to ensure the results are reproducible.

Then, we generate the sample sales data using a normal distribution with a mean (loc) of 100 and a standard deviation (scale) of 20, creating 1000 data points.

Next, we initialize a figure for our plot and use Seaborn's `kdeplot` function to create the KDE plot of the sales data. The `shade` parameter adds shading under the curve, and `color` specifies the color of the plot.

We then add a title and labels for the x and y axes using Matplotlib's `title`, `xlabel`, and `ylabel` functions.

Finally, we display the plot using `plt.show()`. The `grid` function is used to add a grid to the plot for better readability.

【Trivia】

- ▶ The Kernel Density Estimate (KDE) is a non-parametric way to estimate the probability density function of a random variable.
- ▶ KDE plots are particularly useful for visualizing the distribution of data without assuming an underlying distribution model.
- ▶ The bandwidth parameter in KDE controls the smoothness of the estimated density curve: a smaller bandwidth leads

to a more wiggly curve, while a larger bandwidth leads to a smoother curve.

36. Violin Plot Creation with Seaborn

Importance★★★★☆

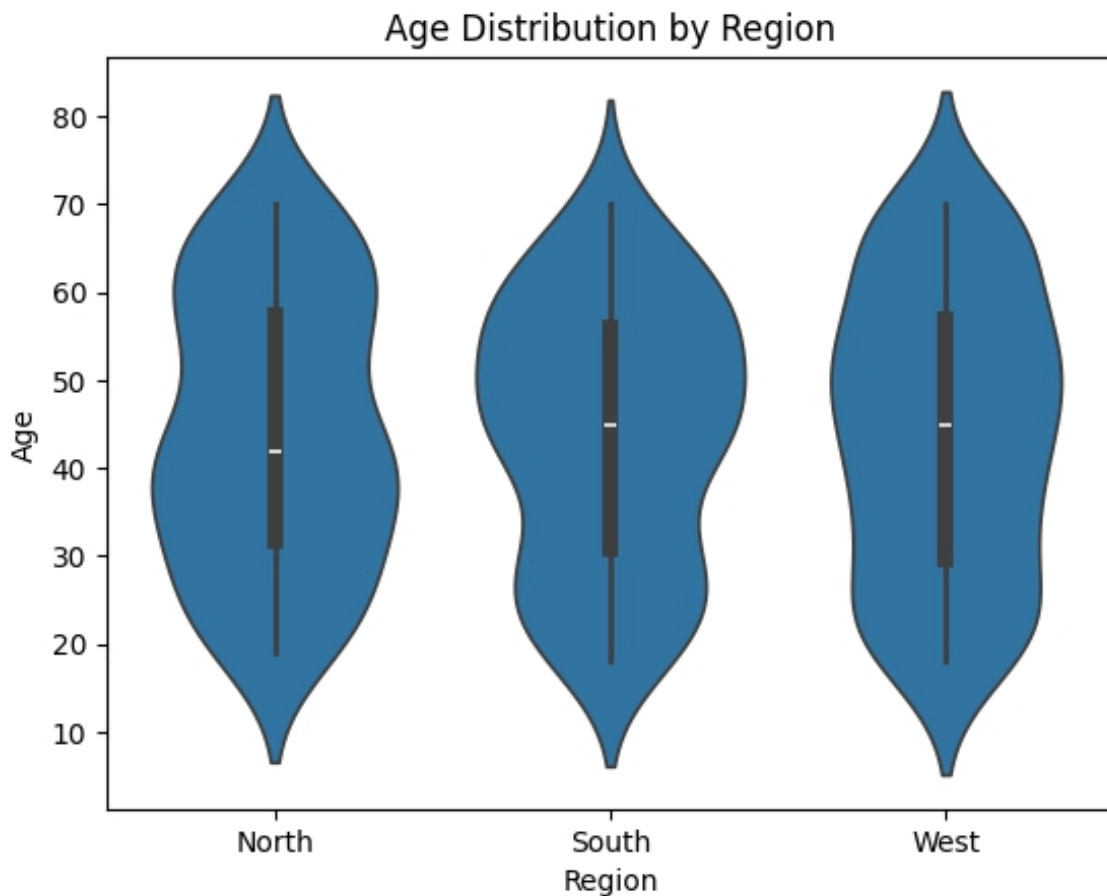
Difficulty★★★★☆

A company wants to analyze the distribution of customer ages across different regions to tailor their marketing strategies. They have gathered age data from three different regions: North, South, and West. Using this data, they would like to create a violin plot to visualize the age distribution in each region. You need to write a Python script that performs the following tasks: Generate random age data for three regions. Create a violin plot to visualize the age distribution for these regions. Ensure the generated data has the following properties: Ages range from 18 to 70. Each region should have 100 data points. Write the code to generate the data and create the violin plot using Seaborn.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
np.random.seed(42)
regions = ['North', 'South', 'West']
ages = [np.random.randint(18, 71, 100) for _ in regions]
data = pd.DataFrame({'Region': np.repeat(regions, 100),
                    'Age': np.concatenate(ages)})
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
np.random.seed(42)
regions = ['North', 'South', 'West']
ages = [np.random.randint(18, 71, 100) for _ in regions]
data = pd.DataFrame({'Region': np.repeat(regions, 100),
                    'Age': np.concatenate(ages)})
sns.violinplot(x='Region', y='Age', data=data)
```

```
plt.title('Age Distribution by Region')
plt.xlabel('Region')
plt.ylabel('Age')
plt.show()
```

The code begins by importing necessary libraries: numpy, pandas, seaborn, and matplotlib.

A random seed is set using `np.random.seed(42)` to ensure reproducibility of the random numbers.

Three regions are defined in a list: North, South, and West.

For each region, 100 random integers representing ages between 18 and 70 are generated using

`np.random.randint(18, 71, 100)`. The data for all regions are then combined into a pandas DataFrame with columns 'Region' and 'Age'. The 'Region' column repeats each region name 100 times, and the 'Age' column contains the concatenated age data.

The violin plot is created using `sns.violinplot()`, with 'Region' as the x-axis and 'Age' as the y-axis. Seaborn's violin plot function provides a method to visualize the distribution of the data, including its density. The plot is titled 'Age Distribution by Region', and the axes are labeled appropriately. Finally, `plt.show()` displays the plot.

Violin plots are useful for understanding the distribution and density of the data. They combine aspects of box plots and kernel density plots, providing a comprehensive view of the data distribution. In this example, the violin plot helps visualize age distributions across different regions, aiding in decision-making for targeted marketing strategies.

【Trivia】

Violin plots were introduced by John W. Tukey, an American mathematician and statistician, known for his development of exploratory data analysis techniques. Violin plots are

especially useful when comparing multiple categories or groups, as they provide detailed insights into the data's distribution, including the presence of multiple modes.

37. Boxen Plot Visualization of Distribution Data

Importance★★★★☆

Difficulty★★★★☆

A company wants to analyze the distribution of its sales data for the past year.

You are tasked with creating a visualization that helps in understanding the spread and outliers in the data.

Specifically, you need to create a boxen plot for the sales data to provide a detailed view of the distribution.

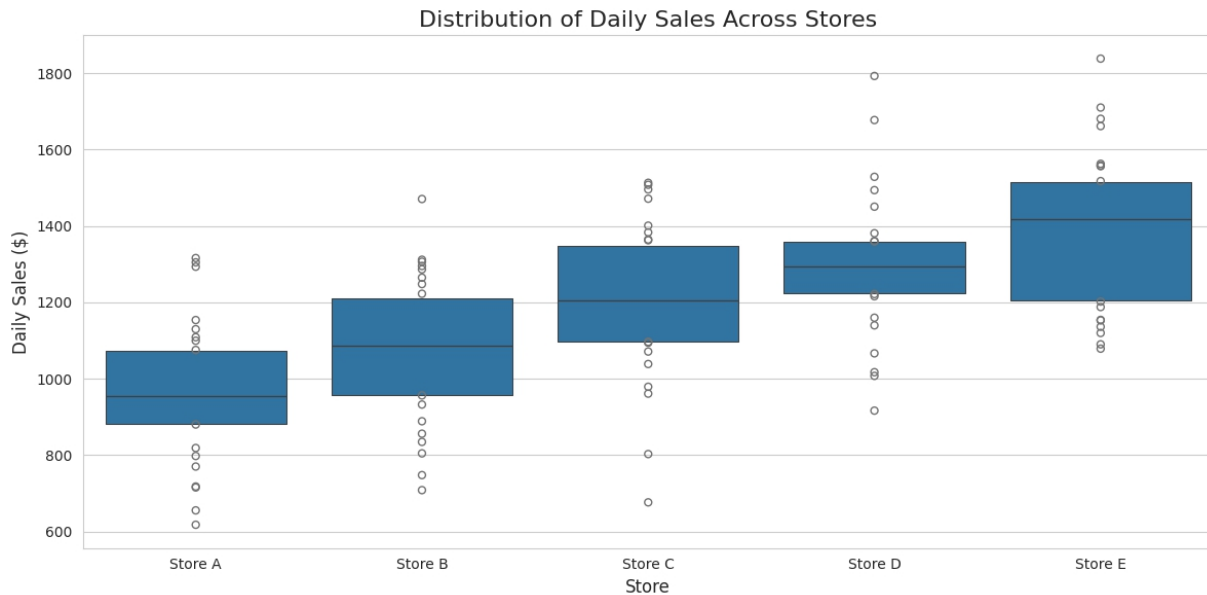
Generate a dataset of 500 random sales records ranging between \$100 and \$10000.

Create a boxen plot using Python to visualize this data.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
np.random.seed(42) # For reproducibility
# Generate random sales data
sales_data = np.random.uniform(100, 10000, 500)
# Convert to DataFrame
sales_df = pd.DataFrame(sales_data, columns=['Sales'])
# Display first few rows
sales_df.head()
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
np.random.seed(42) # For reproducibility
# Generate random sales data
sales_data = np.random.uniform(100, 10000, 500)
# Convert to DataFrame
sales_df = pd.DataFrame(sales_data, columns=['Sales'])
# Create boxen plot
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))
sns.boxenplot(x=sales_df['Sales'])
plt.title('Boxen Plot of Sales Data')
plt.xlabel('Sales Amount ($)')
```



```
plt.show()
```

A boxen plot is a variation of a box plot that provides more detailed information on the distribution, especially on the tails.

This plot is particularly useful when dealing with large datasets or when you need to understand the distribution's spread and outliers more comprehensively.

In this exercise, we first generate a synthetic dataset representing sales records using `np.random.uniform`, which creates 500 random values uniformly distributed between \$100 and \$10,000.

We then convert this data into a pandas DataFrame for easier handling.

Next, we use the seaborn library, which is built on top of matplotlib and provides a high-level interface for drawing attractive statistical graphics.

By calling `sns.boxenplot` and passing our sales data, we create the boxen plot.

The `plt.title` and `plt.xlabel` functions are used to add a title and label to the x-axis, respectively.

Finally, `plt.show` is called to display the plot. This function ensures that the plot is rendered in a graphical window.

【Trivia】

- ▶ Boxen plots, introduced in seaborn version 0.9.0, are specifically designed to provide more insight into the tails of the distribution compared to traditional box plots.
- ▶ The term "boxen plot" comes from the combination of "box plot" and "violin plot", reflecting its hybrid nature.
- ▶ While traditional box plots are limited to showing median, quartiles, and potential outliers, boxen plots use a series of nested boxes to show additional quantiles, giving a more detailed view of the data distribution.

- ▶ Understanding data distributions is crucial in fields like finance, where identifying outliers can be key to risk management and decision-making.
- ▶ The seaborn library is highly recommended for statistical data visualization due to its ease of use and integration with pandas data structures.

38. Joint Plot of Bivariate Data

Importance★★★★☆

Difficulty★★★☆☆

A marketing company wants to understand the relationship between the number of online advertisements shown and the number of products sold.

Create a joint plot to visualize the correlation between these two variables.

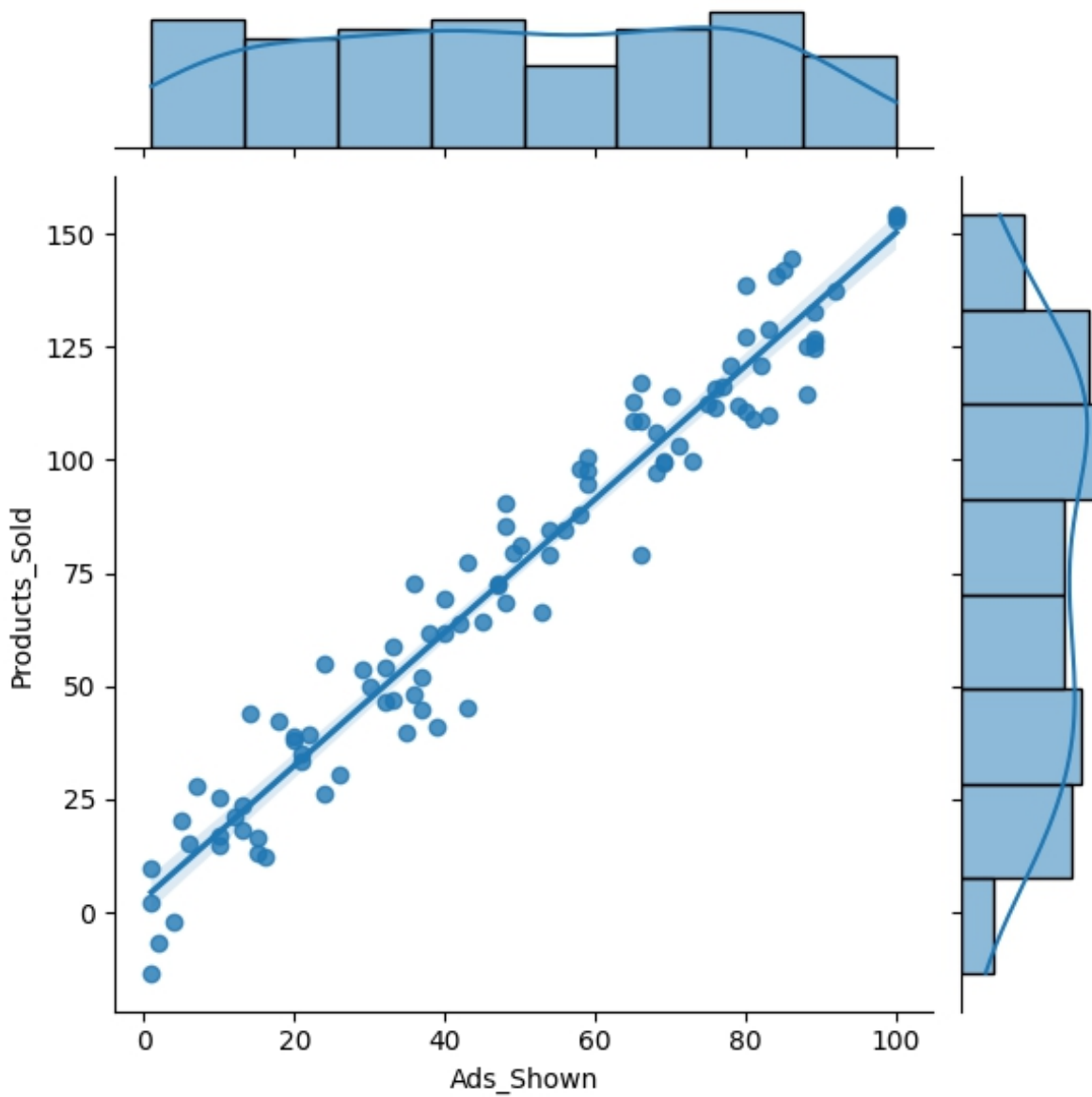
The data includes the number of ads shown (ranging from 1 to 100) and the number of products sold (with some random variation).

Generate the sample data within your code.

【Data Generation Code Example】

```
import numpy as np
np.random.seed(0)
ads_shown = np.random.randint(1, 101, 100)
products_sold = ads_shown * 1.5 + np.random.normal(0,
10, 100)
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
np.random.seed(0)
```

```
ads_shown = np.random.randint(1, 101, 100)
products_sold = ads_shown * 1.5 + np.random.normal(0,
10, 100)
data = pd.DataFrame({'Ads_Shown': ads_shown,
'Products_Sold': products_sold})
sns.jointplot(x='Ads_Shown', y='Products_Sold', data=data,
kind='reg')
plt.show()
```

This exercise focuses on creating a joint plot to visualize the relationship between two variables, in this case, the number of advertisements shown and the number of products sold. Joint plots are useful for examining how two variables are related by combining scatter plots and histograms.

First, the necessary libraries are imported: numpy for generating random data, pandas for data manipulation, matplotlib for plotting, and seaborn for creating the joint plot.

The numpy random seed is set for reproducibility.

Next, the sample data is generated using numpy. The number of ads shown is generated as random integers between 1 and 100. The number of products sold is calculated based on the ads shown, with some added random variation to simulate real-world data.

The data is then stored in a pandas DataFrame, which makes it easy to handle and pass to the plotting function.

Seaborn's jointplot function is used to create the plot, specifying the x and y variables and the type of plot ('reg' for regression). The plot is then displayed using plt.show().

This exercise demonstrates how to visualize the correlation between two variables using Python's data manipulation and visualization libraries. It also highlights the importance

of data preparation and the integration of different libraries to achieve the desired analysis.

【Trivia】

- ▶ Joint plots are part of the Seaborn library, which is built on top of Matplotlib and provides a high-level interface for drawing attractive statistical graphics.
- ▶ The 'kind' parameter in the `sns.jointplot` function can take various values like 'scatter', 'reg', 'resid', 'kde', or 'hex', each providing a different type of joint plot.
- ▶ Adding a regression line to a scatter plot can help in understanding the trend and strength of the relationship between the variables.

39. Lmplot Regression Analysis

Importance★★★★☆

Difficulty★★★☆☆

You are a data analyst at a retail company. Your task is to visualize the relationship between advertising spend and sales for various products.

Generate a regression plot using seaborn's Lmplot to display this relationship.

Use the following sample data:

advertising_spend: A list of advertising expenses in thousands of dollars.

sales: A list of corresponding sales figures in thousands of units.

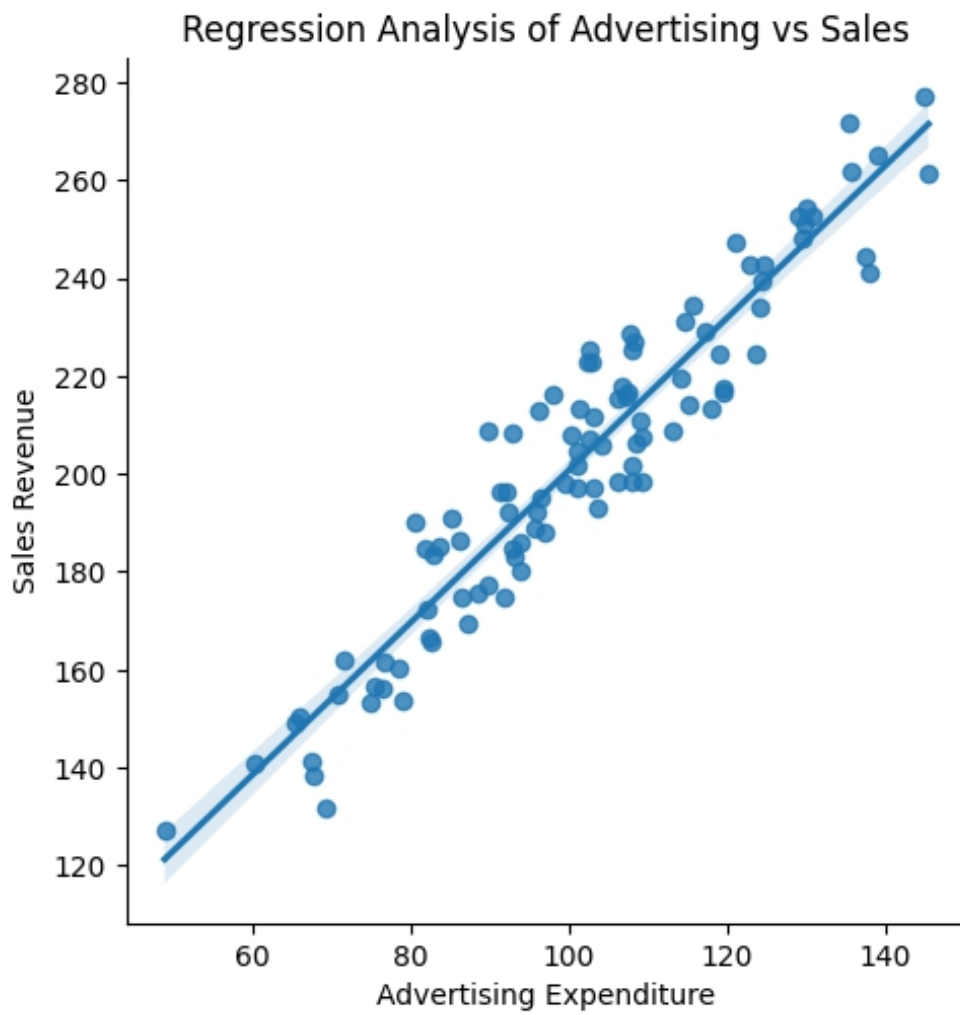
Create the data within the code, and then generate a regression plot showing the relationship.

Ensure the plot includes a regression line and a scatter plot of the data points.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
data = {
    "advertising_spend": np.random.randint(50, 150, 100),
    "sales": np.random.randint(200, 1000, 100)
}
df = pd.DataFrame(data)
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
data = {
    "advertising_spend": np.random.randint(50, 150, 100),
    "sales": np.random.randint(200, 1000, 100)
}
df = pd.DataFrame(data)
```



```
sns.Implot(x="advertising_spend", y="sales", data=df)
plt.title("Regression Analysis of Advertising Spend vs Sales")
plt.xlabel("Advertising Spend (Thousands of $)")
plt.ylabel("Sales (Thousands of Units)")
plt.show()
```

To create the regression analysis plot, first, we import the necessary libraries: pandas for data manipulation, numpy for generating random data, seaborn for visualization, and matplotlib for plotting.

We then generate sample data with advertising spend and sales figures using numpy's random integer generation. This data is stored in a dictionary and then converted into a DataFrame using pandas.

The core of the task is the visualization part, where we use seaborn's Implot to create a scatter plot with a regression line. Implot takes the DataFrame and the column names for the x and y axes as parameters.

Finally, we add titles and labels to the plot using matplotlib's plt functions to make the plot informative and clear. The plt.show() function displays the plot.

【Trivia】

- ▶ The Implot function in seaborn combines both scatter plots and regression lines, making it a powerful tool for quickly visualizing linear relationships.
- ▶ Seaborn is built on top of matplotlib and provides a high-level interface for drawing attractive and informative statistical graphics.
- ▶ Regression analysis is a fundamental statistical method used in predictive analytics, making it essential for understanding relationships between variables and forecasting.

40. Creating a Pair Plot for Customer Data Analysis

Importance★★★★☆

Difficulty★★★☆☆

You are a data analyst working for an e-commerce company. The marketing team wants to understand the relationships between different customer attributes and their spending behavior.

They have provided you with a dataset containing information about customers' age, income, time spent on the website, and total amount spent.

Your task is to create a pair plot using Seaborn to visualize the relationships between these variables.

The pair plot should include scatter plots for all pairs of variables and histograms for each variable on the diagonal. Additionally, use different colors to represent customer categories (e.g., 'Regular', 'Premium', 'VIP') in the plot. Generate sample data within your code to represent this scenario.

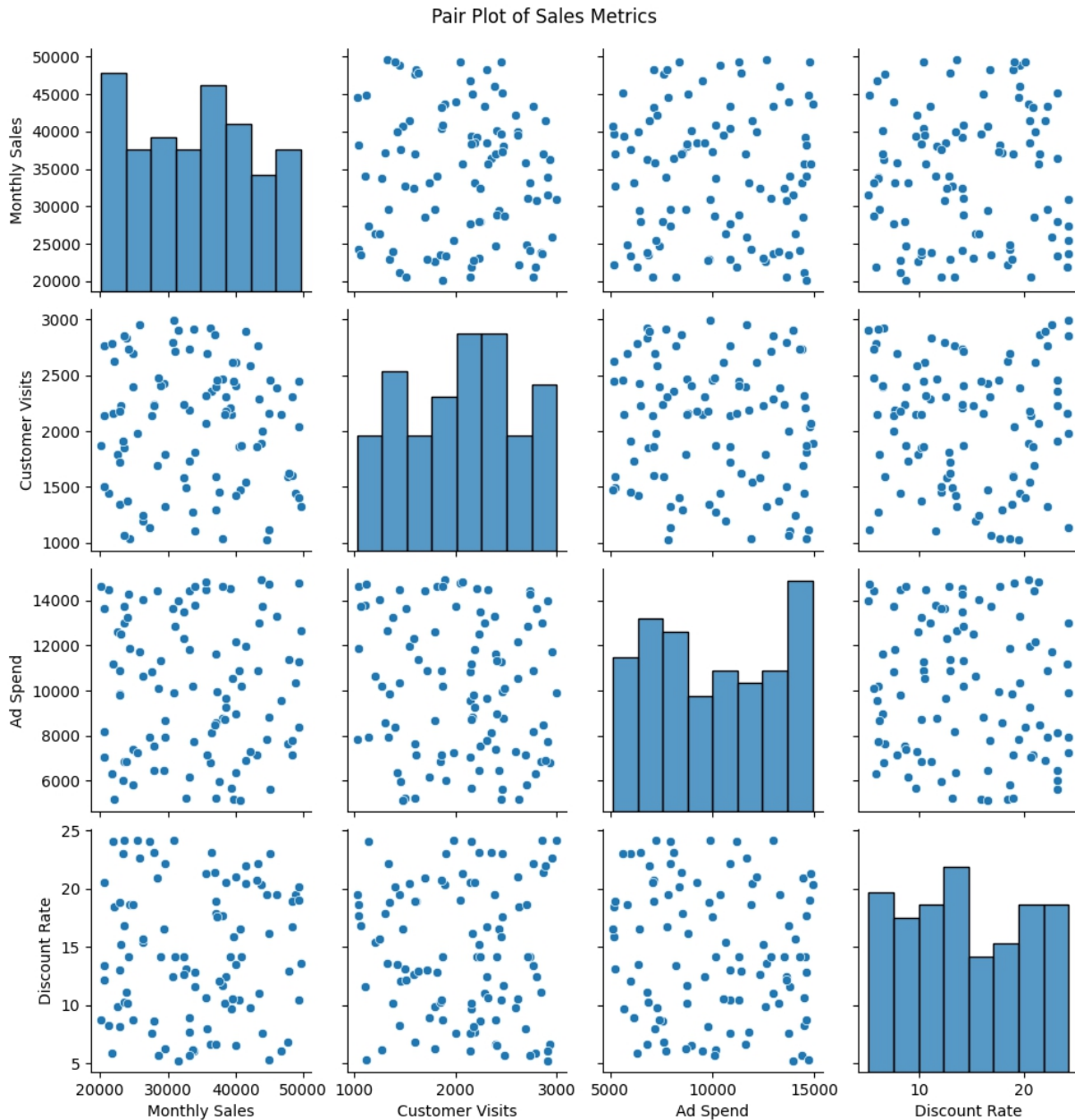
Ensure that your visualization is clear and informative for the marketing team to draw insights from.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
np.random.seed(42)
n_samples = 200
age = np.random.randint(18, 70, n_samples)
income = np.random.normal(50000, 15000, n_samples)
```

```
time_spent = np.random.exponential(60, n_samples)
amount_spent = np.random.normal(500, 200, n_samples) +
income * 0.01
categories = np.random.choice(['Regular', 'Premium', 'VIP'],
n_samples, p=[0.6, 0.3, 0.1])
data = pd.DataFrame({'Age': age,
'Income': income,
'Time_Spent': time_spent,
'Amount_Spent': amount_spent,
'Category': categories})
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import pandas as pd
import seaborn as sns
```

```

import matplotlib.pyplot as plt
np.random.seed(42)
n_samples = 200
age = np.random.randint(18, 70, n_samples)
income = np.random.normal(50000, 15000, n_samples)
time_spent = np.random.exponential(60, n_samples)
amount_spent = np.random.normal(500, 200, n_samples) +
income * 0.01
categories = np.random.choice(['Regular', 'Premium', 'VIP'],
n_samples, p=[0.6, 0.3, 0.1])
data = pd.DataFrame({'Age': age,
'Income': income,
'Time_Spent': time_spent,
'Amount_Spent': amount_spent,
'Category': categories})
# Set the style for the plot
sns.set(style="ticks", color_codes=True)
# Create the pair plot
g = sns.pairplot(data, hue="Category", vars=["Age",
"Income", "Time_Spent", "Amount_Spent"],
diag_kind="hist", plot_kws={"alpha": 0.6}, height=2.5)
# Customize the plot
g.fig.suptitle("Customer Attributes and Spending Behavior",
y=1.02)
# Adjust the layout and display the plot
plt.tight_layout()
plt.show()

```

This code creates a pair plot to visualize relationships between customer attributes and spending behavior.

Let's break down the key components and explain the data visualization process:

Data Generation:

We use NumPy to generate random data for 200 customers. The data includes age (18-70), income (normal distribution), time spent on the website (exponential distribution), and amount spent (related to income).

We also assign customer categories (Regular, Premium, VIP) with different probabilities.

Data Preparation:

The generated data is organized into a pandas DataFrame, which is a 2D labeled data structure.

Each column represents a different attribute: Age, Income, Time_Spent, Amount_Spent, and Category.

Seaborn Pair Plot:

We use seaborn's pairplot function to create the visualization.

pairplot creates a grid of axes with scatter plots for each pair of variables and histograms on the diagonal.

The 'hue' parameter is set to "Category", which colors the data points based on the customer category.

We specify the variables to include in the plot using the 'vars' parameter.

'diag_kind="hist"' sets the diagonal plots to be histograms.

'plot_kws={"alpha": 0.6}' sets the transparency of the scatter plot points.

'height=2.5' sets the size of each subplot.

Customization:

We set a title for the entire figure using g.fig.suptitle().

plt.tight_layout() adjusts the spacing between subplots for a cleaner look.

Display:

Finally, plt.show() displays the plot.

This pair plot allows the marketing team to quickly visualize:

- ▶ The distribution of each variable (on the diagonal)
- ▶ The relationships between pairs of variables (scatter plots)
- ▶ How these relationships might differ across customer categories (color coding)

The resulting visualization provides a comprehensive overview of the customer data, enabling the marketing team to identify patterns, correlations, and potential segmentation strategies based on customer attributes and spending behavior.

【Trivia】

- ▶ Seaborn is built on top of Matplotlib and provides a high-level interface for drawing attractive statistical graphics.
- ▶ The pair plot is particularly useful for exploring correlations between multiple variables simultaneously.
- ▶ The exponential distribution used for 'time_spent' is often appropriate for modeling the time between events, like customer visits to a website.
- ▶ The seed in `np.random.seed(42)` ensures reproducibility of the random data generation. The number 42 is a reference to "The Hitchhiker's Guide to the Galaxy" by Douglas Adams.
- ▶ In data visualization, it's important to consider color blindness. Seaborn's default color palette is generally accessible, but you can use `sns.color_palette()` to create custom, colorblind-friendly palettes.
- ▶ The `pairplot` function in Seaborn automatically handles both continuous and categorical variables, adjusting the plot type accordingly.
- ▶ Pair plots can become computationally intensive and visually cluttered with a large number of variables. It's generally recommended to limit them to 5-6 key variables.

41. Heatmap of Correlation Matrix

Importance★★★★☆

Difficulty★★★★☆☆

You are working as a data analyst for a financial firm. Your task is to analyze the correlation between different financial indicators for a given set of data.

You need to create a heatmap to visualize these correlations.

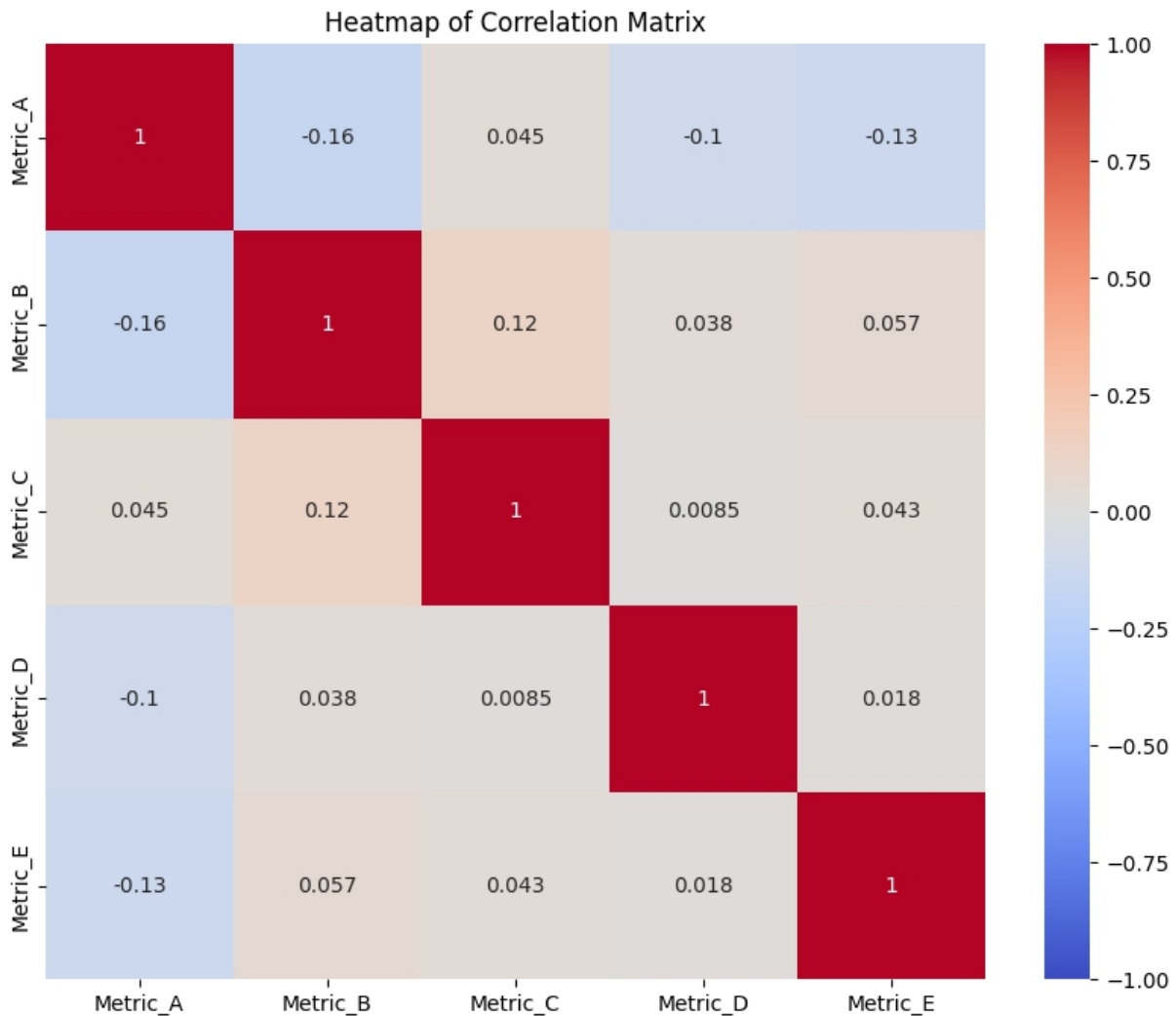
Use the provided code to generate sample data and then write the necessary Python code to create and display a heatmap of the correlation matrix.

This will help the firm understand the relationships between different financial metrics.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(42)
data = pd.DataFrame(np.random.randn(100, 5), columns=
['Metric_A', 'Metric_B', 'Metric_C', 'Metric_D', 'Metric_E'])
```


【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
np.random.seed(42)
data = pd.DataFrame(np.random.randn(100, 5), columns=
['Metric_A', 'Metric_B', 'Metric_C', 'Metric_D', 'Metric_E'])
```

```
corr = data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', vmin=-1,
vmax=1)
plt.title('Heatmap of Correlation Matrix')
plt.show()
```

To solve this problem, first, import the necessary libraries: pandas for data manipulation, numpy for numerical operations, seaborn for advanced visualizations, and matplotlib.pyplot for basic plotting functions.

We begin by setting a random seed using `np.random.seed(42)` to ensure reproducibility.

Next, generate a DataFrame `data` with 100 rows and 5 columns of random numbers, representing different financial metrics: 'Metric_A', 'Metric_B', 'Metric_C', 'Metric_D', and 'Metric_E'. This is achieved using

```
pd.DataFrame(np.random.randn(100, 5), columns=
['Metric_A', 'Metric_B', 'Metric_C', 'Metric_D', 'Metric_E']).
```

Calculate the correlation matrix of the DataFrame using `data.corr()`, which computes pairwise correlation of columns, excluding NA/null values.

To visualize the correlation matrix, use `sns.heatmap()` from the seaborn library. Create a figure with specific dimensions using `plt.figure(figsize=(10, 8))`. Then, generate the heatmap by passing the correlation matrix `corr` to `sns.heatmap()` with additional parameters like `annot=True` to display the correlation values on the heatmap, `cmap='coolwarm'` for the color map, and `vmin=-1, vmax=1` to set the range of the color scale.

Finally, set the title of the heatmap with `plt.title('Heatmap of Correlation Matrix')` and display the plot using `plt.show()`.

【Trivia】

- ▶ A correlation matrix is a table showing correlation coefficients between many variables. Each cell in the table shows the correlation between two variables. The value is between -1 and 1.
- ▶ Heatmaps are a great way to visualize the strength of correlations between variables in a dataset. High positive or negative correlations are easily spotted with contrasting colors.
- ▶ Seaborn's `heatmap()` function provides several options to enhance the visualization, such as annotating cells with correlation values and using different color maps to represent the data effectively.

42. Scatter Matrix Plot of Multivariate Data

Importance★★★★☆

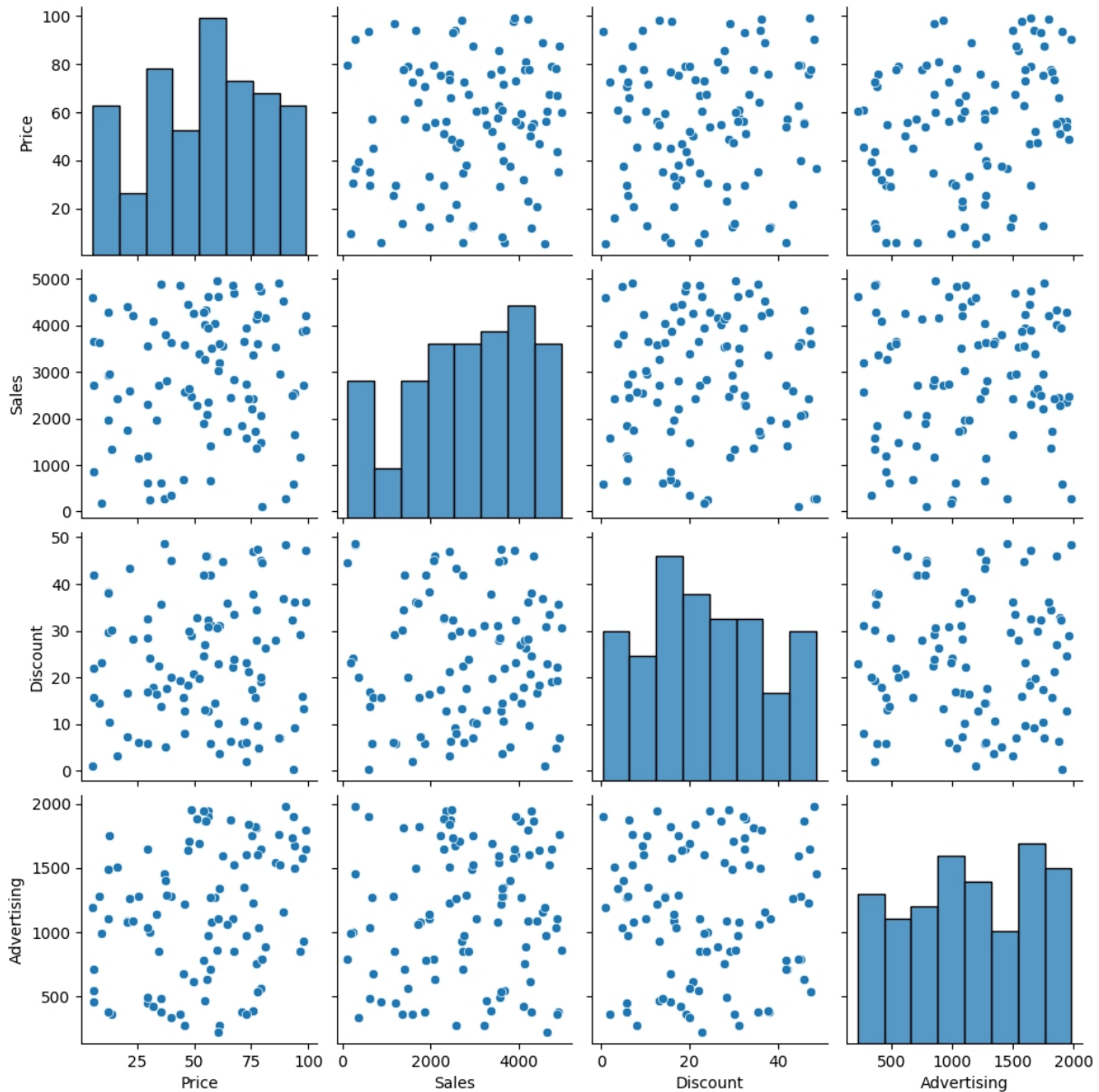
Difficulty★★★★☆

You are a data analyst working for a retail company. Your team has collected data on various product attributes and sales performance. The goal is to identify relationships between different attributes to optimize product offerings. Using the data provided, create a scatter matrix plot to visualize the relationships between the following attributes: 'Price', 'Sales', 'Discount', and 'Advertising'. Generate a synthetic dataset for these attributes and create the scatter matrix plot using Python. Ensure your visualization clearly shows the relationships between all pairs of attributes.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
# Generate random data for the attributes
data = pd.DataFrame({
    'Price': np.random.uniform(5, 100, 100),
    'Sales': np.random.uniform(50, 5000, 100),
    'Discount': np.random.uniform(0, 50, 100),
    'Advertising': np.random.uniform(200, 2000, 100)
})
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Generate random data for the attributes
data = pd.DataFrame({
    'Price': np.random.uniform(5, 100, 100),
    'Sales': np.random.uniform(50, 5000, 100),
    'Discount': np.random.uniform(0, 50, 100),
    'Advertising': np.random.uniform(200, 2000, 100)
})
# Create a scatter matrix plot
sns.pairplot(data)
plt.show()
```

To visualize the relationships between multiple variables, a scatter matrix plot, also known as a pair plot, is a powerful tool.

This plot shows scatter plots for every pair of variables, along with the distribution of each variable along the diagonal.

First, we import the necessary libraries: NumPy for generating random data, Pandas for handling the data in a DataFrame, Matplotlib for plotting, and Seaborn for creating the pair plot.

We then generate synthetic data for four attributes: 'Price', 'Sales', 'Discount', and 'Advertising'. Each attribute contains 100 random values within specified ranges.

We create the DataFrame using Pandas. The DataFrame structure makes it easy to manipulate and visualize the data.

Next, we use the `sns.pairplot()` function from Seaborn to create the scatter matrix plot. This function takes the DataFrame as input and generates the pair plot, showing scatter plots for each pair of variables and histograms for individual variables.

Finally, `plt.show()` displays the plot. This visualization helps in understanding the relationships and correlations between the different attributes.

【Trivia】

Scatter matrix plots are particularly useful in exploratory data analysis (EDA) to identify potential correlations and interactions between variables.

Seaborn's `pairplot()` function simplifies the creation of these plots, providing an intuitive way to visualize complex multivariate data.

Such visualizations are often used in machine learning to understand the data better before building predictive models. They help in detecting patterns, clusters, and outliers in the data.

43. Parallel Coordinates Plot Creation

Importance★★★★☆

Difficulty★★★★☆

A marketing research company wants to analyze customer satisfaction data across multiple product categories to identify patterns and correlations.

They have collected data on customer ratings for four products: Product A, Product B, Product C, and Product D. Each rating ranges from 1 to 5.

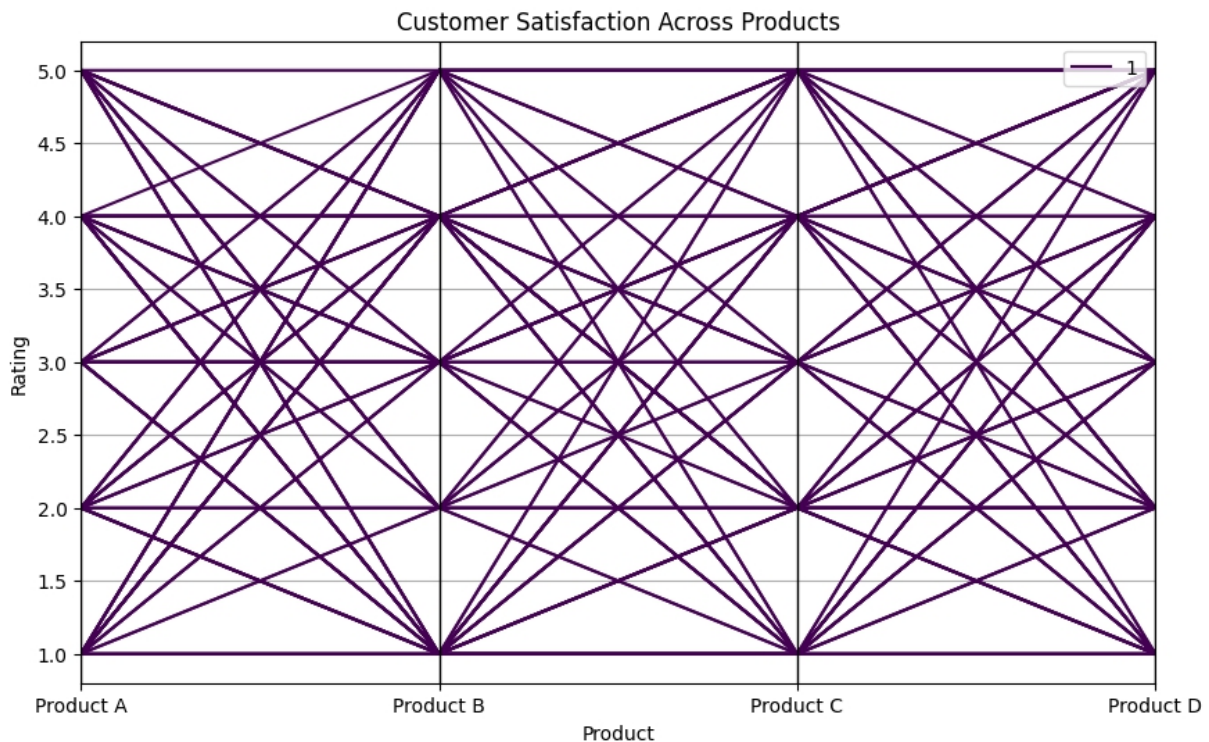
Create a parallel coordinates plot to visualize the relationships and patterns in the ratings.

Generate synthetic data for this analysis, and ensure your code produces a clear and informative plot.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(0)
data = {f'Product {chr(65+i)}': np.random.randint(1, 6,
100) for i in range(4)}
df = pd.DataFrame(data)
```


【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pandas.plotting import parallel_coordinates
np.random.seed(0)
data = {'Product {chr(65+i)}': np.random.randint(1, 6,
100) for i in range(4)}
df = pd.DataFrame(data)
plt.figure(figsize=(10, 6))
parallel_coordinates(df.assign(Group=1),
class_column='Group', colormap='viridis')
plt.title('Customer Satisfaction Across Products')
```

```
plt.xlabel('Product')  
plt.ylabel('Rating')  
plt.show()
```

Parallel coordinates plots are a common way to visualize multivariate data.

In this exercise, we generate synthetic customer satisfaction ratings for four products, ranging from 1 to 5.

Using NumPy, we create random integers to simulate the ratings, ensuring reproducibility with a random seed.

This data is then stored in a pandas DataFrame.

The `parallel_coordinates` function from pandas is used to create the plot.

This function requires a DataFrame and a column name to distinguish different classes.

Since our data does not have classes, we create a dummy class column with a single value.

The `colormap` parameter allows us to specify a color map for the lines in the plot.

We then use Matplotlib to set up the figure and axes, giving the plot a title, and labeling the axes for clarity.

The final step is to display the plot using `plt.show()`.

This exercise helps you understand how to preprocess data and visualize complex relationships using parallel coordinates plots.

【Trivia】

- ▶ Parallel coordinates plots were first introduced by Alfred Inselberg in the 1980s.
- ▶ They are especially useful for detecting clusters and outliers in multivariate data.
- ▶ In a parallel coordinates plot, each axis represents a variable, and each line represents an observation.

▶ These plots can become cluttered with large datasets, so sometimes techniques like brushing and linking are used to enhance readability.

44. Andrews Curves Plot for Multivariate Data Analysis

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst at a retail company. Your manager has asked you to analyze the sales data of different products to identify patterns and trends. You have been provided with a dataset containing sales figures for various products across multiple regions. Your task is to create an Andrews Curves plot to visualize the multivariate data and help identify any patterns or clusters in the sales data. Generate the sample data within your code and then create the Andrews Curves plot. The data should include sales figures for at least four products across three regions. Use the following columns for your dataset: 'Region', 'Product_A', 'Product_B', 'Product_C', 'Product_D'. The 'Region' column should contain categorical data representing different regions (e.g., 'North', 'South', 'East', 'West'). The sales figures should be random integers between 50 and 200.

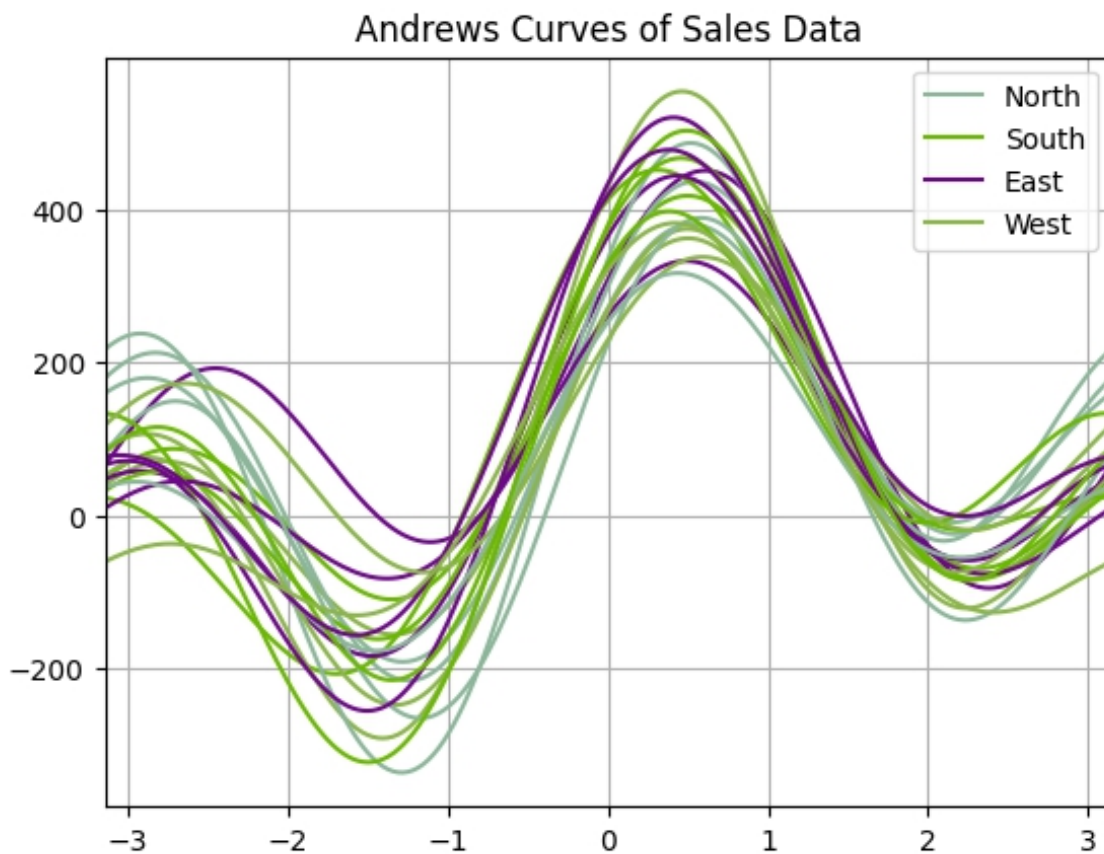
Write the Python code to generate the sample data and create the Andrews Curves plot. The plot should be clearly labeled and should help in visualizing the multivariate data.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
regions = ['North', 'South', 'East', 'West']
data = pd.DataFrame({
    'Region': [regions[i % 4] for i in range(20)],
    'Product_A': np.random.randint(50, 200, 20),
```

```
'Product_B': np.random.randint(50, 200, 20),  
'Product_C': np.random.randint(50, 200, 20),  
'Product_D': np.random.randint(50, 200, 20)  
)
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pandas.plotting import andrews_curves
regions = ['North', 'South', 'East', 'West']
data = pd.DataFrame({
    'Region': [regions[i % 4] for i in range(20)],
    'Product_A': np.random.randint(50, 200, 20),
    'Product_B': np.random.randint(50, 200, 20),
    'Product_C': np.random.randint(50, 200, 20),
```

```
'Product_D': np.random.randint(50, 200, 20)
})
plt.figure()
andrews_curves(data, 'Region')
plt.title('Andrews Curves of Sales Data')
plt.show()
```

Andrews Curves are a way to visualize multivariate data by transforming each observation into a continuous function. This allows for the visualization of high-dimensional data in a two-dimensional plot.

In this exercise, you first generate a sample dataset with sales figures for four products across four regions. The sales figures are random integers between 50 and 200. The dataset is created using pandas, a powerful data manipulation library in Python.

Next, you use the `andrews_curves` function from the `pandas.plotting` module to create the Andrews Curves plot. This function requires the dataset and the column name representing the categorical variable (in this case, 'Region'). The plot is then displayed using `matplotlib.pyplot`. The title of the plot is set to 'Andrews Curves of Sales Data' to provide context.

This exercise helps in understanding how to manipulate data using pandas and visualize it using Andrews Curves. It is particularly useful for identifying patterns and clusters in multivariate data.

【Trivia】

- ▶ Andrews Curves are named after David F. Andrews, who introduced them in 1972.
- ▶ They are particularly useful for detecting outliers and clusters in high-dimensional data.

- ▶ The transformation used in Andrews Curves is based on Fourier series, which is a way to represent a function as a sum of sine and cosine terms.
- ▶ While Andrews Curves can be insightful, they may not always be the best choice for very large datasets due to potential overplotting.

45. RadViz Plot for Multivariate Data Visualization

Importance★★★★☆

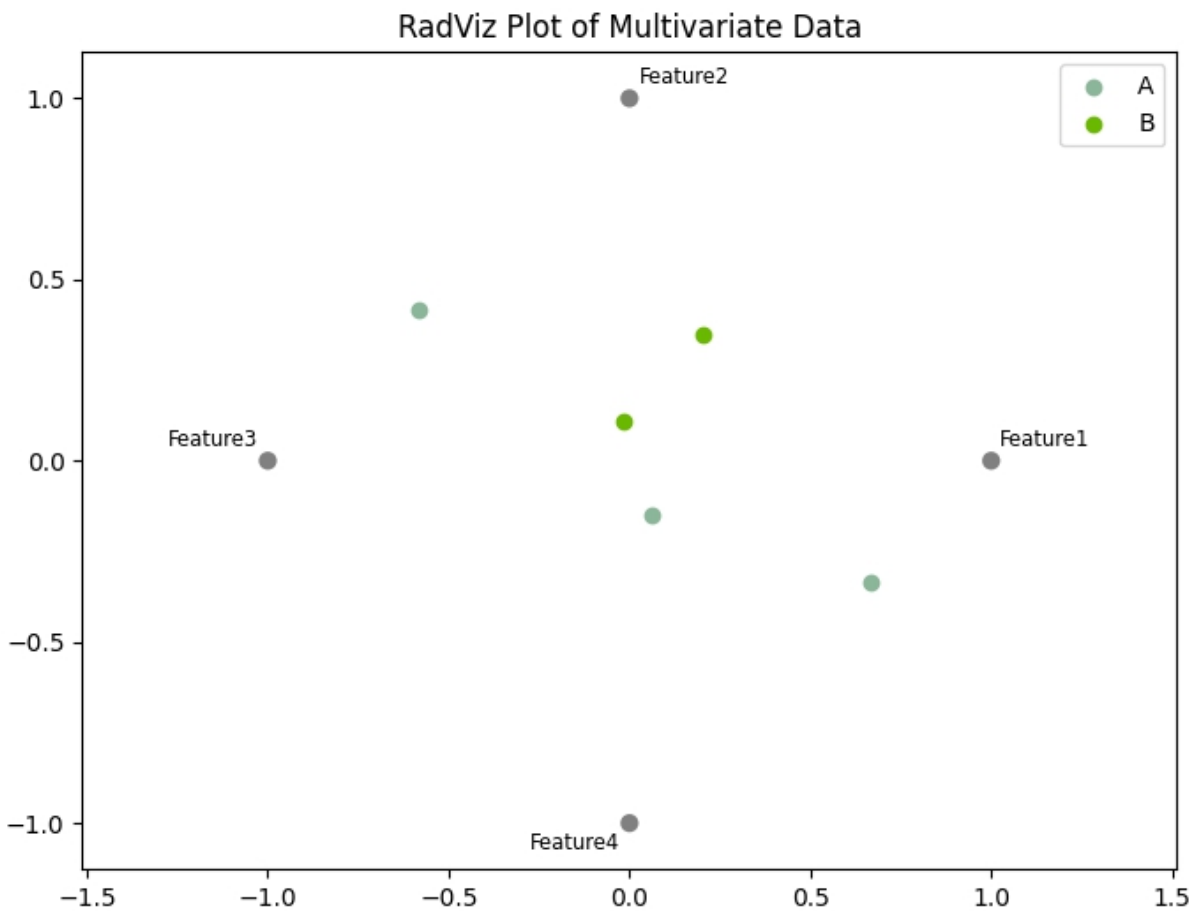
Difficulty★★★★☆

A client working for a data analytics company wants to visualize the relationships between different features of their product using a RadViz plot. They have provided you with a dataset that includes several features. Your task is to create a RadViz plot to help them understand the correlations and distributions of these features. Generate a sample dataset for demonstration purposes and plot it using a RadViz plot.

【Data Generation Code Example】

```
import pandas as pd
## Generate sample data
data = pd.DataFrame({
'Feature1': [0.1, 0.3, 0.5, 0.2, 0.4],
'Feature2': [0.6, 0.7, 0.1, 0.8, 0.5],
'Feature3': [0.9, 0.3, 0.2, 0.4, 0.6],
'Feature4': [0.5, 0.6, 0.7, 0.8, 0.9],
'Category': ['A', 'B', 'A', 'B', 'A']
})
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import matplotlib.pyplot as plt
from pandas.plotting import radviz
## Generate sample data
data = pd.DataFrame({
'Feature1': [0.1, 0.3, 0.5, 0.2, 0.4],
'Feature2': [0.6, 0.7, 0.1, 0.8, 0.5],
'Feature3': [0.9, 0.3, 0.2, 0.4, 0.6],
'Feature4': [0.5, 0.6, 0.7, 0.8, 0.9],
```

```
'Category': ['A', 'B', 'A', 'B', 'A']
})
## Create RadViz plot
plt.figure(figsize=(8, 6))
radviz(data, 'Category')
plt.title('RadViz Plot of Multivariate Data')
plt.show()
```

To create a RadViz plot, we first import the necessary libraries, including pandas for data manipulation, matplotlib for plotting, and radviz from pandas.plotting for creating the RadViz plot.

Next, we generate a sample dataset using pandas' DataFrame. The dataset contains four features (Feature1, Feature2, Feature3, Feature4) and a categorical column ('Category') that indicates the class of each data point. We then use the radviz function from pandas.plotting to create the RadViz plot. The function takes two arguments: the dataset and the column name that contains the categorical data. The radviz function automatically plots the data points in a circular layout, where each feature is positioned at a point on the circumference of the circle. Finally, we use matplotlib's plt.show() function to display the plot. The plt.title() function is used to add a title to the plot for better understanding.

The RadViz plot helps in visualizing multivariate data by displaying the relationships and distributions of different features in a circular layout. This plot can be particularly useful for identifying patterns and correlations between features in a dataset.

【Trivia】

RadViz, short for Radial Visualization, is a method used to project multidimensional data onto a 2D plane. The

technique places each feature at equidistant points on the circumference of a circle. Each data point is plotted inside the circle based on the weighted sum of the features. This visualization method is particularly effective in identifying clusters and patterns in multivariate datasets. RadViz is commonly used in fields such as data mining, bioinformatics, and finance for exploratory data analysis.

46. Creating a Lag Plot for Time Series Analysis

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst working for an e-commerce company. The company wants to understand the relationship between daily sales volumes over time.

They have asked you to create a lag plot to visualize the correlation between sales on consecutive days.

Your task is to:

Generate a sample dataset of daily sales for the last 100 days.

Create a lag plot using this data, with a lag of 1 day.

Add appropriate labels and a title to the plot.

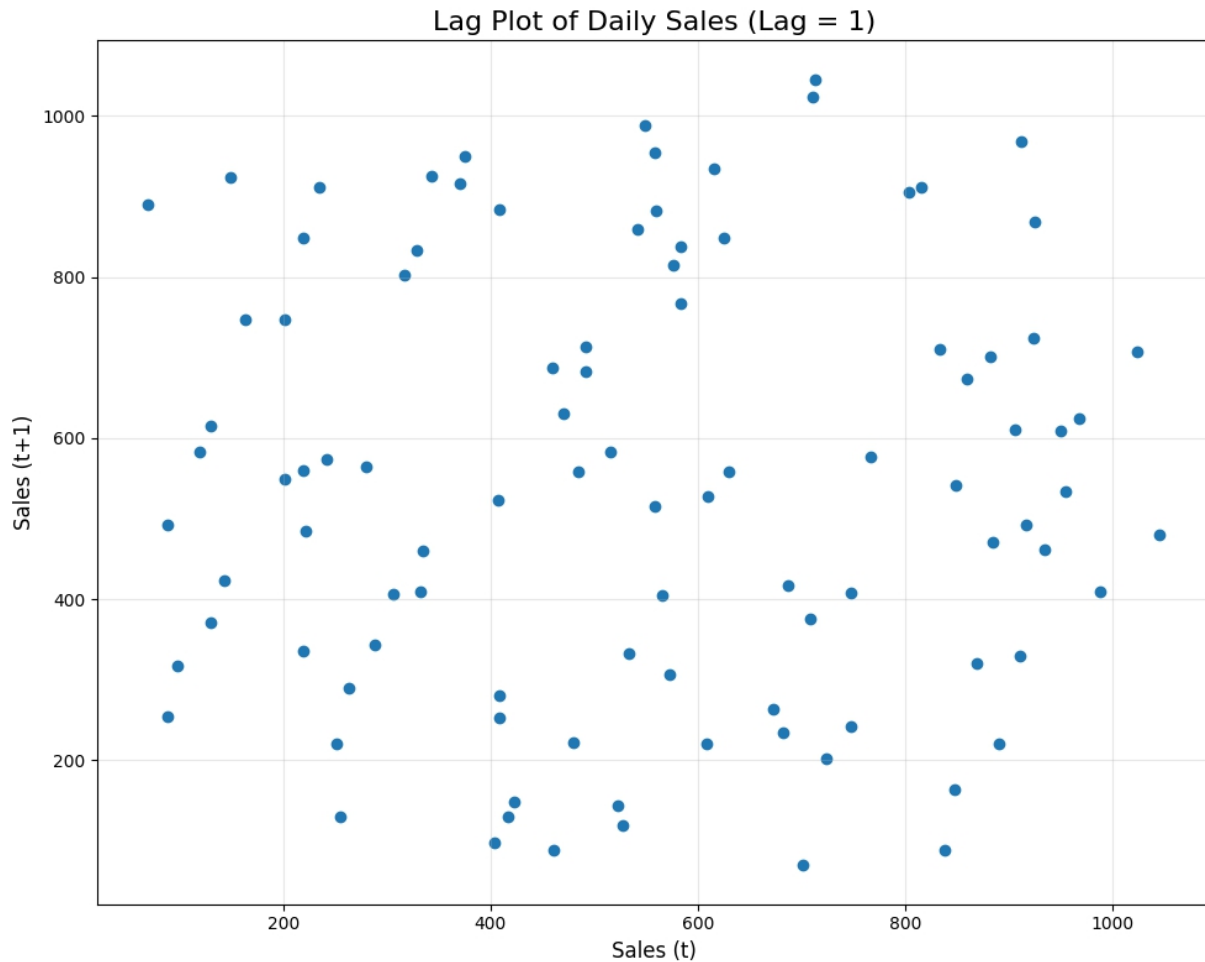
Ensure the plot is visually appealing and easy to interpret.

Use the following code to generate the sample data, then write a Python script to create the required lag plot.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
np.random.seed(42)
date_range =
pd.date_range(end=pd.Timestamp.now().floor('D'),
periods=100)
sales = np.random.randint(100, 1000, size=100) +
np.sin(np.arange(100) * 0.3) * 50
df = pd.DataFrame({'Date': date_range, 'Sales':
sales.astype(int)})
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas.plotting import lag_plot
np.random.seed(42)
date_range =
pd.date_range(end=pd.Timestamp.now().floor('D'),
periods=100)
```

```
sales = np.random.randint(100, 1000, size=100) +  
np.sin(np.arange(100) * 0.3) * 50  
df = pd.DataFrame({'Date': date_range, 'Sales':  
sales.astype(int)})  
plt.figure(figsize=(10, 8))  
lag_plot(df['Sales'], lag=1)  
plt.title('Lag Plot of Daily Sales (Lag = 1)', fontsize=16)  
plt.xlabel('Sales (t)', fontsize=12)  
plt.ylabel('Sales (t+1)', fontsize=12)  
plt.grid(True, alpha=0.3)  
plt.tight_layout()  
plt.show()
```

This code creates a lag plot for time series analysis of daily sales data.

Let's break down the solution and explain each part in detail:

Data Generation:

We use NumPy and Pandas to create a sample dataset. `np.random.seed(42)` ensures reproducibility of the random data.

We generate dates for the last 100 days using `pd.date_range()`.

Sales data is created using random integers and a sine function to add some cyclical pattern.

The data is stored in a Pandas DataFrame with 'Date' and 'Sales' columns.

Importing Required Libraries:

We import NumPy for numerical operations, Pandas for data manipulation, Matplotlib for plotting, and the `lag_plot` function from Pandas plotting module.

Creating the Lag Plot:

`plt.figure(figsize=(10, 8))` sets up a new figure with specified dimensions.

`lag_plot(df['Sales'], lag=1)` creates the lag plot using the 'Sales' column from our DataFrame.

The `lag=1` parameter means we're comparing each day's sales with the previous day's sales.

Customizing the Plot:

`plt.title()` adds a title to the plot.

`plt.xlabel()` and `plt.ylabel()` label the x and y axes.

`plt.grid(True, alpha=0.3)` adds a light grid to the plot for better readability.

`plt.tight_layout()` adjusts the plot layout to prevent overlapping elements.

Displaying the Plot:

`plt.show()` displays the final plot.

The resulting lag plot helps visualize the correlation between sales on consecutive days.

Each point on the plot represents two consecutive days, with the x-coordinate being the sales on one day and the y-coordinate being the sales on the following day.

Interpreting the Lag Plot:

If points cluster along a diagonal line from bottom-left to top-right, it indicates a positive correlation between consecutive days' sales.

A random scatter suggests little to no correlation.

Any visible patterns (like clusters or curves) can indicate more complex relationships or seasonality in the data.

This visualization is valuable for understanding time series patterns and can help in forecasting future sales based on past performance.

【Trivia】

- ▶ Lag plots are a powerful tool in time series analysis, helping to identify autocorrelation in data.
- ▶ The concept of "lag" in time series refers to the time difference between observations. A lag of 1 compares each observation with the immediately preceding one.
- ▶ Lag plots can reveal various patterns:
 - A diagonal line indicates strong autocorrelation
 - A circular pattern might suggest a cyclical trend
 - A random scatter implies no significant autocorrelation
- ▶ In finance and economics, lag plots are often used to analyze stock prices, economic indicators, and other time-dependent variables.
- ▶ The `pandas.plotting.lag_plot()` function is a convenient way to create lag plots in Python, but you can also create them manually using scatter plots.
- ▶ Lag plots are closely related to autocorrelation plots (ACF plots), another important tool in time series analysis.
- ▶ When working with real-world data, it's often useful to create multiple lag plots with different lag values to understand short-term and long-term patterns.
- ▶ Lag plots can help in identifying outliers or anomalies in time series data, as these will appear as points far from the main cluster.
- ▶ The interpretation of lag plots can be enhanced by combining them with other time series techniques like decomposition or spectral analysis.
- ▶ In machine learning, features derived from lag analysis can be valuable inputs for time series forecasting models.

47. Autocorrelation Plot of Time Series Data

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst at a retail company. Your manager has asked you to analyze the sales data to identify any patterns or trends. Specifically, they want you to generate an autocorrelation plot to understand how sales figures correlate with themselves over different time lags.

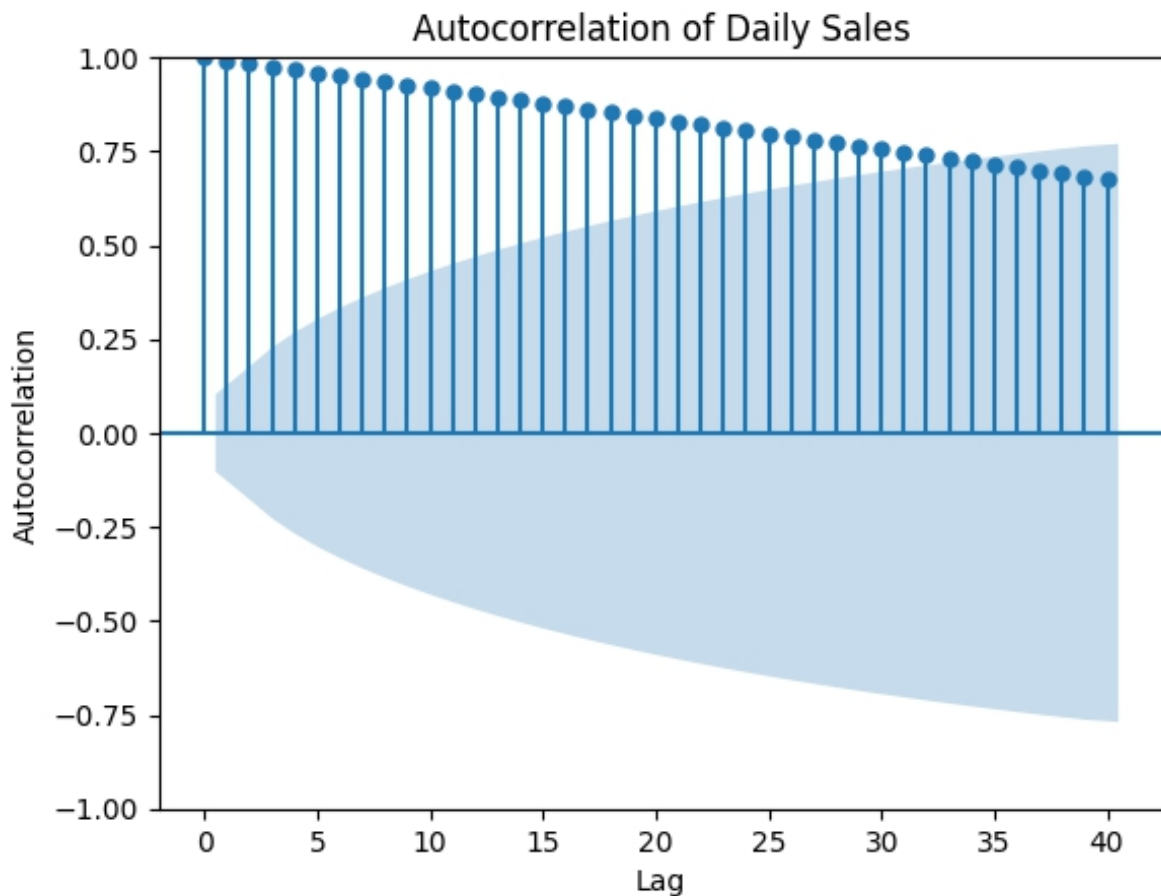
To help you get started, you need to create a synthetic time series dataset representing daily sales for one year. Then, generate an autocorrelation plot using Python to visualize the data.

Write the code to create the dataset and generate the autocorrelation plot.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.graphics.tsaplots import plot_acf
np.random.seed(0)
days = 365
sales = np.cumsum(np.random.randn(days) * 10 + 100)
data = pd.Series(sales)
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.graphics.tsaplots import plot_acf
np.random.seed(0)
days = 365
sales = np.cumsum(np.random.randn(days) * 10 + 100)
data = pd.Series(sales)
plt.figure(figsize=(10, 6))
```

```
plot_acf(data, lags=40)
plt.title('Autocorrelation of Daily Sales')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.show()
```

To generate an autocorrelation plot of time series data, we first need to create a synthetic dataset.

We use the numpy library to generate random sales data for 365 days. The np.cumsum function is used to create a cumulative sum of normally distributed random numbers, simulating daily sales figures that have an underlying trend. The pandas library is then used to convert this array into a Series object for easier manipulation.

Next, we use the plot_acf function from the statsmodels library to generate the autocorrelation plot. This function computes the autocorrelation of the time series data for different lags and plots the results.

The matplotlib library is used to customize the plot, setting the figure size and adding titles and labels to make the plot more informative. The plt.show() function is called to display the plot.

Autocorrelation plots are useful for identifying patterns in time series data, such as seasonality or trends, by showing how the data correlates with itself over different time lags.

【Trivia】

- ▶ Autocorrelation, also known as serial correlation, is the correlation of a signal with a delayed copy of itself as a function of delay.
- ▶ In time series analysis, autocorrelation can help identify repeating patterns, such as seasonal effects, and is crucial for model selection in forecasting.

- ▶ The statsmodels library in Python provides various tools for time series analysis, including functions for autocorrelation, partial autocorrelation, and more advanced time series models like ARIMA.

48. Bootstrap Plot of Statistical Data

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst at a retail company. Your manager has asked you to analyze the sales data to understand the distribution of sales and to create a bootstrap plot to visualize this distribution.

Generate a sample dataset of daily sales figures for the past year (365 days).

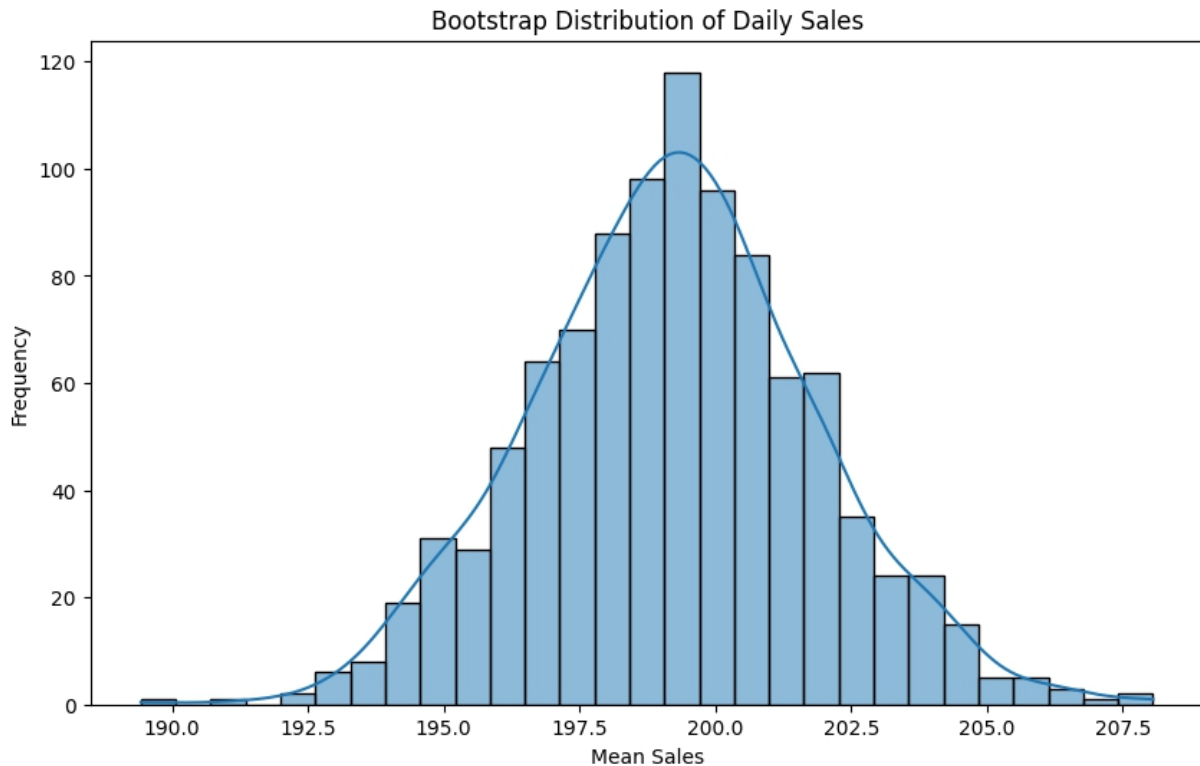
Use this data to create a bootstrap plot that shows the distribution of the sales data.

Ensure that the plot is clear and informative.

【Data Generation Code Example】

```
import numpy as np
np.random.seed(0)
daily_sales = np.random.normal(loc=200, scale=50,
size=365)
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
np.random.seed(0)
daily_sales = np.random.normal(loc=200, scale=50,
size=365)
bootstrap_samples = [np.random.choice(daily_sales,
size=365, replace=True).mean() for _ in range(1000)]
plt.figure(figsize=(10, 6))
sns.histplot(bootstrap_samples, kde=True)
plt.title('Bootstrap Distribution of Daily Sales')
plt.xlabel('Mean Sales')
```

```
plt.ylabel('Frequency')  
plt.show()
```

First, we import the necessary libraries: numpy, matplotlib.pyplot, and seaborn.

We use numpy to generate a sample dataset of daily sales figures for the past year.

The sales data is generated using a normal distribution with a mean (loc) of 200 and a standard deviation (scale) of 50.

We set the random seed to ensure reproducibility.

Next, we create bootstrap samples from the daily sales data.

Bootstrap sampling involves randomly selecting data points with replacement to create new samples.

We generate 1000 bootstrap samples, each of size 365 (the same as the original dataset), and calculate the mean of each sample.

We then use seaborn's histplot function to create a histogram of the bootstrap sample means.

The kde=True parameter adds a Kernel Density Estimate (KDE) plot, which provides a smooth curve representing the distribution.

We set the figure size to make the plot more readable and add titles and labels to the plot for clarity.

Finally, we display the plot using plt.show().

【Trivia】

- ▶ Bootstrap sampling is a powerful statistical technique used to estimate the distribution of a statistic (e.g., mean, median) by resampling with replacement from the original data.
- ▶ It is particularly useful when the sample size is small or when the underlying distribution is unknown.

- ▶ The term "bootstrap" comes from the phrase "pulling oneself up by one's bootstraps," reflecting the method's ability to generate estimates from the data itself without relying on external assumptions.
- ▶ Seaborn is a Python visualization library based on matplotlib that provides a high-level interface for drawing attractive statistical graphics.

49. Creating a Hexbin Plot with Pandas

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst working for a retail company. Your task is to analyze the relationship between the amount spent by customers and the number of items they purchase. You decide to use a hexbin plot to visualize the density of points where most customers' spending and purchase behavior cluster.

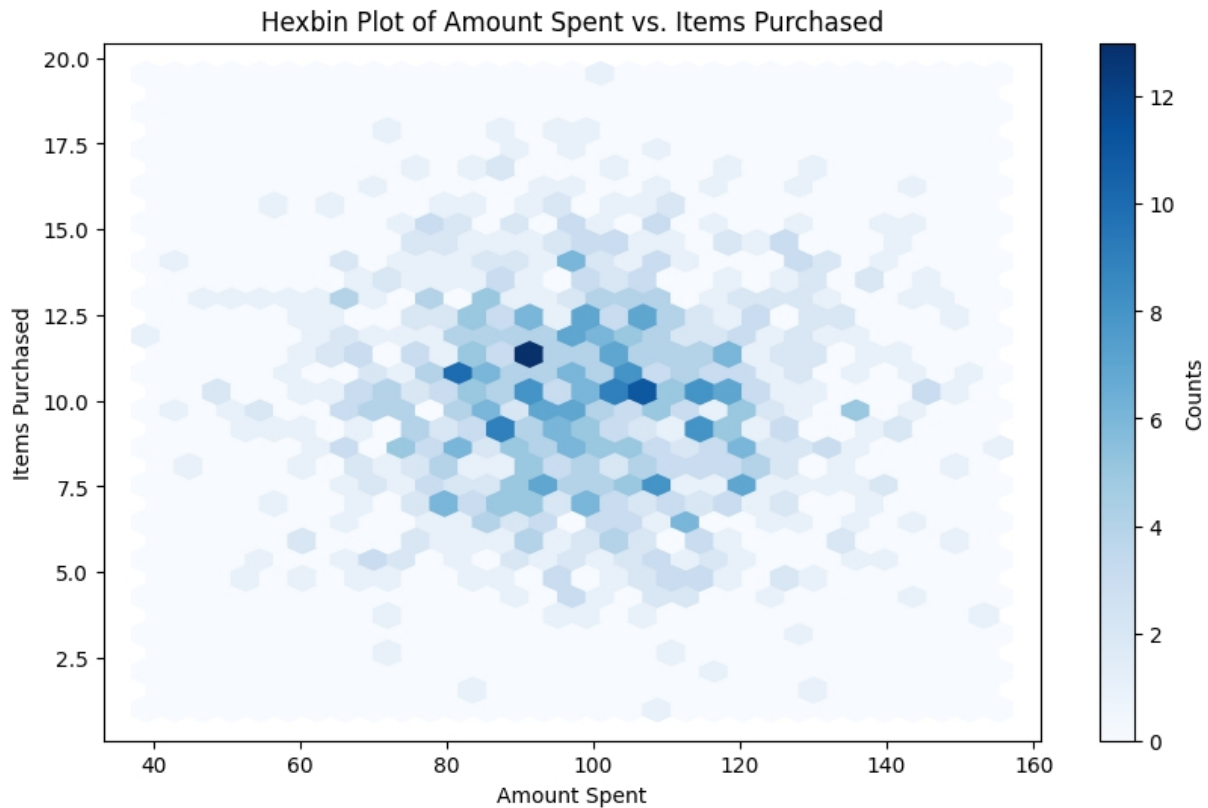
Create a sample dataset with two columns: 'AmountSpent' (amount of money spent by the customers) and 'ItemsPurchased' (number of items purchased by the customers).

Generate a hexbin plot using this data to help your team understand the spending patterns and purchasing behavior of the customers.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
Create sample data np.random.seed(0)
amount_spent = np.random.normal(100, 20, 1000)
items_purchased = np.random.normal(10, 3, 1000)
Create DataFrame df = pd.DataFrame({'AmountSpent':
amount_spent, 'ItemsPurchased': items_purchased})
df.head()
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
amount_spent = np.random.normal(100, 20, 1000)
items_purchased = np.random.normal(10, 3, 1000)
df = pd.DataFrame({'AmountSpent': amount_spent,
                  'ItemsPurchased': items_purchased})
plt.figure(figsize=(10, 6))
plt.hexbin(df['AmountSpent'], df['ItemsPurchased'],
          gridsize=30, cmap='Blues')
```

```
plt.colorbar(label='Counts')
plt.xlabel('Amount Spent')
plt.ylabel('Items Purchased')
plt.title('Hexbin Plot of Amount Spent vs. Items Purchased')
plt.show()
```

A hexbin plot is a two-dimensional histogram that is useful for visualizing the relationship between two variables when you have a large number of data points. Instead of plotting each individual data point, which can result in overplotting, a hexbin plot groups points into hexagonal bins and colors them according to the number of points in each bin. This allows you to see the density and distribution of the data more clearly.

First, we import the necessary libraries: pandas for data manipulation, numpy for generating random data, and matplotlib for plotting.

We then set a random seed for reproducibility and generate random data for the 'AmountSpent' and 'ItemsPurchased' columns using a normal distribution. 'AmountSpent' is generated with a mean of 100 and a standard deviation of 20, while 'ItemsPurchased' is generated with a mean of 10 and a standard deviation of 3.

We create a DataFrame with this data and then use matplotlib to create the hexbin plot. We specify the x and y variables, set the grid size of the hexagons, and choose a color map ('Blues'). The color bar is added to indicate the counts in each bin, and we label the axes and the plot for clarity. Finally, we display the plot. This hexbin plot helps visualize the density of customer spending and purchasing behavior, highlighting areas where most customers' behaviors cluster.

【Trivia】

Hexbin plots are particularly useful in cases where scatter plots fail to provide clear insights due to overplotting, which happens when too many points overlap, making it difficult to discern patterns.

The hexagonal binning technique was popularized by the statistical software package Hexbin in the 1980s, providing a way to manage large datasets in a visually interpretable manner.

Hexbin plots are especially useful in fields such as astronomy, where data from observations often involve large datasets with overlapping points.

50. Creating a Scatter Plot Matrix for Customer Data Analysis

Importance★★★★☆

Difficulty★★★☆☆

You are a data analyst for an e-commerce company. The marketing team wants to understand the relationships between various customer attributes and their spending behavior.

They have provided you with a dataset containing information on customer age, years as a customer, annual income, and total spending.

Your task is to create a scatter plot matrix using pandas to visualize the relationships between these variables.

Specifically, you need to:

Create a pandas DataFrame with the following columns:

'Age', 'Years_as_Customer', 'Annual_Income', and 'Total_Spending'.

Generate sample data for 100 customers.

Use pandas and seaborn to create a scatter plot matrix.

Ensure the diagonal plots show histograms of each variable.

Add appropriate labels and a title to the plot.

Write a Python script that accomplishes these tasks and displays the scatter plot matrix.

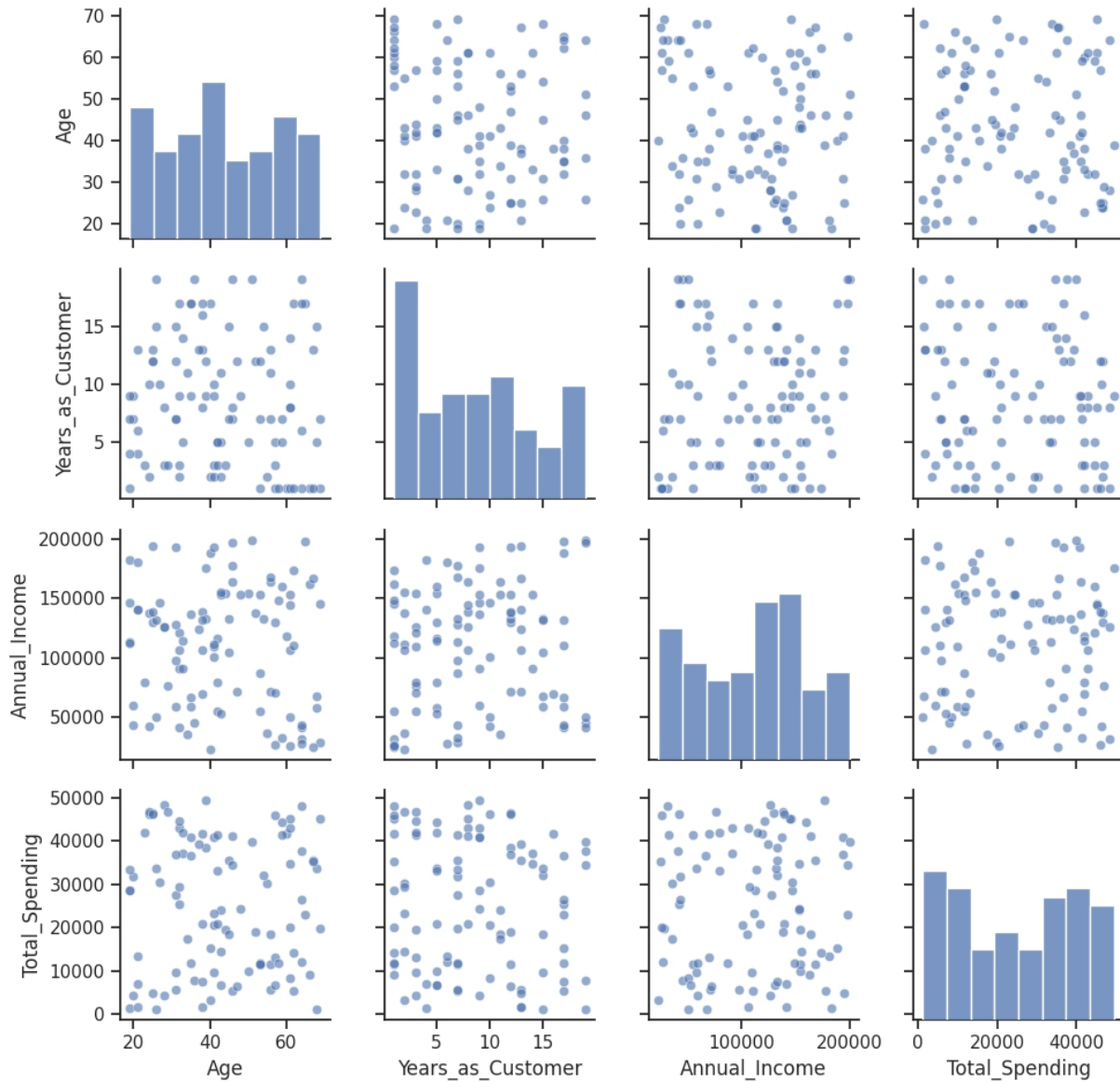
【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(42)
data = pd.DataFrame({
    'Age': np.random.randint(18, 70, 100),
    'Years_as_Customer': np.random.randint(1, 20, 100),
```

```
'Annual_Income': np.random.randint(20000, 200000, 100),  
'Total_Spending': np.random.randint(1000, 50000, 100)  
})
```

【Diagram Answer】

Customer Data Scatter Plot Matrix



【Code Answer】

```
import pandas as pd
import numpy as np
import seaborn as sns
```



```

import matplotlib.pyplot as plt
np.random.seed(42)
data = pd.DataFrame({
    'Age': np.random.randint(18, 70, 100),
    'Years_as_Customer': np.random.randint(1, 20, 100),
    'Annual_Income': np.random.randint(20000, 200000, 100),
    'Total_Spending': np.random.randint(1000, 50000, 100)
})
# Set up the plot style and size
sns.set(style="ticks", color_codes=True)
plt.figure(figsize=(12, 10))
# Create the scatter plot matrix
scatter_matrix = sns.pairplot(data, diag_kind="hist",
    plot_kws={"alpha": 0.6})
# Set the title
scatter_matrix.fig.suptitle("Customer Data Scatter Plot
Matrix", y=1.02)
# Adjust layout and display the plot
plt.tight_layout()
plt.show()

```

This code creates a scatter plot matrix to visualize relationships between customer attributes.

Let's break down the key components and explain the data processing and visualization techniques used:

Data Generation:

We use numpy's random functions to generate sample data for 100 customers.

The data includes age (18-70), years as a customer (1-20), annual income (20,000-200,000), and total spending (1,000-

50,000).

This simulated data is stored in a pandas DataFrame, which is a 2D labeled data structure.

Data Visualization Setup:

We import seaborn (sns) and matplotlib.pyplot (plt) for advanced data visualization.

seaborn is built on top of matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Plotting:

We use seaborn's pairplot function to create the scatter plot matrix.

pairplot creates a grid of axes with each variable in the dataset shared across the y-axes across a single row and the x-axes across a single column.

The diagonal plots are set to show histograms (diag_kind="hist") instead of scatter plots, providing a distribution view of each variable.

Customization:

We set the style to "ticks" and enable color codes for better visual appeal.

The figure size is set to 12x10 inches for better readability.

We add a title to the entire plot using fig.suptitle().

The alpha parameter in plot_kws sets the transparency of the points, helping to visualize overlapping data points.

Display:

plt.tight_layout() adjusts the plot to ensure all labels are visible.

plt.show() displays the final plot.

This scatter plot matrix allows for quick visual analysis of relationships between all pairs of variables.

Each cell shows the relationship between two variables, with the variable names on the diagonal.

This is particularly useful for identifying correlations, clusters, or outliers in the data.

【Trivia】

- ▶ Scatter plot matrices, also known as SPLOM (Scatter PLOt Matrix), were introduced by John W. Tukey and Paul A. Tukey in 1981.
- ▶ The diagonal elements in a scatter plot matrix often show the distribution of a single variable. Histograms are common, but kernel density estimates or box plots can also be used.
- ▶ Seaborn's pairplot function is a wrapper around PairGrid, which offers more flexibility for customizing the plot.
- ▶ When dealing with large datasets, consider using hexbin plots or 2D kernel density estimation instead of scatter plots to avoid overplotting.
- ▶ Color coding points based on a categorical variable can add an extra dimension to the scatter plot matrix, revealing group-specific patterns.
- ▶ For datasets with many variables, consider using dimensionality reduction techniques like PCA before creating a scatter plot matrix to focus on the most important relationships.

51. Generate a Box Plot Using Pandas

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst working for a retail company. Your manager has asked you to analyze the sales data for different product categories to understand their distribution. Generate a box plot to visualize the distribution of sales for each product category.

Create a sample dataset with the following columns: 'Category' (with values 'Electronics', 'Clothing', 'Groceries') and 'Sales' (random integers between 100 and 1000).

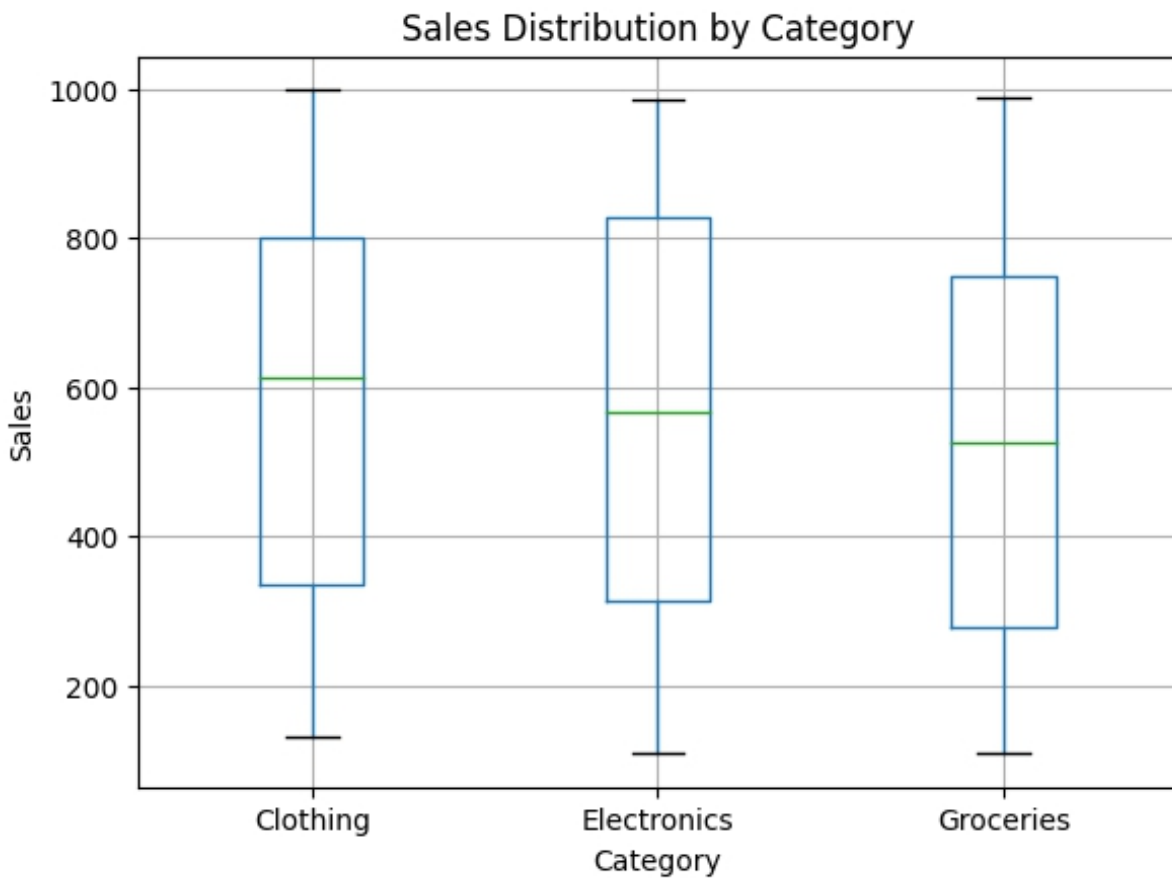
Use this dataset to create the box plot.

Ensure the box plot is labeled appropriately with titles and axis labels.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(0)
categories = ['Electronics', 'Clothing', 'Groceries']
data = {'Category': [categories[i % 3] for i in range(300)],
        'Sales': np.random.randint(100, 1000, 300)}
df = pd.DataFrame(data)
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
categories = ['Electronics', 'Clothing', 'Groceries']
data = {'Category': [categories[i % 3] for i in range(300)],
'Sales': np.random.randint(100, 1000, 300)}
df = pd.DataFrame(data)
plt.figure(figsize=(10, 6))
df.boxplot(column='Sales', by='Category')
```

```
plt.title('Sales Distribution by Category')
plt.suptitle('')
plt.xlabel('Category')
plt.ylabel('Sales')
plt.show()
```

To generate a box plot using Pandas, you first need to create a DataFrame with the necessary data.

In this case, the DataFrame contains two columns: 'Category' and 'Sales'.

The 'Category' column includes three different product categories: 'Electronics', 'Clothing', and 'Groceries'.

The 'Sales' column contains random integers between 100 and 1000 to simulate sales data.

The box plot is created using the boxplot method of the DataFrame, specifying 'Sales' as the column to plot and 'Category' as the grouping variable.

The plt.figure function is used to set the size of the plot, and plt.show is called to display the plot.

The plot is labeled with a title, and the x and y axes are labeled appropriately to make the plot informative.

【Trivia】

- ▶ Box plots are also known as whisker plots.
- ▶ They provide a graphical summary of data, showing the median, quartiles, and potential outliers.
- ▶ John Tukey, an American mathematician, introduced the box plot in 1977.
- ▶ Box plots are particularly useful for comparing distributions across multiple groups or categories.


```
sns.violinplot(x='Category', y='Sales', data=data)
plt.title('Violin Plot of Daily Sales Distribution')
plt.show()
```

To generate the sample data, we use NumPy to create two sets of random numbers representing sales figures for 'Category A' and 'Category B'.

Each category has 30 daily sales data points. 'Category A' sales are normally distributed with a mean of 200 and a standard deviation of 30, while 'Category B' sales have a mean of 150 and a standard deviation of 20.

This data is combined into a single DataFrame with columns for 'Category' and 'Sales'.

To create the violin plot, we use the Seaborn library, which provides a high-level interface for drawing attractive statistical graphics.

We call the `sns.violinplot` function, specifying the DataFrame and the columns to use for the x-axis (Category) and y-axis (Sales).

Finally, we set the title of the plot and use `plt.show()` to display the plot.

The violin plot provides a visual comparison of the sales distributions for the two categories. It combines aspects of a box plot and a kernel density plot, showing the distribution's shape, central tendency, and variability.

【Trivia】

Violin plots are particularly useful for comparing multiple categories of data because they display the entire distribution.

They are more informative than box plots alone because they show the density of the data at different values, which can reveal multimodal distributions (distributions with multiple peaks).

Seaborn's violin plot function also allows for additional customization, such as splitting the violins for each category to compare two distributions side by side.

53. Plotting a KDE Plot Using Pandas

Importance★★★★☆

Difficulty★★★★☆

You are working as a data analyst for a retail company. Your manager has asked you to analyze the distribution of the total purchase amounts made by customers. To better understand the distribution, you need to create a Kernel Density Estimate (KDE) plot. Using Python and pandas, generate a sample dataset of customer purchase amounts and create a KDE plot to visualize the distribution.

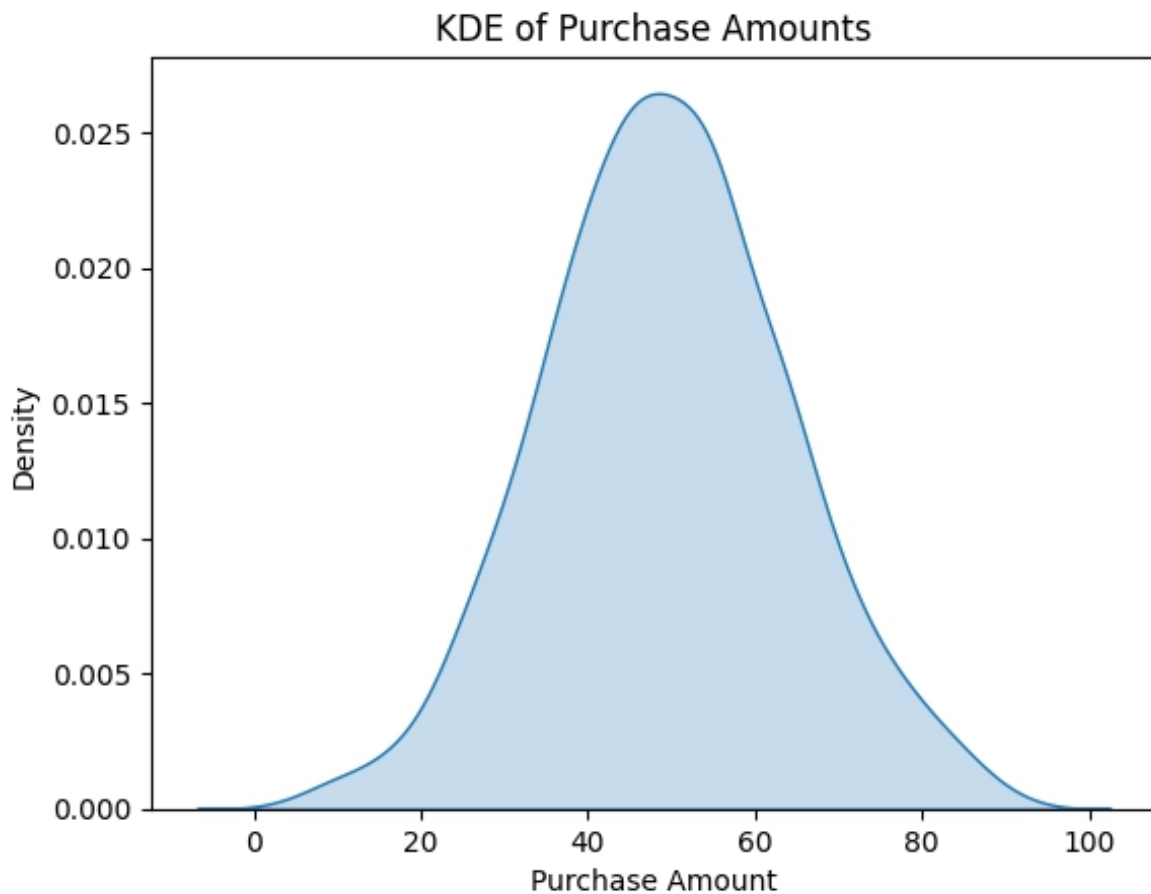
The dataset should have a column named 'PurchaseAmount' with 1000 randomly generated purchase amounts following a normal distribution with a mean of 50 and a standard deviation of 15.

Write a code to create this dataset and plot the KDE. Ensure your plot is clear and properly labeled.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(0)
data = pd.DataFrame({'PurchaseAmount':
np.random.normal(50, 15, 1000)})
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
np.random.seed(0)
data = pd.DataFrame({'PurchaseAmount':
np.random.normal(50, 15, 1000)})
sns.kdeplot(data['PurchaseAmount'], shade=True)
plt.title('KDE of Purchase Amounts')
plt.xlabel('Purchase Amount')
```

```
plt.ylabel('Density')  
plt.show()
```

Kernel Density Estimation (KDE) is a non-parametric way to estimate the probability density function of a random variable.

In this task, we are using KDE to visualize the distribution of purchase amounts in a retail dataset.

First, we import necessary libraries: pandas for data manipulation, numpy for generating random data, matplotlib for plotting, and seaborn for creating the KDE plot.

We set a random seed to ensure the reproducibility of our results.

Next, we create a DataFrame with one column, 'PurchaseAmount', containing 1000 randomly generated values from a normal distribution with a mean of 50 and a standard deviation of 15.

To plot the KDE, we use seaborn's kdeplot function, passing in our 'PurchaseAmount' data.

The 'shade' parameter is set to True to fill the area under the KDE curve.

We then set the title and labels for the x and y axes to make the plot informative.

Finally, we display the plot using plt.show().

【Trivia】

- ▶ KDE plots are useful for visualizing the distribution of data without making assumptions about the underlying distribution shape, unlike histograms which can be sensitive to bin size and placement.
- ▶ Seaborn, built on top of matplotlib, provides a high-level interface for drawing attractive and informative statistical graphics.

- ▶ In addition to KDE plots, seaborn also supports other types of plots like bar plots, box plots, and pair plots, making it a versatile tool for data visualization.
- ▶ KDE can be particularly useful when dealing with small datasets, as it can give a smoother estimate of the distribution compared to a histogram.

54. Creating a Density Plot with Pandas

Importance★★★★☆

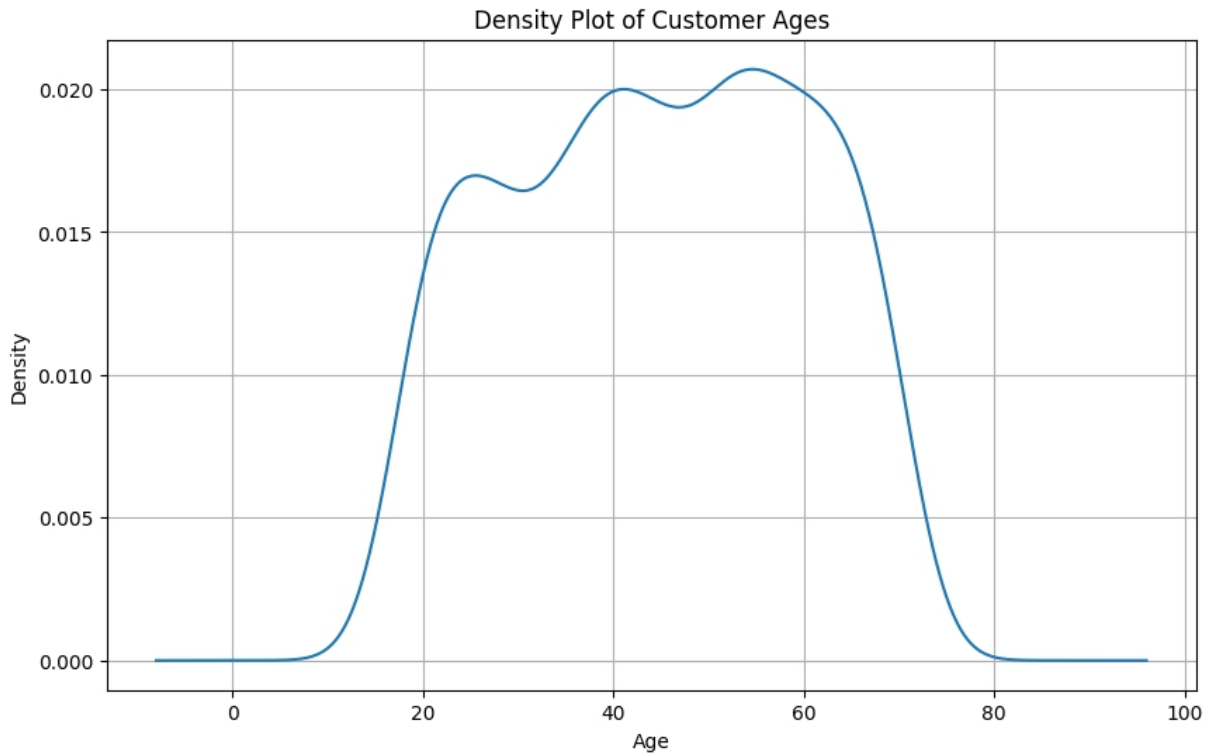
Difficulty★★★★☆

You are working as a data analyst for a retail company. The company wants to analyze the distribution of customer ages to better understand their target demographic. Your task is to create a density plot using a sample dataset of customer ages. First, generate a sample dataset of 1000 customer ages ranging from 18 to 70. Then, using this dataset, create a density plot to visualize the age distribution. Provide the Python code that generates the dataset and the density plot.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
ages = pd.Series(np.random.randint(18, 71, size=1000))
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd # Importing necessary libraries
import numpy as np # Importing necessary libraries
import matplotlib.pyplot as plt # Importing necessary
libraries
ages = pd.Series(np.random.randint(18, 71, size=1000)) #
# Creating a sample dataset of customer ages
plt.figure(figsize=(10, 6)) # # Setting the figure size
ages.plot(kind='density') # # Creating a density plot
plt.title('Density Plot of Customer Ages') # # Adding a title
to the plot
plt.xlabel('Age') # # Adding an x-axis label
plt.ylabel('Density') # # Adding a y-axis label
```



```
plt.grid(True) # # Adding a grid for better readability  
plt.show() # # Displaying the plot
```

To create a density plot using Pandas, follow these steps: First, you need to generate a sample dataset. This can be done using numpy to create a series of random integers representing ages. In the sample code, `numpy.random.randint` generates 1000 random integers between 18 and 70, which are then converted to a Pandas Series.

Once the data is prepared, you can create a density plot using the `plot` method with `kind='density'`. This method computes and plots a Kernel Density Estimate (KDE), which is a smoothed approximation of the data distribution. To make the plot more informative, set the figure size using `plt.figure`, and add titles and labels for the axes using `plt.title`, `plt.xlabel`, and `plt.ylabel`. Adding a grid with `plt.grid(True)` improves readability. Finally, `plt.show()` displays the plot.

This process is useful for visualizing the distribution of data points and identifying patterns or outliers. Density plots are especially helpful when comparing distributions across different datasets.

【Trivia】

- ▶ Kernel Density Estimation (KDE) is a non-parametric way to estimate the probability density function of a random variable. It is used to smooth the distribution of data points.
- ▶ Density plots are similar to histograms, but they provide a continuous estimation of the distribution, making them more suitable for identifying the underlying pattern of data.
- ▶ In data analysis, understanding the distribution of data is crucial for making informed decisions, as it reveals insights

about the spread, central tendency, and variability of the dataset.

55. Bar Plot Visualization with Pandas

Importance★★★★☆

Difficulty★★☆☆☆

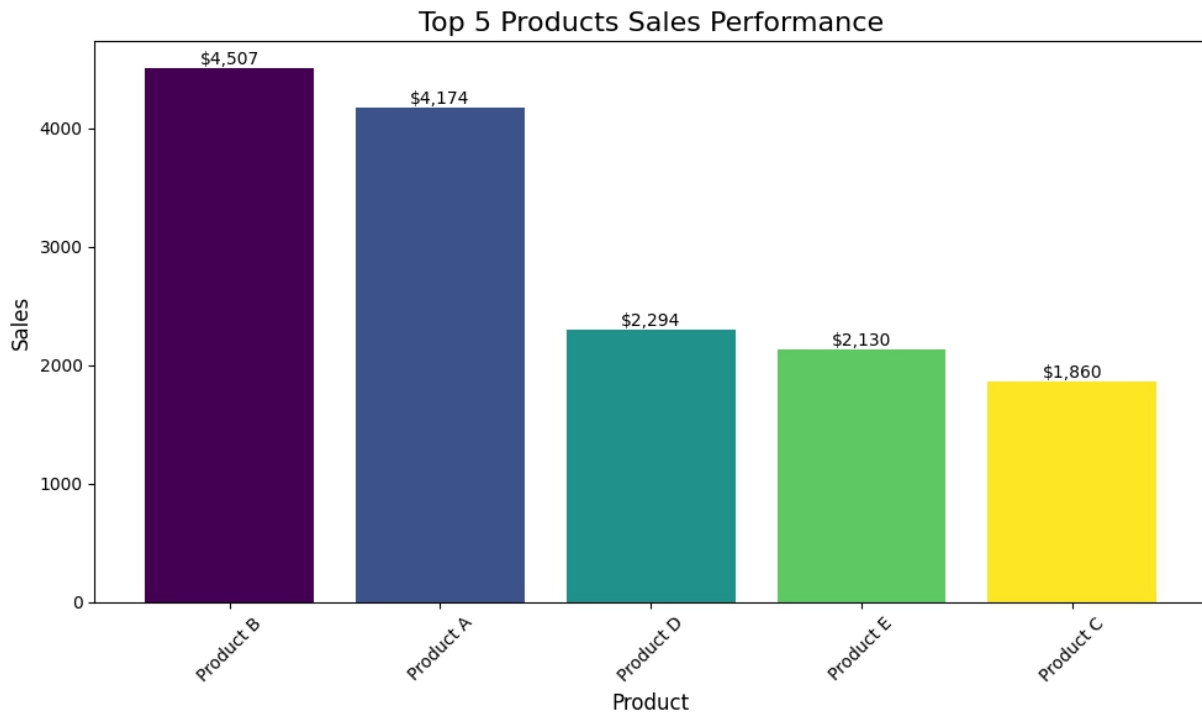
You are a data analyst at a retail company. The company wants to visualize the monthly sales data for the first half of the year to identify trends and compare sales performance. Your task is to create a bar plot using Pandas to display the sales data for January to June.

Generate the following sales data: January: \$10,000 February: \$12,000 March: \$9,000 April: \$15,000 May: \$8,000 June: \$14,000 Create a bar plot to visualize this data.

【Data Generation Code Example】

```
import pandas as pd
data = {'Month': ['January', 'February', 'March', 'April', 'May', 'June'], 'Sales': [10000, 12000, 9000, 15000, 8000, 14000]}
df = pd.DataFrame(data)
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.DataFrame({'Month': ['January', 'February', 'March',
'April', 'May', 'June'], 'Sales': [10000, 12000, 9000, 15000,
8000, 14000]})
df.plot(kind='bar', x='Month', y='Sales', legend=False)
plt.title('Monthly Sales Data')
plt.xlabel('Month')
plt.ylabel('Sales ($)')
plt.show()
```

First, we create a Pandas DataFrame with the given sales data.

The data consists of two columns: 'Month' and 'Sales'. The 'Month' column includes the names of the months from January to June, and the 'Sales' column includes the corresponding sales figures in dollars. To visualize the data, we use the plot method of the DataFrame with kind='bar', specifying the 'Month' column for the x-axis and the 'Sales' column for the y-axis. The legend=False argument is used to hide the legend. We then customize the plot with the title, xlabel, and ylabel functions from Matplotlib to add a title and label the axes. Finally, we use plt.show() to display the plot. This exercise helps in understanding how to create a bar plot using Pandas and Matplotlib, which is a fundamental skill for data visualization in Python.

【Trivia】

- ▶ The plot method in Pandas is a wrapper around Matplotlib's plotting functions, making it easier to create basic plots from DataFrame data.
- ▶ Bar plots are particularly useful for comparing categorical data, such as sales figures for different months or regions.
- ▶ The plt.show() function is essential for rendering the plot when using Matplotlib, as it triggers the display of the current figure.

56. Creating an Area Plot Using Pandas

Importance★★★★☆

Difficulty★★★★☆

You are working as a data analyst for a retail company and need to visualize the monthly sales data of three different product categories (Electronics, Furniture, and Clothing) over a year.

Your goal is to create an area plot to show the sales trends for these categories.

Use Python and Pandas to generate the required data and plot the area chart.

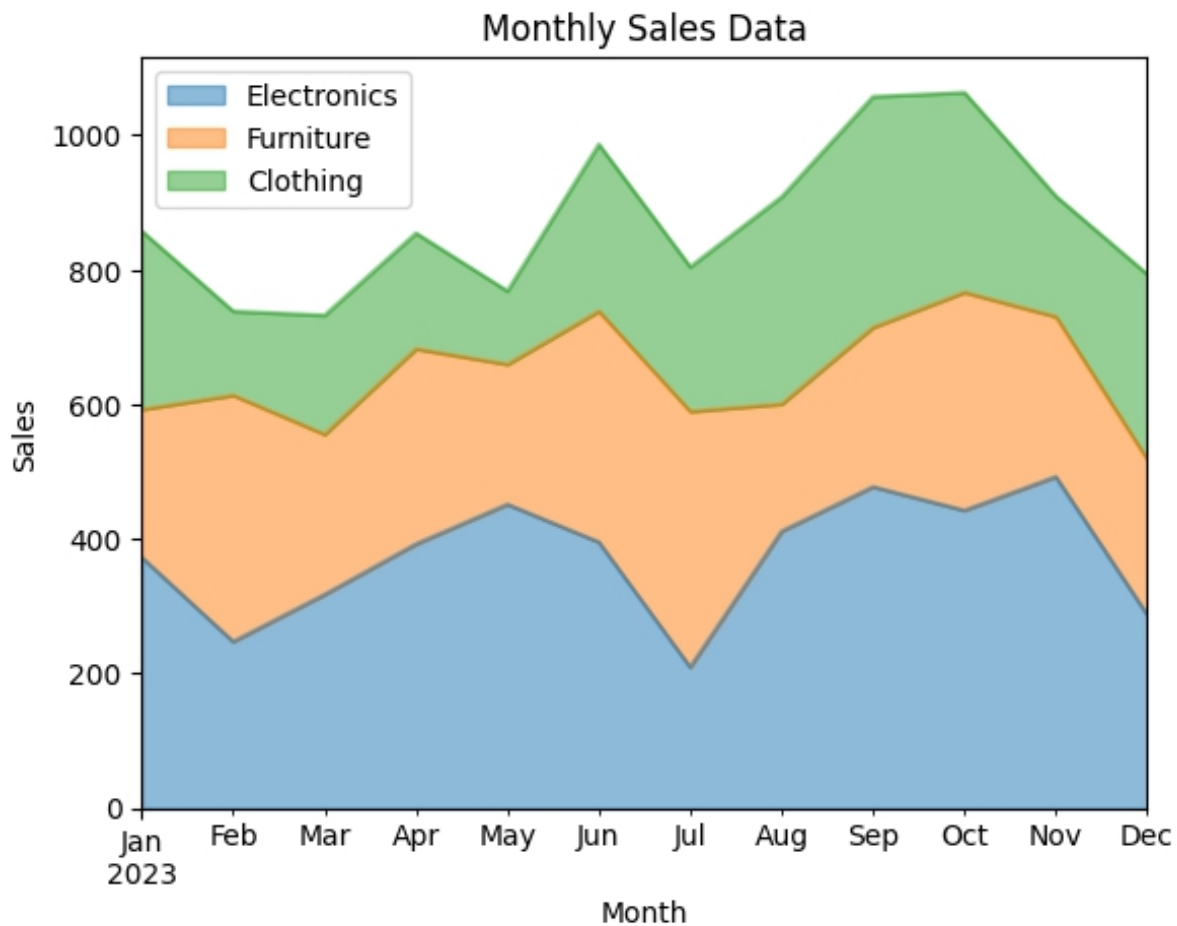
Ensure that the plot clearly shows the sales trends for each product category.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(0)
months = pd.date_range('2023-01-01', periods=12,
freq='M')
electronics_sales = np.random.randint(200, 500, size=12)
furniture_sales = np.random.randint(150, 400, size=12)
clothing_sales = np.random.randint(100, 350, size=12)
sales_data = pd.DataFrame({
'Month': months,
'Electronics': electronics_sales,
'Furniture': furniture_sales,
'Clothing': clothing_sales
})
```



【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
months = pd.date_range('2023-01-01', periods=12,
freq='M')
electronics_sales = np.random.randint(200, 500, size=12)
furniture_sales = np.random.randint(150, 400, size=12)
clothing_sales = np.random.randint(100, 350, size=12)
```



```
sales_data = pd.DataFrame({
    'Month': months,
    'Electronics': electronics_sales,
    'Furniture': furniture_sales,
    'Clothing': clothing_sales
})
sales_data.set_index('Month').plot(kind='area',
    stacked=True, alpha=0.5)
plt.title('Monthly Sales Data')
plt.xlabel('Month')
plt.ylabel('Sales')
plt.legend(loc='upper left')
plt.show()
```

This exercise requires you to create a monthly sales data visualization for three product categories using an area plot. First, you need to import the necessary libraries: pandas for data manipulation, numpy for generating random sales data, and matplotlib for plotting.

The code sets a random seed for reproducibility and generates a range of dates for the months in 2023. Random sales data for Electronics, Furniture, and Clothing are created using numpy's randint function.

These data are then combined into a pandas DataFrame with months as the index.

To create the area plot, the DataFrame is indexed by 'Month', and the plot function is used with the 'area' kind. The 'stacked' parameter ensures the plot is stacked, and 'alpha' sets the transparency.

Finally, titles and labels are added for clarity, and the plot is displayed using plt.show().

【Trivia】

- ▶ Area plots are useful for visualizing cumulative data over time, showing how different components contribute to the total.
- ▶ Transparency (alpha) in plotting helps to see overlapping areas more clearly, which is particularly useful in stacked area plots.
- ▶ Ensuring that legends are clearly labeled and placed helps in making the plot more readable and informative.
- ▶ The random seed in numpy ensures that the randomly generated numbers can be reproduced, which is important for debugging and consistency in data analysis.

57. Scatter Plot Creation with Pandas

Importance★★★★☆

Difficulty★★★★☆☆

You are a data analyst for a retail company and need to visualize the relationship between the advertising budget and the sales revenue.

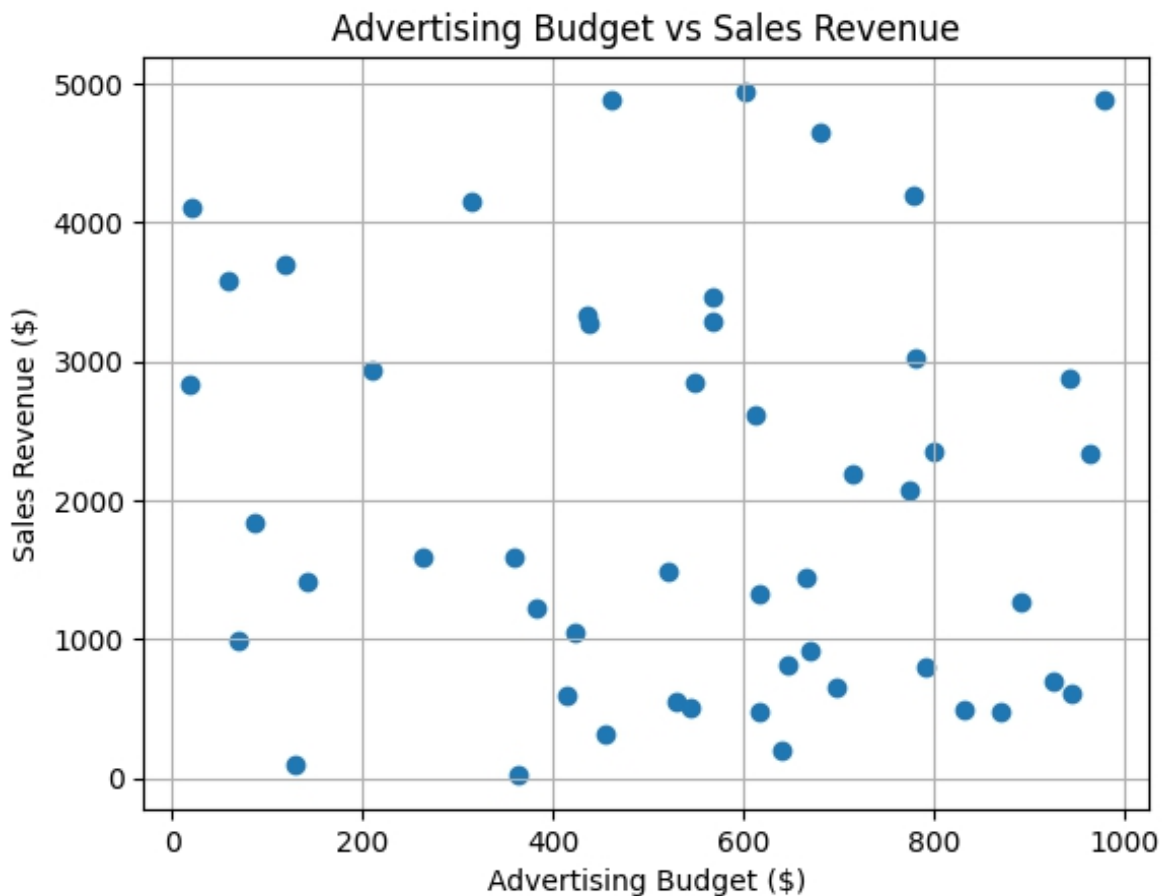
Create a scatter plot using Pandas to show this relationship. The dataset contains two columns: 'AdvertisingBudget' and 'SalesRevenue'.

Generate the data with random values for this exercise. Ensure that the scatter plot is clear and well-labeled.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(0)
data = pd.DataFrame({
    'AdvertisingBudget': np.random.rand(50) * 1000,
    'SalesRevenue': np.random.rand(50) * 5000
})
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
data = pd.DataFrame({
    'AdvertisingBudget': np.random.rand(50) * 1000,
    'SalesRevenue': np.random.rand(50) * 5000
})
plt.scatter(data['AdvertisingBudget'], data['SalesRevenue'])
plt.title('Advertising Budget vs Sales Revenue')
```

```
plt.xlabel('Advertising Budget ($)')
plt.ylabel('Sales Revenue ($)')
plt.grid(True)
plt.show()
```

First, we import the necessary libraries: pandas, numpy, and matplotlib.pyplot.

Pandas is used for data manipulation, numpy for generating random data, and matplotlib for plotting.

We set a random seed to ensure the generated data is reproducible.

We then create a DataFrame with two columns: 'AdvertisingBudget' and 'SalesRevenue'.

Each column is populated with 50 random values scaled appropriately (AdvertisingBudget between 0 and 1000 dollars, SalesRevenue between 0 and 5000 dollars).

To create the scatter plot, we use plt.scatter(), passing the 'AdvertisingBudget' as the x-axis and 'SalesRevenue' as the y-axis.

We add a title and labels for both axes to ensure the plot is understandable.

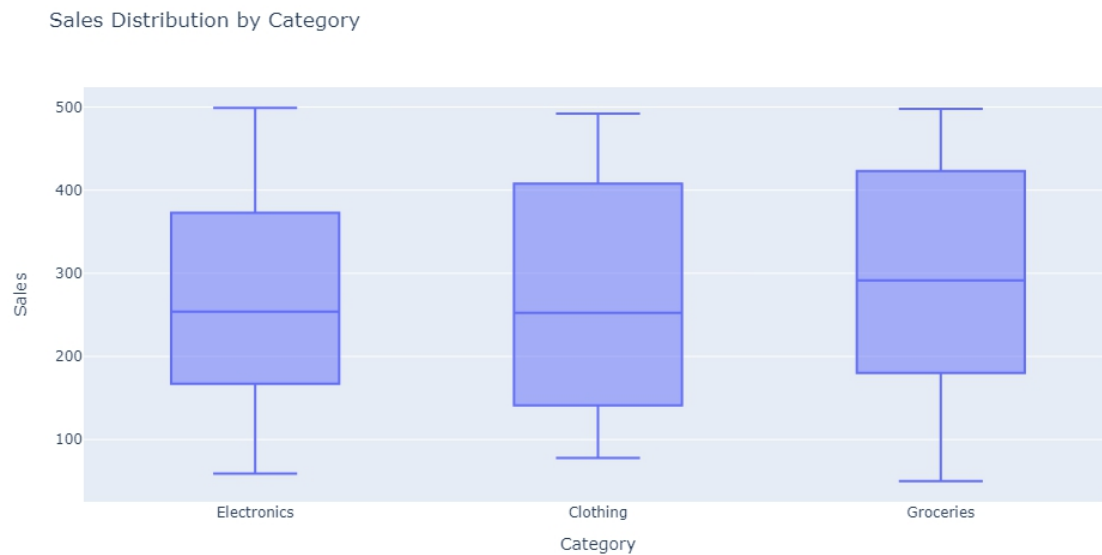
Finally, we enable the grid for better readability and display the plot using plt.show().

【Trivia】

- ▶ Scatter plots are useful for identifying potential correlations between two variables.
- ▶ In a scatter plot, each point represents an observation from the dataset, providing a visual representation of the data distribution.
- ▶ Adding a regression line to a scatter plot can help to visualize the trend and strength of the relationship between the variables.

- ▶ Matplotlib allows customization of scatter plots, such as changing point colors, sizes, and adding annotations, which can enhance data visualization.

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import plotly.express as px
np.random.seed(42)
categories = ['Electronics', 'Clothing', 'Home & Garden',
             'Books', 'Toys']
data = {'Category': np.random.choice(categories, 100),
        'Sales': np.random.lognormal(mean=3, sigma=1,
        size=100)}
df = pd.DataFrame(data)
fig = px.box(df, x='Category', y='Sales', title='Sales
Distribution by Category',
labels={'Category': 'Product Category', 'Sales': 'Sales
Amount'})
fig.show()
```

Box plots are used to display the distribution of a dataset based on a five-number summary: minimum, first quartile, median, third quartile, and maximum.

In this exercise, the data is generated using a log-normal distribution to simulate sales data across different product categories.

The code starts by importing necessary libraries: pandas for data manipulation, numpy for random data generation, and plotly for visualization.

A dataset is created with random categories and sales values using numpy's random.choice and random.lognormal functions.

Pandas is used to create a DataFrame from this data.

Plotly's express module (px) is used to generate a box plot.

The px.box function is called with the DataFrame, specifying the x-axis as 'Category' and y-axis as 'Sales'.

The title and labels parameters are used to customize the plot.

Finally, fig.show() displays the box plot.

【Trivia】

- ▶ Box plots were introduced by John Tukey in 1970 as a way to visually summarize data.
- ▶ The "box" in a box plot shows the interquartile range (IQR), which is the range between the first and third quartiles.
- ▶ The "whiskers" extend from the box to the smallest and largest values within 1.5 times the IQR from the quartiles.
- ▶ Outliers are plotted as individual points beyond the whiskers.
- ▶ Plotly is a powerful graphing library that supports interactive plots, making it useful for exploring data visually.

59. Creating a Violin Plot Using Plotly

Importance★★★★☆

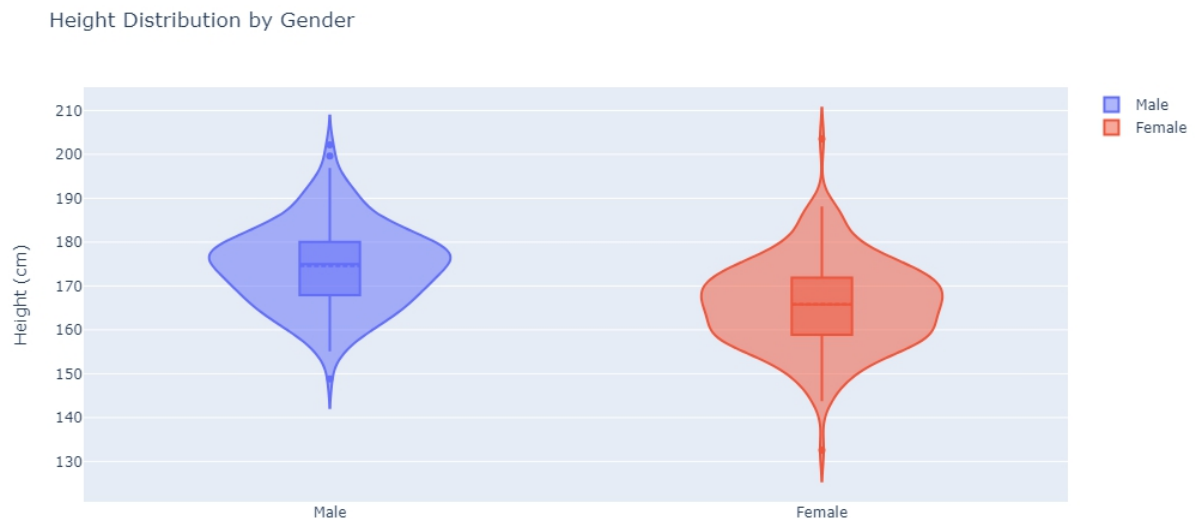
Difficulty★★★★☆☆

A client is interested in visualizing the distribution of two different datasets to compare their variability and central tendency. They have provided two sets of data: one representing heights of males and another representing heights of females. Your task is to create a violin plot to visualize these distributions using Plotly. This visualization will help the client understand the differences and similarities between the two datasets. Please use the following code to generate sample data and then create the violin plot.

【Data Generation Code Example】

```
import numpy as np
np.random.seed(42)
heights_male = np.random.normal(175, 10, 200)
heights_female = np.random.normal(165, 10, 200)
```

【Diagram Answer】



【Code Answer】

```
import plotly.graph_objects as go
import numpy as np
np.random.seed(42)
heights_male = np.random.normal(175, 10, 200)
heights_female = np.random.normal(165, 10, 200)
fig = go.Figure()
fig.add_trace(go.Violin(y=heights_male, name='Male',
box_visible=True, meanline_visible=True))
fig.add_trace(go.Violin(y=heights_female, name='Female',
box_visible=True, meanline_visible=True))
fig.update_layout(title='Height Distribution by Gender',
yaxis_title='Height (cm)')
fig.show()
```

To create a violin plot using Plotly, you first need to import the necessary libraries. In this case, `plotly.graph_objects` is used for creating and displaying the plot.

The sample data is generated using NumPy's `random.normal` function, which creates arrays of normally distributed data. The `np.random.seed(42)` ensures that the data is reproducible. Two datasets are generated: one for males with a mean height of 175 cm and a standard deviation of 10 cm, and another for females with a mean height of 165 cm and a standard deviation of 10 cm. Each dataset contains 200 data points.

Next, you create a Figure object using `go.Figure()`. To this figure, you add two violin traces using `fig.add_trace(go.Violin())`. Each trace represents one of the datasets (male and female heights). The parameters `box_visible=True` and `meanline_visible=True` ensure that the box plot and mean line are visible within the violin plot. Finally, the layout of the plot is updated with a title and y-axis label using `fig.update_layout()`, and the plot is displayed with `fig.show()`.

Violin plots are useful for visualizing the distribution of data across different categories. They combine aspects of box plots and density plots, showing the probability density of the data at different values, along with summary statistics like the median and interquartile range. This makes them a powerful tool for comparing distributions.

【Trivia】

- ▶ Violin plots are particularly useful when comparing multiple distributions, as they can show differences in distribution shape, central tendency, and variability.
- ▶ The combination of box plot and density plot features in a violin plot provides a more comprehensive understanding of the data compared to using either method alone.
- ▶ Plotly is a versatile library for creating interactive plots, making it easier to explore data visually and gain insights

through interactive features like zooming and hovering.

60. Creating Interactive Line Plots with Plotly

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst working for a fitness tracking company.

The company has collected daily step count data for 100 users over a 30-day period.

Your task is to create an interactive line plot using Plotly to visualize the average daily step count across all users.

The plot should show the trend of average steps over time, with the ability to hover over data points for more information.

Your objectives are:

Generate sample data for 100 users over 30 days.

Calculate the average daily step count across all users.

Create an interactive line plot using Plotly.

Include appropriate labels, title, and hover information.

Please write a Python script that accomplishes these tasks.

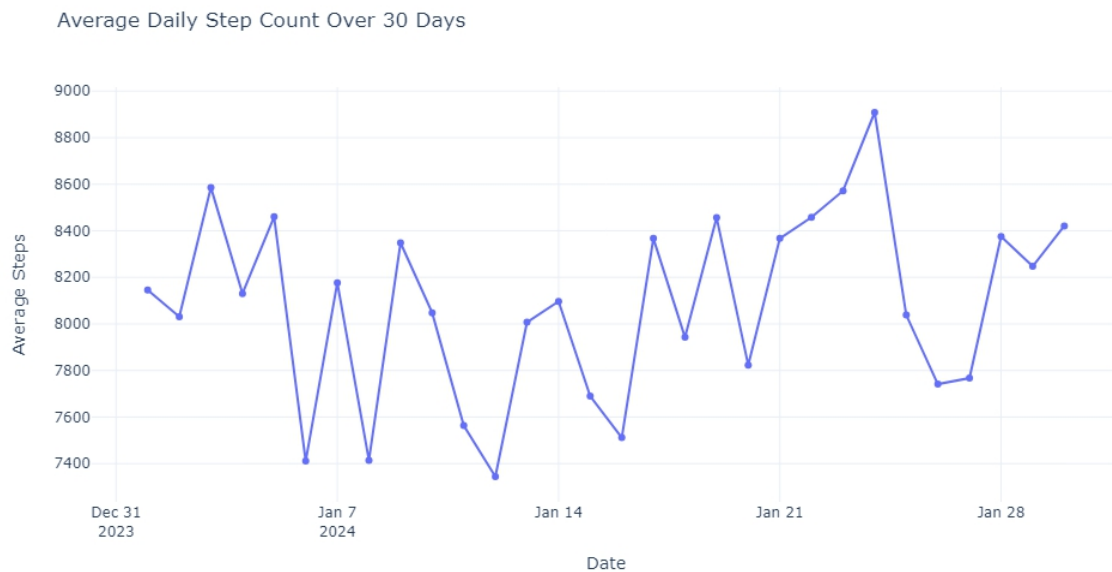
Make sure to include code for data generation within your script.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
np.random.seed(42)
date_range = pd.date_range(start='2024-01-01',
periods=30)
users = range(1, 101)
data = np.random.randint(1000, 15000, size=(len(users),
len(date_range)))
```

```
df = pd.DataFrame(data, columns=date_range,  
index=users)  
df_melted = df.reset_index().melt(id_vars='index',  
var_name='Date', value_name='Steps')  
df_melted = df_melted.rename(columns={'index': 'User'})
```

【Diagram Answer】



【Code Answer】

```
import numpy as np
import pandas as pd
import plotly.graph_objects as go
## Generate sample data
np.random.seed(42)
date_range = pd.date_range(start='2024-01-01',
periods=30)
users = range(1, 101)
data = np.random.randint(1000, 15000, size=(len(users),
len(date_range)))
df = pd.DataFrame(data, columns=date_range,
index=users)
## Calculate average daily step count
avg_steps = df.mean()
## Create interactive line plot
```



```

fig = go.Figure()
fig.add_trace(go.Scatter(x=avg_steps.index,
y=avg_steps.values,
mode='lines+markers',
name='Average Steps',
hovertemplate='Date: %{x}Average Steps: %{y:.0f}
<extra></extra>'))
## Customize layout
fig.update_layout(
title='Average Daily Step Count Over 30 Days',
xaxis_title='Date',
yaxis_title='Average Steps',
hovermode='x unified',
template='plotly_white'
)
## Show the plot
fig.show()

```

This Python script creates an interactive line plot using Plotly to visualize the average daily step count across 100 users over a 30-day period.

Let's break down the code and explain each part in detail:

Data Generation:

We use NumPy and Pandas to generate sample data.

A random seed is set for reproducibility.

We create a date range for 30 days starting from January 1, 2024.

We generate random step counts between 1000 and 15000 for 100 users over 30 days.

The data is stored in a Pandas DataFrame.

Data Processing:

We calculate the average daily step count across all users using the `mean()` function.

This gives us a series of average steps for each day.

Creating the Interactive Line Plot:

We import Plotly's `graph_objects` module to create the plot.

We initialize a Figure object using `go.Figure()`.

We add a trace to the figure using `fig.add_trace()`.

The trace is a Scatter plot with mode set to `'lines+markers'`, which creates a line with markers at each data point.

We set the x-axis to the dates and the y-axis to the average step counts.

The `hovertemplate` is customized to show the date and average steps when hovering over a point.

Customizing the Layout:

We use `fig.update_layout()` to customize the appearance of the plot.

We set a title for the plot, labels for the x and y axes, and choose a white template for a clean look.

The `hovermode` is set to `'x unified'`, which shows hover information for all traces at the same x-coordinate.

Displaying the Plot:

Finally, we call `fig.show()` to display the interactive plot.

This script demonstrates several key concepts in data visualization with Python:

Data generation and manipulation using NumPy and Pandas

Calculating summary statistics (average steps)

Creating interactive plots with Plotly

Customizing plot appearance and hover information

The resulting plot allows users to interact with the data, seeing the trend of average steps over time and getting specific values by hovering over data points.

This type of visualization is particularly useful for analyzing trends and patterns in time-series data.

【Trivia】

- ▶ Plotly is an open-source graphing library that allows you to create interactive, publication-quality graphs in Python.
- ▶ The 'plotly_white' template used in this example is one of several built-in themes in Plotly. Other options include 'plotly', 'plotly_dark', 'ggplot2', 'seaborn', and more.
- ▶ The average adult takes between 4,000 to 18,000 steps per day. The wide range in our generated data (1,000 to 15,000) reflects the variability in real-world step counts.
- ▶ Interactive plots are particularly useful for data exploration as they allow users to zoom, pan, and hover over data points for more information.
- ▶ The use of a random seed (`np.random.seed(42)`) ensures that the same "random" data is generated each time the script is run, which is crucial for reproducibility in data analysis and scientific computing.
- ▶ Plotly can be used to create a wide variety of chart types beyond line plots, including bar charts, scatter plots, heatmaps, 3D plots, and more.
- ▶ The hover template in Plotly uses a special syntax where `%{x}` and `%{y}` refer to the x and y values of the data point, respectively. This allows for dynamic updating of hover information.
- ▶ While Matplotlib is often considered the standard plotting library in Python, Plotly has gained popularity due to its interactive features and ability to create web-ready visualizations.

61. Bar Plot Visualization with Plotly

Importance★★★★☆

Difficulty★★★☆☆

A customer wants to visualize their sales data for different product categories over several months to identify trends and patterns.

You need to create a bar plot using Plotly to display this information.

Generate a dataset with random sales numbers for three product categories ("Electronics," "Clothing," "Groceries") over six months.

The dataset should have columns: "Month," "Category," and "Sales."

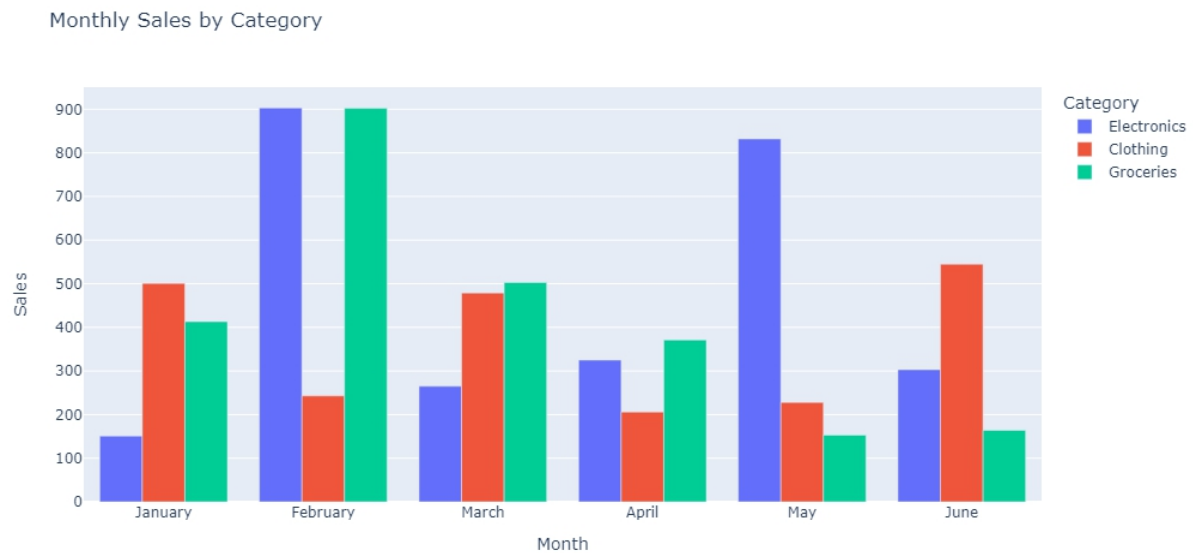
Write the Python code to generate this dataset and create a bar plot using Plotly.

Ensure the plot has appropriate labels and a title.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
months = ["January", "February", "March", "April", "May",
"June"]
categories = ["Electronics", "Clothing", "Groceries"]
data = {"Month": [month for month in months for _ in
categories],
"Category": [category for _ in months for category in
categories],
"Sales": np.random.randint(100, 1000, size=len(months) *
len(categories))}
df = pd.DataFrame(data)
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import plotly.express as px
months = ["January", "February", "March", "April", "May",
"June"]
categories = ["Electronics", "Clothing", "Groceries"]
data = {"Month": [month for month in months for _ in
categories],
"Category": [category for _ in months for category in
categories],
"Sales": np.random.randint(100, 1000, size=len(months) *
len(categories))}
df = pd.DataFrame(data)
fig = px.bar(df, x="Month", y="Sales", color="Category",
barmode="group", title="Monthly Sales by Category")
```

```
fig.show()
```

The code begins by importing necessary libraries: pandas for data manipulation, numpy for generating random numbers, and plotly.express for creating the plot.

A list of months and categories is created to represent the sales data.

A dictionary named 'data' is constructed where the "Month" key has a list of months repeated for each category, the "Category" key has each category repeated for the number of months,

and the "Sales" key has random sales numbers generated using numpy's randint function.

This dictionary is converted into a pandas DataFrame named 'df'.

The plotly.express bar function (px.bar) is used to create a bar plot.

The x-axis is set to "Month", the y-axis to "Sales", and different colors are used for each "Category".

The barmode is set to "group" to place bars for different categories side by side, and a title is added to the plot.

Finally, fig.show() displays the plot.

【Trivia】

Plotly is a versatile library that allows for interactive plotting in Python.

It can be used to create a variety of plots including line plots, scatter plots, and bar plots.

Plotly's express module is a high-level interface which makes it easy and quick to create plots with just a few lines of code.

Interactive plots generated by Plotly can be easily embedded in web applications and shared.

62. Creating a Pie Chart with Plotly

Importance★★★★☆

Difficulty★★★★☆☆

A retail company wants to visualize the sales distribution across different product categories to understand their market better.

They need a pie chart that shows the percentage of sales for each category.

The data includes the categories 'Electronics', 'Furniture', 'Clothing', and 'Books', with respective sales figures of 15000, 8000, 6000, and 3000.

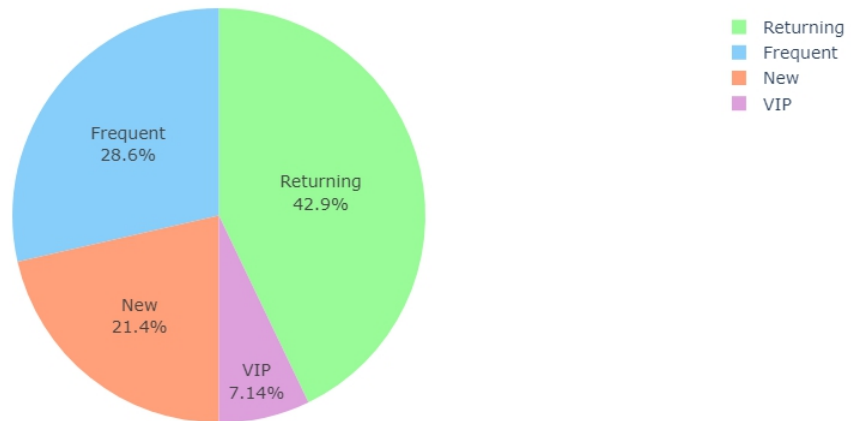
Create a Python script to generate this pie chart using Plotly.

【Data Generation Code Example】

```
sales_data=dict()
sales_data['Categories']=
['Electronics','Furniture','Clothing','Books']
sales_data['Sales']=[15000,8000,6000,3000]
```

【Diagram Answer】

Customer Segmentation by Purchase Behavior



【Code Answer】

```
import plotly.express as px
sales_data=dict()
sales_data['Categories']=
['Electronics','Furniture','Clothing','Books']
sales_data['Sales']=[15000,8000,6000,3000]
fig=px.pie(values=sales_data['Sales'],names=sales_data['C
ategories'],title='Sales Distribution by Category')
fig.show()
```

To create a pie chart using Plotly, you first need to import the Plotly Express module.

Then, you define the sales data in a dictionary with keys 'Categories' and 'Sales'.

The 'Categories' key holds a list of product categories, and the 'Sales' key holds the corresponding sales figures.

The `px.pie` function from Plotly Express is used to create the pie chart.

You pass the sales figures to the values parameter and the categories to the names parameter.

Additionally, you can set the title of the pie chart using the title parameter.

Finally, the show method is called on the figure object to display the chart.

【Trivia】

Pie charts are useful for showing the relative proportions of different categories within a dataset.

However, they can become difficult to interpret if there are too many categories or if the differences between them are very small.

In such cases, other types of visualizations like bar charts or stacked bar charts might be more effective.

Plotly is a powerful graphing library that makes interactive plots easy to create and customize, which can be particularly useful for data presentations and dashboards.

63. Creating a Treemap with Plotly

Importance★★★☆☆

Difficulty★★☆☆☆

A company wants to visualize their sales data to understand the distribution across different product categories and regions.

Your task is to create a treemap using Plotly that displays this information.

The data should include the following fields: 'Region', 'Category', and 'Sales'.

Use the generated data to create a treemap where the size of each rectangle represents the sales volume.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
np.random.seed(0)
regions = ['North', 'South', 'East', 'West']
categories = ['Electronics', 'Clothing', 'Furniture']
data = pd.DataFrame({
    'Region': np.random.choice(regions, 100),
    'Category': np.random.choice(categories, 100),
    'Sales': np.random.randint(1000, 10000, 100)
})
```

【Diagram Answer】

Sales Distribution by Region and Category



【Code Answer】

```
import pandas as pd
import numpy as np
import plotly.express as px
np.random.seed(0)
regions = ['North', 'South', 'East', 'West']
categories = ['Electronics', 'Clothing', 'Furniture']
data = pd.DataFrame({
    'Region': np.random.choice(regions, 100),
    'Category': np.random.choice(categories, 100),
    'Sales': np.random.randint(1000, 10000, 100)
})
fig = px.treemap(data, path=['Region', 'Category'],
values='Sales', title='Sales Distribution by Region and
Category')
fig.show()
```

First, the necessary libraries are imported, including pandas for data manipulation, numpy for random data generation, and plotly.express for creating the treemap.

The random seed is set to ensure reproducibility.

Arrays for regions and categories are defined, and a DataFrame is created with 100 random entries for regions and categories, and random sales values between 1000 and 10000.

Using plotly.express's treemap function, the data is visualized, with 'Region' and 'Category' defining the hierarchy and 'Sales' determining the size of each rectangle. The resulting plot provides a clear, visual representation of sales distribution across different regions and product categories.

【Trivia】

- ▶ Treemaps are particularly useful for visualizing hierarchical data, where the size of each rectangle reflects a quantitative dimension.
- ▶ Plotly is a powerful library for interactive data visualizations, making it easier to explore complex datasets.
- ▶ The choice of using random data helps in learning the process without needing actual data, and the skills can be directly applied to real-world data scenarios.

64. Plotting a Funnel Chart Using Plotly

Importance★★★★☆

Difficulty★★★★☆

A digital marketing agency wants to visualize the conversion rates of their sales funnel.

They have the following stages: 'Website Visits', 'Sign-Ups', 'Free Trials', and 'Purchases'.

Each stage has the following number of users: 5000, 1500, 800, and 200 respectively.

Create a funnel chart to help the agency understand their conversion rates at each stage.

Use Python and Plotly to generate the funnel chart.

Below is the code to create the necessary data for this visualization.

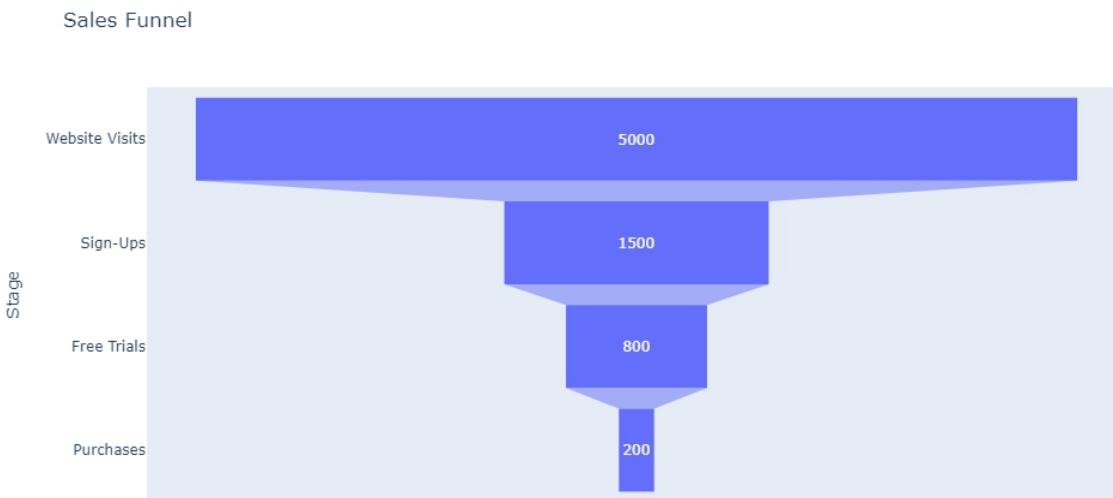
Complete the code to generate the funnel chart.

【Data Generation Code Example】

```
import pandas as pd
import plotly.express as px
data = {
    'Stage': ['Website Visits', 'Sign-Ups', 'Free Trials',
             'Purchases'],
    'Users': [5000, 1500, 800, 200]
}
df = pd.DataFrame(data)
```

Complete the code to plot the funnel chart using Plotly

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import plotly.express as px
data = {
    'Stage': ['Website Visits', 'Sign-Ups', 'Free Trials',
             'Purchases'],
    'Users': [5000, 1500, 800, 200]
}
df = pd.DataFrame(data)
fig = px.funnel(df, x='Users', y='Stage', title='Sales Funnel')
fig.show()
```

To create a funnel chart using Plotly, we first import the necessary libraries, pandas and plotly.express. Pandas is used to handle the data in a structured format, and Plotly Express is a high-level interface for Plotly that makes it easy to create various types of plots.

We then define a dictionary to store the stages of the sales funnel and the number of users at each stage.

This dictionary is converted into a pandas DataFrame for easier manipulation and plotting.

The core part of the code involves using the `px.funnel` function from Plotly Express.

This function requires the DataFrame, the x-axis value (number of users), and the y-axis value (stages of the funnel).

We also add a title to the chart using the `title` parameter.

Finally, the `fig.show()` command displays the funnel chart.

This exercise demonstrates basic data handling with pandas and visualization with Plotly, focusing on creating a funnel chart to represent conversion rates at different stages of a sales process.

【Trivia】

- ▶ Funnel charts are often used in sales and marketing to visualize the stages of a process and identify potential areas where drop-offs occur.
- ▶ Plotly is a powerful graphing library that supports interactive and dynamic visualizations, making it ideal for data analysis and presentation.
- ▶ In addition to funnel charts, Plotly Express can create a wide variety of plots, including scatter plots, line charts, bar charts, and more.

65. Creating a Waterfall Chart with Plotly for Financial Analysis

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst at a growing e-commerce company. The CEO has asked you to create a waterfall chart to visualize the company's financial performance over the past year.

The chart should show how various income and expense categories contribute to the overall change in the company's financial position.

Your task is to:

Create sample data representing different financial categories and their respective values.

Use Plotly to create a waterfall chart that clearly shows how each category impacts the company's bottom line.

Ensure the chart is visually appealing and easy to understand, with appropriate labels and colors.

Add a title and axis labels to the chart.

The waterfall chart should include the following categories:

Starting Balance

Revenue

Cost of Goods Sold

Operating Expenses

Taxes

Investments

Final Balance

Your code should generate the sample data and create the waterfall chart in a single, executable script.

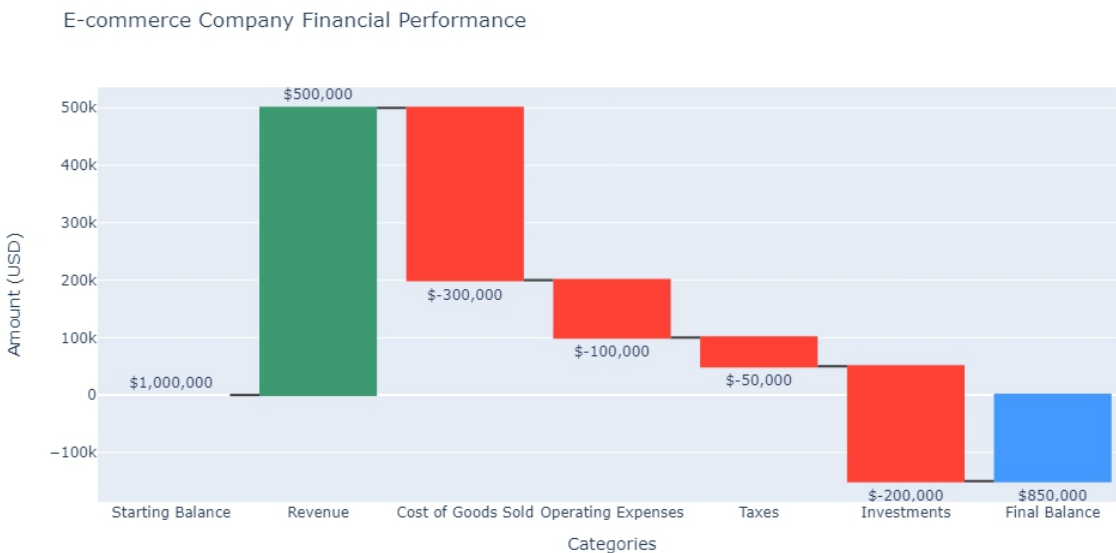
【Data Generation Code Example】

```
import pandas as pd
```



```
data = pd.DataFrame([
    ("Starting Balance", 1000000),
    ("Revenue", 500000),
    ("Cost of Goods Sold", -300000),
    ("Operating Expenses", -100000),
    ("Taxes", -50000),
    ("Investments", -200000),
    ("Final Balance", 850000)
], columns=["Category", "Amount"])
data["Type"] = ["total", "relative", "relative", "relative",
               "relative", "relative", "total"]
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import plotly.graph_objects as go
data = pd.DataFrame([
    ("Starting Balance", 1000000),
    ("Revenue", 500000),
    ("Cost of Goods Sold", -300000),
    ("Operating Expenses", -100000),
    ("Taxes", -50000),
    ("Investments", -200000),
    ("Final Balance", 850000)
], columns=["Category", "Amount"])
data["Type"] = ["total", "relative", "relative", "relative",
                "relative", "relative", "total"]
fig = go.Figure(go.Waterfall(
    name = "Financial Performance",
```

```

orientation = "v",
measure = data["Type"],
x = data["Category"],
textposition = "outside",
text = data["Amount"].apply(lambda x: f"${x:,.0f}"),
y = data["Amount"],
connector = {"line":{"color":"rgb(63, 63, 63)"}},
))
fig.update_layout(
title = "E-commerce Company Financial Performance",
showlegend = False,
xaxis_title = "Categories",
yaxis_title = "Amount (USD)"
)
fig.show()

```

This code creates a waterfall chart using Plotly to visualize the financial performance of an e-commerce company.

Let's break down the code and explain its key components:

Data Preparation:

We use pandas to create a DataFrame with financial categories and their corresponding amounts.

The 'Type' column is added to specify whether each category is a 'total' or 'relative' value.

Plotly Waterfall Chart Creation:

We import the necessary Plotly module: `plotly.graph_objects` as `go`.

A Figure object is created using `go.Figure()`.

Inside the Figure, we create a Waterfall chart using `go.Waterfall()`.

Waterfall Chart Configuration:

name: Sets the name of the chart (used in legends).

orientation: "v" for vertical orientation.

measure: Specifies whether each bar is a total or relative value.

x: Sets the x-axis labels (categories).

textposition: Places the text labels outside the bars.

text: Formats the amount values as currency strings.

y: Sets the y-axis values (amounts).

connector: Configures the lines connecting the bars.

Data Formatting:

We use a lambda function to format the amount values as currency strings: `lambda x: f"${x:,.0f}"`.

This adds a dollar sign, comma separators, and removes decimal places.

Chart Layout:

`update_layout()` is used to customize the chart's appearance.

We set a title, hide the legend, and add axis titles.

Displaying the Chart:

`fig.show()` renders and displays the chart.

This code demonstrates several important aspects of data visualization with Python:

Data manipulation with pandas

Creating interactive charts with Plotly

Customizing chart appearance and labels

Formatting numerical data for better readability

The resulting waterfall chart effectively shows how each financial category contributes to the company's overall financial position, making it easy to identify the impact of revenues, expenses, and investments.

【Trivia】

- ▶ Waterfall charts are also known as bridge charts or cascade charts.
- ▶ They are particularly useful for understanding how an initial value is affected by a series of intermediate positive or negative values.
- ▶ Waterfall charts were popularized by McKinsey & Company in the 1960s for visualizing financial statements.
- ▶ While Plotly is used here, other Python libraries like Matplotlib and Seaborn can also create waterfall charts, though with more complex code.
- ▶ Waterfall charts are commonly used in financial analysis, project management, and inventory analysis.
- ▶ The color coding in waterfall charts typically uses green for positive values, red for negative values, and a neutral color like blue or gray for totals.
- ▶ Interactive features of Plotly charts, such as hover information and zooming, make them particularly useful for detailed data exploration.

66. Generate a Candlestick Chart Using Plotly

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst working for a financial firm. Your task is to generate a candlestick chart to visualize stock price movements over a period of time. The chart should include open, high, low, and close prices for each day. Use the Plotly library in Python to create this visualization.

Create a sample dataset within your code that includes the following columns: 'Date', 'Open', 'High', 'Low', 'Close'. The dataset should cover a period of 10 days. Ensure that the dates are sequential and the prices are realistic.

Write the Python code to generate and display the candlestick chart using Plotly.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
dates = pd.date_range(start='2024-07-01', periods=10)
open_prices = np.random.uniform(low=100, high=200, size=10)
high_prices = open_prices + np.random.uniform(low=1, high=10, size=10)
low_prices = open_prices - np.random.uniform(low=1, high=10, size=10)
close_prices = open_prices + np.random.uniform(low=-5, high=5, size=10)
data = pd.DataFrame({'Date': dates, 'Open': open_prices, 'High': high_prices, 'Low': low_prices, 'Close': close_prices})
```



【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
dates = pd.date_range(start='2024-07-01', periods=10)
open_prices = np.random.uniform(low=100, high=200,
size=10)
high_prices = open_prices + np.random.uniform(low=1,
high=10, size=10)
low_prices = open_prices - np.random.uniform(low=1,
high=10, size=10)
close_prices = open_prices + np.random.uniform(low=-5,
high=5, size=10)
data = pd.DataFrame({'Date': dates, 'Open': open_prices,
'High': high_prices, 'Low': low_prices, 'Close': close_prices})
```



```
fig = go.Figure(data=[go.Candlestick(x=data['Date'],
open=data['Open'], high=data['High'], low=data['Low'],
close=data['Close'])])
fig.update_layout(title='Stock Price Movements',
xaxis_title='Date', yaxis_title='Price')
fig.show()
```

To generate a candlestick chart using Plotly, you first need to create a dataset that includes the necessary columns: 'Date', 'Open', 'High', 'Low', and 'Close'. In this example, the dataset covers a period of 10 days. The dates are generated using `pd.date_range`, and the prices are generated using `np.random.uniform` to ensure they are realistic.

Once the dataset is created, you can use Plotly's `go.Figure` and `go.Candlestick` to create the candlestick chart. The `go.Candlestick` function takes the 'Date', 'Open', 'High', 'Low', and 'Close' columns as inputs. The `fig.update_layout` method is used to add titles to the chart and the axes. Finally, the `fig.show()` method displays the chart.

This exercise helps you practice data manipulation and visualization using Python, focusing on financial data. It demonstrates how to generate sample data and visualize it effectively using Plotly.

【Trivia】

- ▶ Candlestick charts originated in Japan and were used by rice traders in the 18th century to track market prices and daily momentum.
- ▶ Plotly is an open-source graphing library that makes interactive, publication-quality graphs online. It is particularly useful for creating complex visualizations like candlestick charts.
- ▶ The 'Open', 'High', 'Low', and 'Close' prices are essential components of financial data analysis, providing insights

into market trends and price movements.

67. Creating a Heatmap with Plotly

Importance★★★★☆

Difficulty★★★☆☆

You are working as a data analyst for a retail company. Your manager has asked you to analyze the sales data and create a heatmap to visualize the sales performance across different stores and product categories.

The heatmap should help identify patterns and trends in the sales data.

Use Plotly to create the heatmap. Create a heatmap that visualizes the sales performance of different products across various stores.

The data should include store names, product categories, and sales figures.

Use the following sample data: ▶ Stores: ['Store A', 'Store B', 'Store C', 'Store D']

▶ Product Categories: ['Electronics', 'Clothing', 'Groceries', 'Furniture']

▶ Sales Figures: Random integers between 1000 and 5000

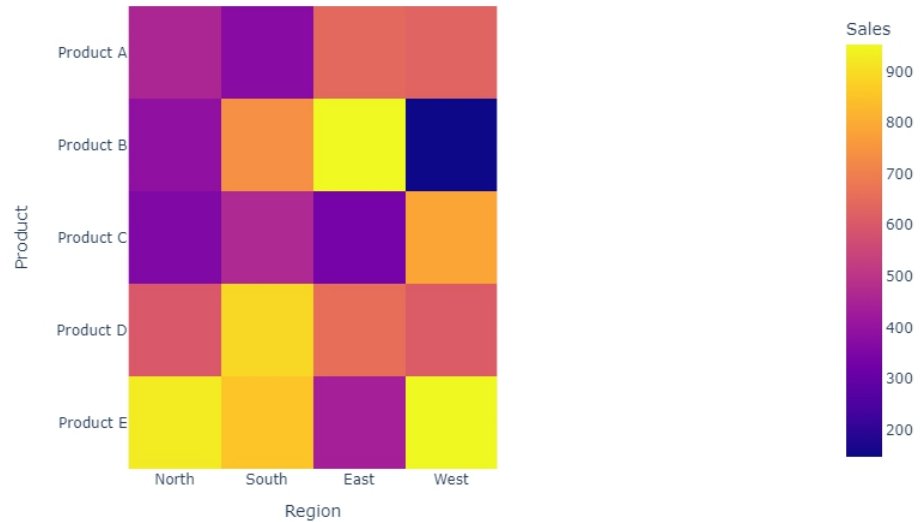
【Data Generation Code Example】

```
import pandas as pd
import numpy as np
## Sample data creation
stores = ['Store A', 'Store B', 'Store C', 'Store D']
categories = ['Electronics', 'Clothing', 'Groceries', 'Furniture']
sales_data = np.random.randint(1000, 5000, size=
(len(stores), len(categories)))
## Creating DataFrame
df = pd.DataFrame(sales_data, index=stores,
columns=categories)
```

df

【Diagram Answer】

Sales Performance Heatmap



【Code Answer】

```
import pandas as pd
import numpy as np
import plotly.express as px
## Sample data creation
stores = ['Store A', 'Store B', 'Store C', 'Store D']
categories = ['Electronics', 'Clothing', 'Groceries', 'Furniture']
sales_data = np.random.randint(1000, 5000, size=(len(stores), len(categories)))
## Creating DataFrame
df = pd.DataFrame(sales_data, index=stores, columns=categories)
## Creating heatmap using Plotly
fig = px.imshow(df, labels=dict(x="Product Categories", y="Stores", color="Sales Figures"), x=categories, y=stores)
fig.show()
```

To create the heatmap, we start by importing the necessary libraries: pandas, numpy, and plotly.express.

We define the sample data, including store names, product categories, and random sales figures.

This data is then structured into a DataFrame using pandas. The px.imshow function from Plotly is used to generate the heatmap.

We pass the DataFrame and label the axes and color scale appropriately.

The heatmap is displayed using fig.show(), allowing us to visually analyze sales performance across stores and product categories.

This exercise demonstrates essential data manipulation and visualization skills, focusing on creating a clear and informative heatmap.

The heatmap provides an intuitive way to identify patterns and trends in the sales data, facilitating better decision-making.

【Trivia】

Heatmaps are a powerful tool for visualizing complex data relationships in a matrix format.

They are widely used in various fields, including biology (e.g., gene expression data), finance (e.g., correlation matrices), and marketing (e.g., sales performance).

Plotly, a versatile visualization library, offers interactive heatmaps, enhancing data exploration and presentation capabilities.

68. Plotting a Contour Plot with Plotly

Importance★★★★☆

Difficulty★★★★☆☆

You are a data analyst working for a geological survey company.

Your task is to visualize the topography of a region using a contour plot.

You are provided with X and Y coordinates representing the geographical area and a function that generates the elevation data (Z values).

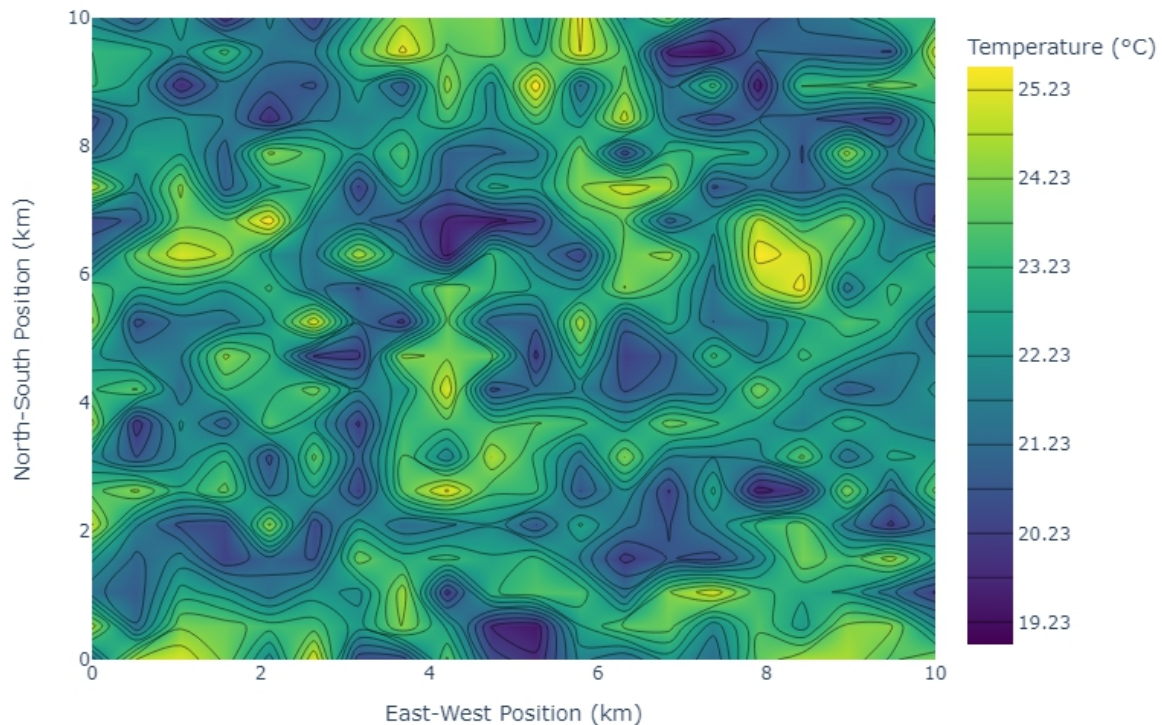
Create a contour plot using Plotly to help your team understand the topographical variations of the region.

【Data Generation Code Example】

```
import numpy as np
# Create data
X = np.linspace(-5, 5, 100)
Y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(X, Y)
Z = np.sin(np.sqrt(X**2 + Y**2))
# Z values represent elevation data
X = X.flatten()
Y = Y.flatten()
Z = Z.flatten()
```

【Diagram Answer】

Temperature Distribution Across the City



【Code Answer】

```
import numpy as np
import plotly.graph_objects as go
# Create data
X = np.linspace(-5, 5, 100)
Y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(X, Y)
Z = np.sin(np.sqrt(X**2 + Y**2))
# Create a contour plot
fig = go.Figure(data=go.Contour(
    x=X[0], # Flattening is not needed
```



```
y=Y[:,0], # Flattening is not needed
z=Z
))
# Show the plot
fig.show()
```

To create a contour plot with Plotly, you first need to prepare your data.

The `numpy.linspace` function generates linearly spaced values, which are used to create a meshgrid representing the X and Y coordinates of the region.

Using `numpy.meshgrid`, these coordinates are combined to form a grid.

The Z values, representing elevation data, are generated using a mathematical function that calculates the elevation at each point in the grid.

In this example, the function `np.sin(np.sqrt(X**2 + Y**2))` is used to create a radial elevation pattern. The `go.Figure` function from Plotly is then used to create a contour plot.

The `go.Contour` method takes the X, Y, and Z values and generates the contour plot.

The X and Y values are not flattened, as Plotly's `go.Contour` can handle 2D arrays directly.

The `fig.show()` method displays the plot, allowing the team to visualize the topography of the region.

【Trivia】

Contour plots are widely used in various fields, including geography, meteorology, and engineering, to represent three-dimensional data in two dimensions.

They help in understanding the elevation, temperature, pressure, and other continuous data variations across a surface.

Plotly's interactive features make it an excellent choice for exploring such data visually.

69. Creating 3D Scatter Plots with Plotly

Importance★★★★☆

Difficulty★★★☆☆

You are a data analyst working for a multinational e-commerce company.

The company wants to visualize the relationship between customer age, annual spending, and the number of purchases made in the last year.

Your task is to create a 3D scatter plot using Plotly to help the marketing team understand these relationships better.

Create a Python script that does the following:

Generate a dataset of 100 customers with the following information:

Age (between 18 and 70)

Annual spending (between \$100 and \$10,000)

Number of purchases (between 1 and 50)

Create a 3D scatter plot using Plotly with the following specifications:

X-axis: Age

Y-axis: Annual spending

Z-axis: Number of purchases

Each point should represent a customer

Color the points based on the annual spending (use a color scale from blue to red)

Add hover text to display the exact values for each customer

Customize the plot with appropriate titles, labels, and a color bar.

Display the plot in your default web colab.

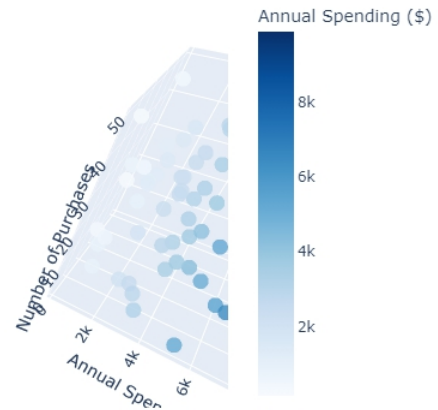
Your code should generate the data and create the plot in a single script, without reading from or writing to external files.

【Data Generation Code Example】

```
import numpy as np
np.random.seed(42)
age = np.random.randint(18, 71, 100)
spending = np.random.uniform(100, 10000, 100)
purchases = np.random.randint(1, 51, 100)
```

【Diagram Answer】

Customer Age, Spending, and Purchase Frequency



【Code Answer】

```
import numpy as np
import plotly.graph_objects as go
np.random.seed(42)
age = np.random.randint(18, 71, 100)
spending = np.random.uniform(100, 10000, 100)
purchases = np.random.randint(1, 51, 100)
fig = go.Figure(data=[go.Scatter3d(
    x=age,
    y=spending,
    z=purchases,
    mode='markers',
    marker=dict(
        size=5,
        color=spending,
        colorscale='Blues',
```

```

opacity=0.8,
colorbar=dict(title='Annual Spending ($)')
),
text=[f'Age: {a}Spending: ${s:.2f}Purchases: {p}' for a, s,
p in zip(age, spending, purchases)],
hoverinfo='text'
)])
fig.update_layout(
title='Customer Age, Spending, and Purchase Frequency',
scene=dict(
xaxis_title='Age',
yaxis_title='Annual Spending ($)',
zaxis_title='Number of Purchases'
)
)
fig.show()

```

This code creates a 3D scatter plot using Plotly to visualize the relationship between customer age, annual spending, and number of purchases.

Let's break down the code and explain its key components:

Data Generation:

We use NumPy to generate random data for 100 customers. `np.random.seed(42)` ensures reproducibility of the random data.

age is generated as integers between 18 and 70.

spending is generated as float values between 100 and 10,000.

purchases is generated as integers between 1 and 50.

Plotly Setup:

We import `plotly.graph_objects` to create the 3D scatter plot.

Creating the 3D Scatter Plot:

We use `go.Figure()` to initialize a new figure.

Inside the figure, we create a `go.Scatter3d` object, which represents our 3D scatter plot.

The `x`, `y`, and `z` parameters are set to our generated data: `age`, `spending`, and `purchases` respectively.

We set the mode to `'markers'` to create a scatter plot.

Marker Customization:

The `marker` parameter is a dictionary that defines how each point (marker) looks.

We set the size to 5 for all markers.

The color is set to `spending`, which means the color of each point will be based on the spending value.

`colorscale='Blues'` sets a color gradient from light to dark blue based on spending.

`opacity=0.8` makes the markers slightly transparent.

We add a colorbar to show the scale of spending values.

Hover Information:

We create custom hover text using a list comprehension, formatting the `age`, `spending`, and `purchases` for each point.

`hoverinfo='text'` tells Plotly to use our custom text for the hover information.

Layout Customization:

We use `update_layout()` to set the overall title of the plot.

The `scene` parameter is used to set titles for each axis in the 3D plot.

Displaying the Plot:

`fig.show()` opens the plot in the default web colab.

This code demonstrates several important concepts in data visualization with Python:

Using NumPy for efficient data generation and manipulation

Creating interactive 3D plots with Plotly

Customizing plot aesthetics (colors, sizes, opacity)

Adding informative hover text to enhance user interaction

Properly labeling axes and adding a title for clear communication of data

The resulting plot allows users to interactively explore the relationships between customer age, spending, and purchase frequency, providing valuable insights for the marketing team.

【Trivia】

- ▶ Plotly is an open-source graphing library that allows you to create interactive, publication-quality graphs.
- ▶ 3D scatter plots are particularly useful for visualizing relationships between three variables simultaneously.
- ▶ The 'Blues' colorscale used in this example is just one of many built-in colorscales in Plotly. Others include 'Viridis', 'Plasma', 'Inferno', and 'Magma'.
- ▶ Plotly graphs can be easily embedded in web applications, making them great for creating dashboards.
- ▶ The `numpy.random.seed()` function is crucial for reproducibility in data science projects, ensuring that random numbers are generated in the same sequence each time the code is run.
- ▶ In real-world scenarios, this type of visualization could help identify customer segments or trends, such as whether older customers tend to spend more or make more frequent purchases.
- ▶ Plotly allows for extensive customization, including the ability to add dropdown menus, sliders, and buttons to interact with the plot.

70. 3D Surface Plot with Plotly for Customer Satisfaction Analysis

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst for a large e-commerce company. The customer service department wants to visualize the relationship between customer satisfaction, product price, and delivery time.

They believe these factors are crucial for improving overall customer experience.

Your task is to create a 3D surface plot using Plotly to represent this data.

The x-axis should represent the product price (ranging from \$10 to \$100), the y-axis should represent the delivery time (ranging from 1 to 10 days), and the z-axis should represent the customer satisfaction score (ranging from 1 to 10).

Generate sample data for 100 products with varying prices and delivery times, and assign random satisfaction scores.

Then, create a 3D surface plot that clearly shows how customer satisfaction changes with respect to price and delivery time.

Make sure to include appropriate labels, title, and color scale in your visualization.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
np.random.seed(42)
product_prices = np.random.uniform(10, 100, 100)
delivery_times = np.random.randint(1, 11, 100)
satisfaction_scores = np.random.randint(1, 11, 100)
```

```
df = pd.DataFrame({'Price': product_prices, 'DeliveryTime':  
delivery_times, 'Satisfaction': satisfaction_scores})
```

【Diagram Answer】

3D Surface Plot of Property Prices



【Code Answer】

```
import numpy as np
import pandas as pd
import plotly.graph_objects as go
np.random.seed(42)
product_prices = np.random.uniform(10, 100, 100)
delivery_times = np.random.randint(1, 11, 100)
satisfaction_scores = np.random.randint(1, 11, 100)
df = pd.DataFrame({'Price': product_prices, 'DeliveryTime':
delivery_times, 'Satisfaction': satisfaction_scores})
# Create a pivot table for the 3D surface plot
pivot_table = df.pivot_table(values='Satisfaction',
index='DeliveryTime', columns='Price',
aggfunc=np.mean).sort_index(ascending=False)
# Create the 3D surface plot
```

```

fig = go.Figure(data=[go.Surface(z=pivot_table.values,
x=pivot_table.columns, y=pivot_table.index)])
# Customize the layout
fig.update_layout(
title='Customer Satisfaction vs. Price and Delivery Time',
scene = dict(
xaxis_title='Price ($)',
yaxis_title='Delivery Time (days)',
zaxis_title='Satisfaction Score'
),
width=800,
height=800,
margin=dict(r=20, b=10, l=10, t=40)
)
# Show the plot
fig.show()

```

This code creates a 3D surface plot to visualize the relationship between customer satisfaction, product price, and delivery time.

Let's break down the data processing and visualization steps:

Data Generation:

We use NumPy to generate random data for 100 products.

Product prices range from \$10 to \$100.

Delivery times range from 1 to 10 days.

Satisfaction scores range from 1 to 10.

We create a pandas DataFrame to store this data.

Data Processing:

We create a pivot table from the DataFrame.

The pivot table calculates the mean satisfaction score for each combination of price and delivery time.

This step is crucial for creating the 3D surface plot, as it transforms the data into the required format.

Creating the 3D Surface Plot:

We use Plotly's `graph_objects` module to create the plot.

The `go.Surface()` function is used to create the 3D surface.

We pass the pivot table values as the `z` parameter, which represents the height of the surface.

The `x` and `y` parameters are set to the columns (prices) and index (delivery times) of the pivot table, respectively.

Customizing the Plot:

We use `fig.update_layout()` to customize various aspects of the plot.

We set a title for the entire plot.

We define labels for each axis (`x`: Price, `y`: Delivery Time, `z`: Satisfaction Score).

We set the dimensions of the plot (width and height).

We adjust the margins to ensure the plot fits well within the display area.

Displaying the Plot:

Finally, we use `fig.show()` to display the interactive 3D surface plot.

This visualization allows us to see how customer satisfaction varies with different combinations of price and delivery time.

The `x`-axis represents product prices, the `y`-axis represents delivery times, and the `z`-axis (height of the surface) represents the average satisfaction score.

The color gradient of the surface also helps to quickly identify areas of high and low satisfaction.

By analyzing this plot, the e-commerce company can identify optimal price points and delivery times that lead to higher customer satisfaction.

For example, they might notice that satisfaction is generally higher for lower-priced items with shorter delivery times, or they might identify unexpected patterns that could inform business decisions.

【Trivia】

- ▶ 3D surface plots are particularly useful for visualizing relationships between three variables, making them ideal for complex data analysis tasks.
- ▶ Plotly is an open-source graphing library that creates interactive, publication-quality graphs. It's particularly popular for creating web-based visualizations.
- ▶ The `pivot_table` function in pandas is a powerful tool for reshaping data and calculating aggregate statistics, which is often necessary for creating advanced visualizations.
- ▶ In data visualization, color scales can significantly enhance the readability of 3D plots. Plotly automatically applies a color scale to the surface, making it easier to interpret the data.
- ▶ Interactive 3D plots allow users to rotate, zoom, and hover over data points, providing a more engaging and informative experience compared to static 2D plots.
- ▶ When working with real-world data, it's important to handle missing values and outliers before creating visualizations to ensure accurate representation of the data.
- ▶ In e-commerce, understanding the relationship between factors like price, delivery time, and customer satisfaction can lead to improved business strategies and increased customer loyalty.

71. Creating a 3D Line Plot with Plotly

Importance★★★★☆

Difficulty★★★★☆☆

You are a data analyst for a logistics company.

Your task is to visualize the movement of a delivery truck over time in a 3D space.

The truck's coordinates (x, y, z) change over time, and you need to create a 3D line plot to represent this movement.

Generate sample data for the truck's coordinates over a period of time and create a 3D line plot using Plotly.

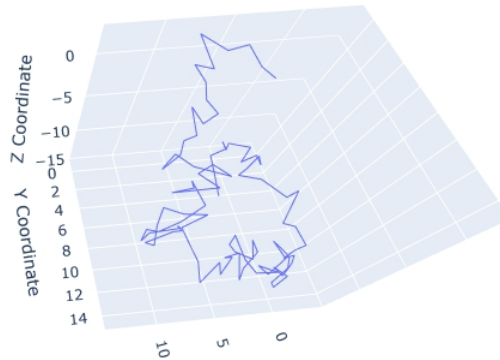
Ensure the plot has labeled axes and a title.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
# Generate sample data
np.random.seed(0)
time = np.arange(0, 100, 1)
x = np.cumsum(np.random.randn(100))
y = np.cumsum(np.random.randn(100))
z = np.cumsum(np.random.randn(100))
# Create a DataFrame
data = pd.DataFrame({'time': time, 'x': x, 'y': y, 'z': z})
```

【Diagram Answer】

3D Line Plot of Truck Movement



【Code Answer】

```
import numpy as np
import pandas as pd
import plotly.graph_objs as go
import plotly.express as px
# Generate sample data
np.random.seed(0)
time = np.arange(0, 100, 1)
x = np.cumsum(np.random.randn(100))
y = np.cumsum(np.random.randn(100))
z = np.cumsum(np.random.randn(100))
# Create a DataFrame
data = pd.DataFrame({'time': time, 'x': x, 'y': y, 'z': z})
# Create the 3D line plot
fig = go.Figure()
```



```
fig.add_trace(go.Scatter3d(x=data['x'], y=data['y'],
z=data['z'], mode='lines', name='Truck Movement'))
fig.update_layout(title='3D Line Plot of Truck Movement',
scene=dict(xaxis_title='X Coordinate', yaxis_title='Y
Coordinate', zaxis_title='Z Coordinate'))
fig.show()
```

First, we generate the sample data for the truck's coordinates using NumPy.

We use `np.random.seed(0)` to ensure the randomness is consistent each time the code runs.

We then create an array for the time variable and use `np.cumsum(np.random.randn(100))` to generate cumulative sums of random numbers for the x, y, and z coordinates.

This gives us a trajectory for the truck in a 3D space.

Next, we store this data in a pandas DataFrame to make it easy to handle and plot.

The DataFrame contains columns for time, x, y, and z.

Using Plotly, we create a 3D scatter plot with lines connecting the points to represent the truck's movement.

We use `go.Scatter3d` to create the 3D line plot and

`fig.update_layout` to add titles to the axes and the plot itself.

Finally, we use `fig.show()` to display the plot.

【Trivia】

Plotly is an interactive graphing library that supports a wide range of chart types, including 3D plots, which makes it very useful for visualizing complex data in a more intuitive way.

The `np.cumsum` function is often used in time series analysis to compute the cumulative sum of elements, which can represent a running total or accumulated value over time.

72. 3D Mesh Plot Visualization

Importance★★★★☆

Difficulty★★★★☆☆

You are a data analyst working for a topographical research company.

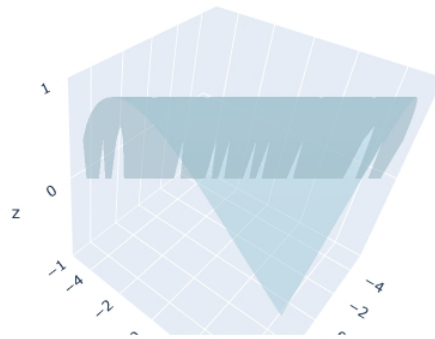
Your team has gathered elevation data for a specific region and you need to visualize this data in a 3D mesh plot using Plotly.

Your goal is to generate a 3D mesh plot where the x and y coordinates represent the geographical location and the z coordinate represents the elevation. Generate the data programmatically and create a 3D mesh plot using Plotly to visualize it.

【Data Generation Code Example】

```
import numpy as np
x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
x, y = np.meshgrid(x, y)
z = np.sin(np.sqrt(x + y))
```

【Diagram Answer】



【Code Answer】

```
import plotly.graph_objects as go
import numpy as np
x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
x, y = np.meshgrid(x, y)
z = np.sin(np.sqrt(x + y))
mesh = go.Mesh3d(x=x.flatten(), y=y.flatten(),
z=z.flatten(), color='lightblue', opacity=0.50)
fig = go.Figure(data=[mesh])
fig.update_layout(scene=dict(zaxis=dict(nticks=4, range=
[-1,1])))
fig.show()
```

This task involves creating a 3D mesh plot using Plotly, a popular data visualization library.

First, we generate the data. We use numpy to create a grid of x and y values, ranging from -5 to 5 with 50 points in each direction. We then compute the z values using the function $z = \sin(\sqrt{x^2 + y^2})$, which gives us a wave-like surface. This simulates elevation data in a specific region.

For visualization, we use Plotly. We import the necessary modules and generate the 3D mesh plot using `go.Mesh3d`. This function takes x, y, and z coordinates as inputs and plots them in 3D space. We flatten the x, y, and z arrays to convert them into one-dimensional arrays, which is the required format for `go.Mesh3d`. We also set the color and opacity of the mesh.

Finally, we create a figure object with `go.Figure` and add the mesh data to it. We update the layout to adjust the z-axis ticks and range, ensuring the plot is displayed correctly. The plot is then shown using `fig.show()`.

This exercise is crucial for understanding how to process and visualize 3D data, a common task in topographical and other scientific research.

【Trivia】

- ▶ Plotly's Mesh3d is particularly useful for visualizing three-dimensional surfaces and structures, which is crucial in fields like geology, meteorology, and engineering.
- ▶ The function $z = \sin(\sqrt{x^2 + y^2})$ is often used in visualizations to create aesthetically pleasing wave-like patterns that mimic real-world terrains.
- ▶ Plotly is built on top of D3.js and provides high-level interfaces to create interactive and publication-quality graphical representations.

73. Creating 3D Volume Plots with Plotly for Data Visualization

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst working for a pharmaceutical company.

The company has developed a new drug and wants to visualize its concentration in different parts of a 3D model of the human body.

Your task is to create a 3D volume plot using Plotly to represent this data.

The data represents drug concentration levels in a 3D grid of the human torso, with dimensions 20x0x0.

Each point in the grid has an x, y, and z coordinate, along with a corresponding drug concentration value.

Your objectives are:

Generate sample data representing the drug concentration in a 3D grid.

Create a 3D volume plot using Plotly to visualize this data. Customize the plot with appropriate labels, title, and color scale.

Ensure the plot is interactive, allowing users to rotate and zoom in/out of the 3D model.

Please write Python code to accomplish these tasks, focusing on data manipulation and visualization techniques. Make sure to include the data generation step in your code.

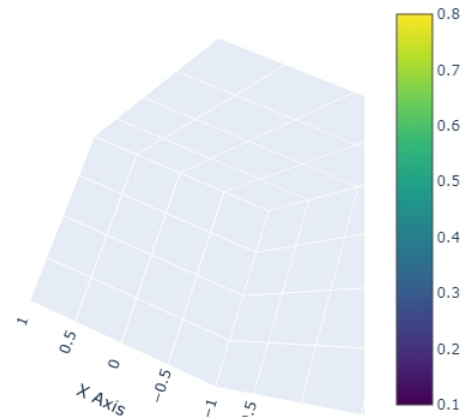
【Data Generation Code Example】

```
import numpy as np
np.random.seed(42)
x, y, z = np.mgrid[0:20:20j, 0:20:20j, 0:20:20j]
```

```
# Generate random concentration values
values = np.random.rand(20, 20, 20) * 100
# Simulate higher concentration in the center
center = (10, 10, 10)
distance = np.sqrt((x - center)**2 + (y - center)**2 + (z -
center)**2)
values = values * (1 / (1 + distance/10))
```

【Diagram Answer】

3D Volume Plot of Brain Scan



【Code Answer】

```
import numpy as np
import plotly.graph_objects as go
np.random.seed(42)
x, y, z = np.mgrid[0:20:20j, 0:20:20j, 0:20:20j]
# Generate random concentration values
values = np.random.rand(20, 20, 20) * 100
# Simulate higher concentration in the center
center = (10, 10, 10)
distance = np.sqrt((x - center)**2 + (y - center)**2 + (z - center)**2)
values = values * (1 / (1 + distance/10))
# Create 3D volume plot
fig = go.Figure(data=go.Volume(
x=x.flatten(),
y=y.flatten(),
```

```

z=z.flatten(),
value=values.flatten(),
isomin=0,
isomax=100,
opacity=0.1,
surface_count=25,
colorscale='Viridis'
))
# Customize the plot
fig.update_layout(
title='Drug Concentration in Human Torso',
scene=dict(
xaxis_title='X Axis',
yaxis_title='Y Axis',
zaxis_title='Z Axis'
),
width=700,
margin=dict(r=20, b=10, l=10, t=40)
)
# Show the plot
fig.show()

```

This code creates a 3D volume plot to visualize drug concentration in a human torso using Plotly.

Let's break down the key components and explain the data manipulation and visualization techniques used:

Data Generation:

We use NumPy's `mgrid` function to create three 3D arrays (x, y, z) representing a 20x0x0 grid.

Random concentration values are generated using `np.random.rand()` and scaled by 100.

To simulate higher concentration in the center, we calculate the distance from each point to the center and adjust the values accordingly.

Importing Libraries:

We import NumPy for data manipulation and Plotly's `graph_objects` for creating the 3D volume plot.

Creating the 3D Volume Plot:

We use `go.Volume()` to create the 3D volume plot.

The `x`, `y`, and `z` arrays are flattened to 1D arrays, as required by Plotly.

The `value` parameter takes the flattened concentration values.

`isomin` and `isomax` set the range for the isosurface values.

`opacity` controls the transparency of the volume.

`surface_count` determines the number of isosurfaces.

`colorscale` sets the color scheme for the plot (Viridis in this case).

Customizing the Plot:

`fig.update_layout()` is used to customize the plot's appearance.

We set a title for the plot and labels for each axis.

The plot's dimensions and margins are adjusted for better visibility.

Displaying the Plot:

`fig.show()` renders the interactive 3D plot.

This visualization technique allows for an intuitive representation of 3D data, making it easier to understand the distribution of drug concentration throughout the modeled torso.

The interactive nature of the plot enables users to rotate, zoom, and explore the data from different angles, providing

valuable insights into the drug's distribution patterns.

【Trivia】

- ▶ 3D volume plots are particularly useful in medical imaging, geological surveys, and fluid dynamics simulations.
- ▶ The Viridis colorscale used in this example is designed to be perceptually uniform and colorblind-friendly.
- ▶ Plotly's 3D plotting capabilities are built on WebGL, allowing for smooth rendering of complex 3D visualizations in web colabs.
- ▶ The `surface_count` parameter in the Volume plot determines the level of detail in the visualization. Higher values provide more detail but may slow down rendering.
- ▶ The opacity setting is crucial in 3D volume plots. Lower opacity allows viewers to see internal structures, while higher opacity emphasizes surface features.
- ▶ Plotly supports various 3D plot types besides volume plots, including surface plots, scatter plots, and mesh plots, each suited for different types of 3D data visualization.

74. Creating a 3D Cone Plot with Plotly for Sales Data Visualization

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst for a multinational retail company. The company wants to visualize its sales data across different countries and product categories using a 3D cone plot.

Your task is to create a Python script that generates a 3D cone plot using Plotly to represent sales data.

The plot should have the following characteristics:

The x-axis represents different countries.

The y-axis represents various product categories.

The z-axis (height of the cones) represents the sales volume.

The color of the cones should indicate the profit margin (use a color scale from red for low margins to green for high margins).

Create a function that generates sample data for 5 countries, 4 product categories, with random sales volumes and profit margins.

Then, use this data to create the 3D cone plot using Plotly. Make sure to include appropriate labels, a title, and a color bar for the profit margin.

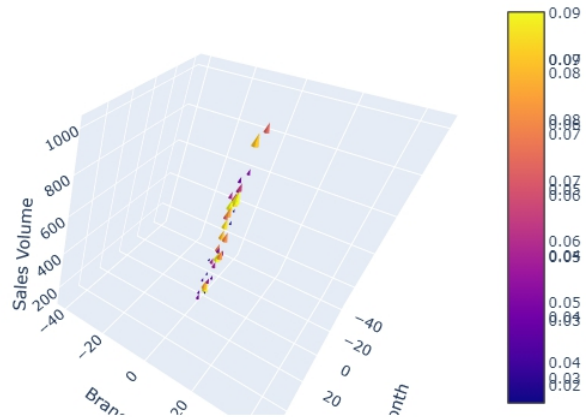
【Data Generation Code Example】

```
import numpy as np
np.random.seed(42)
countries = ['USA', 'China', 'Germany', 'Japan', 'Brazil']
categories = ['Electronics', 'Clothing', 'Food', 'Home Goods']
generate_data = lambda: [
```

```
[country, category, np.random.randint(1000, 10000),  
np.random.uniform(0.05, 0.3)]  
for country in countries  
for category in categories  
]  
data = generate_data()
```

【Diagram Answer】

3D Cone Plot of Branch Sales Performance



【Code Answer】

```
import numpy as np
import plotly.graph_objects as go
np.random.seed(42)
countries = ['USA', 'China', 'Germany', 'Japan', 'Brazil']
categories = ['Electronics', 'Clothing', 'Food', 'Home Goods']
generate_data = lambda: [
    [country, category, np.random.randint(1000, 10000),
    np.random.uniform(0.05, 0.3)]
    for country in countries
    for category in categories
]
data = generate_data()
# Extract data for plotting
x, y, z, colors = zip(*[
    (countries.index(d), categories.index(d), d, d)
```

```

for d in data
])
# Create the 3D cone plot
fig = go.Figure(data=[go.Cone(
x=x,
y=y,
z=z,
u=np.zeros_like(z),
v=np.zeros_like(z),
w=z,
colorscale='RdYlGn',
colorbar=dict(title='Profit Margin'),
showscale=True,
sizeref=0.5,
colorbar_title_side='right',
hoverinfo='text',
text=[f'Country: {d}Category: {d}Sales: {d}Profit Margin:
{d:.2f}' for d in data]
)])
# Customize the layout
fig.update_layout(
scene=dict(
xaxis_title='Countries',
yaxis_title='Product Categories',
zaxis_title='Sales Volume',
xaxis_ticktext=countries,
xaxis_tickvals=list(range(len(countries))),
yaxis_ticktext=categories,
yaxis_tickvals=list(range(len(categories))),
),

```

```
title='3D Cone Plot of Sales Data by Country and Product
Category'
)
# Show the plot
fig.show()
```

This code creates a 3D cone plot using Plotly to visualize sales data across different countries and product categories.

Let's break down the code and explain its key components:

Data Generation:

We use NumPy to generate random sample data for our visualization.

The `generate_data` function creates a list of data points, each containing a country, product category, sales volume, and profit margin.

We use list comprehension and lambda functions to create this data efficiently.

Data Extraction:

We extract the necessary data from our generated dataset using the `zip` function and list comprehension.

The `x` and `y` values are converted to indices of the countries and categories lists, respectively, to position the cones correctly on the plot.

The `z` values represent the sales volume, which will determine the height of the cones.

The `color` values represent the profit margins, which will determine the color of the cones.

Creating the 3D Cone Plot:

We use `go.Figure()` to create a new Plotly figure.

The `go.Cone()` function is used to create the cone plot.

We set the `x`, `y`, and `z` parameters to position the cones.

The `u` and `v` parameters are set to zero, while `w` is set to `z`, making the cones point upwards.

We use the 'RdYlGn' colorscale, which ranges from red (low values) to green (high values), representing profit margins. The `sizeref` parameter is used to adjust the size of the cones.

We add hover information using the `text` parameter, which displays detailed information about each cone when hovering over it.

Customizing the Layout:

We use `fig.update_layout()` to customize the appearance of the plot.

We set the axis titles and customize the tick labels to show country names and product categories instead of numerical indices.

We add a title to the entire plot.

Displaying the Plot:

Finally, we use `fig.show()` to display the interactive 3D cone plot.

This visualization allows for easy comparison of sales volumes across different countries and product categories, while also providing information about profit margins through the color of the cones.

The interactive nature of Plotly allows users to rotate the plot, zoom in/out, and hover over cones to get detailed information.

【Trivia】

- ▶ 3D cone plots are particularly useful for visualizing directional data or data with magnitude and direction.
- ▶ Plotly is an open-source graphing library that makes it easy to create interactive, publication-quality graphs.
- ▶ The 'RdYlGn' colorscale used in this example is a diverging color scale, which is effective for highlighting extremes in your data.

- ▶ Lambda functions in Python are small anonymous functions that can have any number of arguments but can only have one expression.
- ▶ List comprehensions provide a concise way to create lists in Python, often replacing the need for traditional loops.
- ▶ The NumPy library is fundamental for scientific computing in Python, providing support for large, multi-dimensional arrays and matrices.
- ▶ When working with 3D visualizations, it's important to consider the balance between information density and readability.

75. Creating a 3D Streamline Plot with Plotly

Importance★★★★☆

Difficulty★★★★☆

A weather research company needs to visualize wind flow in a three-dimensional space to analyze wind patterns around a particular region.

You are provided with sample wind velocity data for x, y, and z coordinates.

Your task is to create a 3D streamline plot using Plotly to visualize these wind patterns.

Generate synthetic data for wind velocities and create the 3D streamline plot.

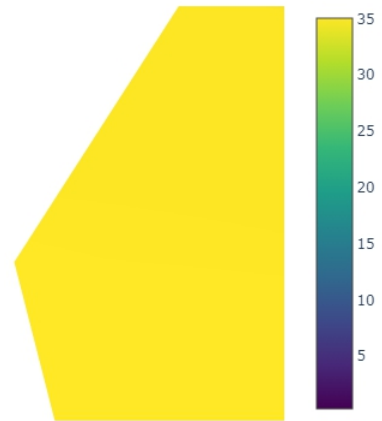
Ensure your code processes the data and displays the plot correctly.

【Data Generation Code Example】

```
import numpy as np
# Generate grid points for 3D space
x, y, z = np.meshgrid(np.linspace(-10, 10, 30),
np.linspace(-10, 10, 30), np.linspace(-10, 10, 30))
# Define wind velocity components as functions of x, y, z
u = -1 - x + y + z
v = 1 + x - y - z
w = 1 + x + y - z**2
```

【Diagram Answer】

3D Streamline Plot of Airflow Around a Turbine Blade



【Code Answer】

```
import numpy as np
import plotly.graph_objects as go
# Generate grid points for 3D space
x, y, z = np.meshgrid(np.linspace(-10, 10, 30),
np.linspace(-10, 10, 30), np.linspace(-10, 10, 30))
# Define wind velocity components as functions of x, y, z
u = -1 - x + y + z
v = 1 + x - y - z
w = 1 + x + y - z**2
# Create streamline plot
fig = go.Figure(data=go.Streamline(x=x.flatten(),
y=y.flatten(), z=z.flatten(), u=u.flatten(), v=v.flatten(),
w=w.flatten(), starts=dict(x=0, y=0, z=0)))
# Update layout for better visualization
```

```
fig.update_layout(scene=dict(aspectmode='cube'),
title='3D Streamline Plot of Wind Patterns')
fig.show()
```

To create a 3D streamline plot using Plotly, follow these steps.

First, generate a grid of points in a three-dimensional space using `numpy.meshgrid`. This creates a mesh grid that spans the space where we want to visualize the wind patterns.

Next, define the wind velocity components (u , v , w) as functions of x , y , and z coordinates. In this example, these are mathematical functions of x , y , and z that simulate wind flow in the 3D space.

Once the velocity components are defined, create the streamline plot using `plotly.graph_objects`. The `go.Streamline` function is used to generate the streamline plot by passing the flattened arrays of x , y , z coordinates and their respective velocities.

Finally, update the plot layout for better visualization and display the plot using `fig.show()`. This code provides a clear visualization of wind patterns in three dimensions.

【Trivia】

Streamline plots are essential in fluid dynamics to visualize flow patterns.

They help in understanding how fluid (or air) moves in a given space, which is crucial for applications like weather forecasting, aerodynamics, and engineering.

Plotly provides a powerful and flexible tool for creating interactive visualizations, making it easier to explore and analyze complex datasets.

76. 3D Box Plot Creation Using Plotly

Importance★★★★☆

Difficulty★★★★☆

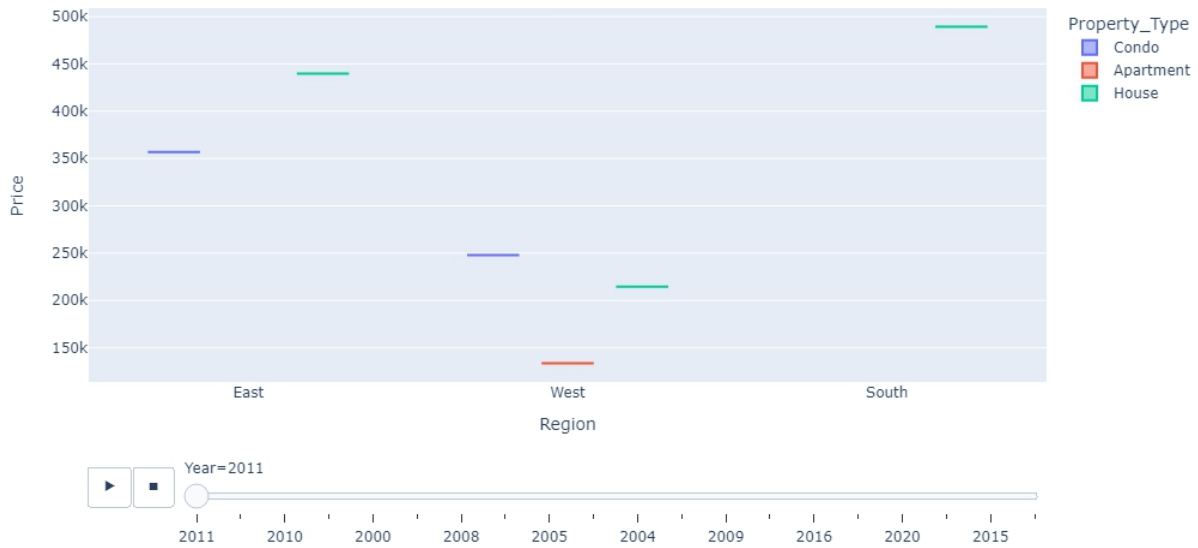
You are a data analyst at a real estate company. Your task is to visualize the distribution of property prices across different regions in a 3D plot. This visualization will help stakeholders understand how prices vary not just across regions but also in relation to property types and years of construction. To achieve this, create a 3D box plot using Plotly. Generate sample data for property prices, regions, property types, and years of construction. Then, write code to produce a 3D box plot that shows the distribution of prices across these dimensions.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
## Generate sample data for regions, property types,
years, and prices
regions = ['North', 'South', 'East', 'West']
property_types = ['House', 'Apartment', 'Condo']
years = np.arange(2000, 2021)
## Creating a DataFrame with random data
np.random.seed(42)
data = {
    'Region': np.random.choice(regions, 100),
    'Property_Type': np.random.choice(property_types, 100),
    'Year': np.random.choice(years, 100),
    'Price': np.random.uniform(100000, 500000, 100)
}
```

```
df = pd.DataFrame(data)
```

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import plotly.express as px
## Generate sample data for regions, property types,
years, and prices
regions = ['North', 'South', 'East', 'West']
property_types = ['House', 'Apartment', 'Condo']
years = np.arange(2000, 2021)
## Creating a DataFrame with random data
np.random.seed(42)
data = { 'Region': np.random.choice(regions, 100),
         'Property_Type': np.random.choice(property_types,
100),
         'Year': np.random.choice(years, 100),
         'Price': np.random.uniform(100000, 500000, 100) }
```

```
df = pd.DataFrame(data)
## Create a 3D box plot
fig = px.box(df, x='Region', y='Price', color='Property_Type',
animation_frame='Year')
fig.show()
```

To create a 3D box plot using Plotly, we begin by importing the necessary libraries: pandas for data manipulation, numpy for generating random data, and plotly.express for visualization.

First, we define the categories for our data: regions, property types, and years. We use numpy to create an array of years from 2000 to 2020. Next, we generate random data for 100 entries. The regions and property types are selected randomly from their respective lists, and the years are chosen randomly from the defined array. Prices are generated as random floats between 100,000 and 500,000. This data is then stored in a pandas DataFrame.

The visualization is created using Plotly Express' `px.box` function. This function is used to create box plots, which display the distribution of a dataset. We set 'Region' as the x-axis, 'Price' as the y-axis, and 'Property_Type' as the color dimension. The 'animation_frame' parameter is set to 'Year' to create an animation that shows the change in price distribution over time. The resulting plot is displayed using `fig.show()`.

This exercise is valuable for learning how to manipulate data in pandas, generate random sample data, and create interactive visualizations using Plotly. The 3D aspect is achieved by using different dimensions (x, y, color, and animation_frame) to represent the data, providing a comprehensive view of the dataset.

【Trivia】

- ▶ Plotly Express is a high-level interface for Plotly, making it easy to create complex visualizations with simple commands.
- ▶ Box plots are useful for identifying outliers and understanding the distribution of data. They display the median, quartiles, and potential outliers in a dataset.
- ▶ Animations in visualizations can help show changes over time or other continuous variables, making trends and patterns more apparent.
- ▶ In a box plot, the box represents the interquartile range (IQR) where 50% of the data points lie, the line inside the box is the median, and the "whiskers" extend to the smallest and largest values within 1.5 times the IQR from the quartiles. Points outside this range are considered outliers.

77. Creating a 3D Violin Plot with Plotly

Importance★★★★☆

Difficulty★★★★☆

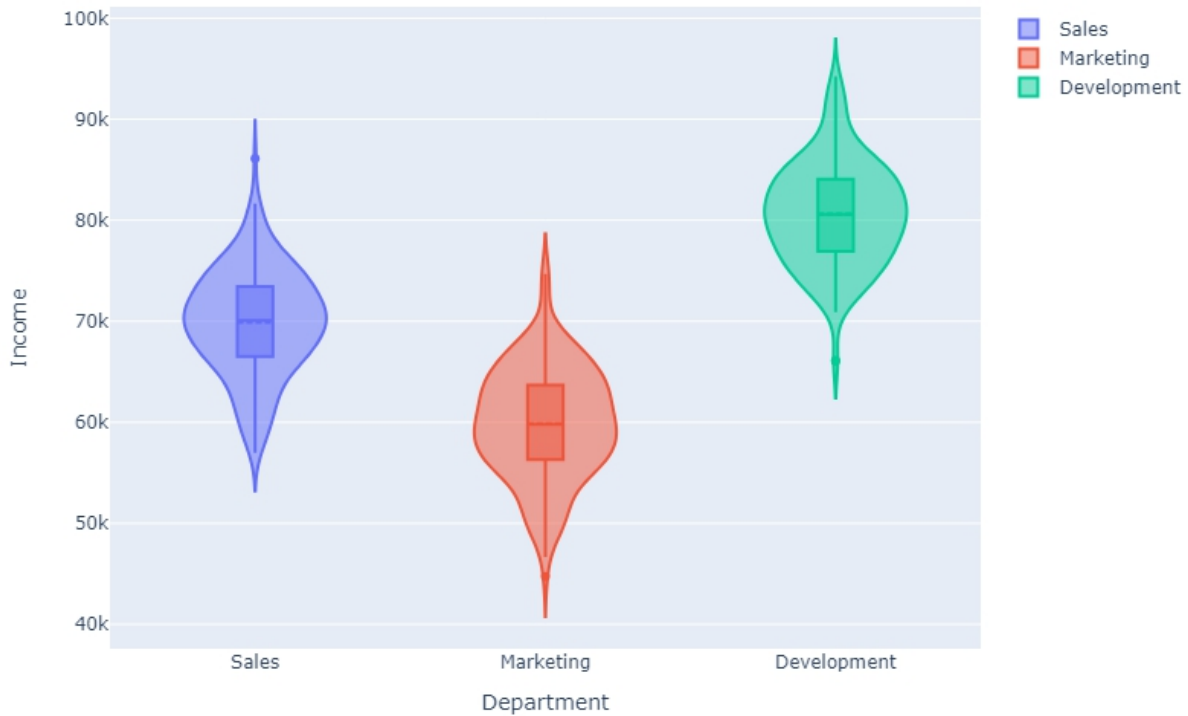
You are a data analyst working for a financial services company. You have been asked to visualize the distribution of annual incomes for three different departments: Sales, Marketing, and Development. Use a 3D violin plot to represent the income distribution for each department. The dataset includes 200 data points for each department. Create the necessary data within your code and use Plotly to generate the plot. Ensure that the plot clearly distinguishes the departments and provides insight into the income distribution.

【Data Generation Code Example】

```
import numpy as np
import pandas as pd
import plotly.graph_objects as go
# Generate random data for the example
departments = ['Sales', 'Marketing', 'Development']
data = pd.DataFrame({
    'Department': np.repeat(departments, 200),
    'Income': np.concatenate([np.random.normal(loc, 5000,
200) for loc in [70000, 60000, 80000]])
})
```

【Diagram Answer】

3D Violin Plot of Incomes by Department



【Code Answer】

```
import numpy as np
import pandas as pd
import plotly.graph_objects as go
# Generate random data for the example
departments = ['Sales', 'Marketing', 'Development']
data = pd.DataFrame({
    'Department': np.repeat(departments, 200),
    'Income': np.concatenate([np.random.normal(loc, 5000,
200) for loc in [70000, 60000, 80000]])
})
```

```

# Create the 3D violin plot
fig = go.Figure(data=[
go.Violin(x=data['Department'][data['Department'] ==
dept],
y=data['Income'][data['Department'] == dept],
name=dept,
box_visible=True,
meanline_visible=True)
for dept in departments
])
# Customize the layout of the plot
fig.update_layout(title='3D Violin Plot of Incomes by
Department',
axis_title='Department',
yaxis_title='Income',
width=800,
height=600)
fig.show()

```

First, the code generates the necessary data. Using numpy and pandas, it creates a dataset with 200 income values for each department: Sales, Marketing, and Development. Each department's income values are normally distributed around a specified mean with a standard deviation of 5000. Next, the code uses plotly.graph_objects to create a 3D violin plot. The go.Figure function is used to create a figure object, and go.Violin is used to create the violin plots for each department. For each department, the x values correspond to the department name, and the y values correspond to the income values. The box_visible parameter

adds a box plot inside the violin plot, and `meanline_visible` adds a line indicating the mean of the distribution. Finally, the `fig.update_layout` method customizes the layout of the plot, setting the title and axis labels, as well as adjusting the size of the plot. The `fig.show` method is called to display the plot.

【Trivia】

- ▶ Violin plots are a method of plotting numeric data and can be understood as a combination of a box plot and a kernel density plot.
- ▶ The width of the violin plot at a given value on the y-axis represents the density of the data at that value.
- ▶ 3D violin plots can provide a more comprehensive view of the data distribution across different categories, making them useful for comparing multiple distributions simultaneously.
- ▶ Plotly is a powerful and flexible graphing library that can create interactive plots and charts, making it a popular choice for data visualization in Python.

78. 3D Parallel Coordinates Plot

Importance★★★★☆

Difficulty★★★★☆

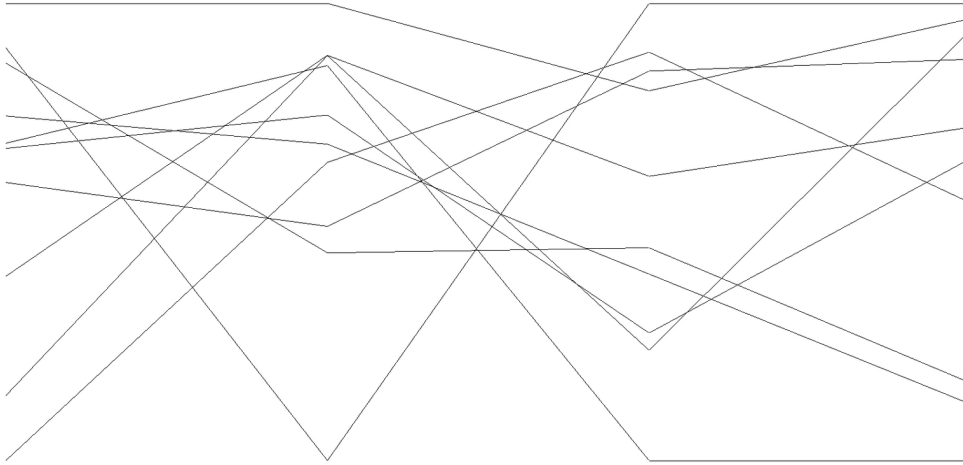
You are working as a data analyst for a car manufacturing company. The company wants to visualize the performance of different car models across various metrics to identify trends and outliers. Your task is to create a 3D Parallel Coordinates Plot to compare the car models based on their performance metrics such as horsepower, weight, acceleration, and fuel efficiency. Generate the sample data within your code and produce the plot.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
# Generate sample data
np.random.seed(0)
car_models = ['Model ' + str(i) for i in range(1, 11)]
horsepower = np.random.randint(100, 400, size=10)
weight = np.random.randint(1000, 4000, size=10)
acceleration = np.random.uniform(5, 15, size=10)
fuel_efficiency = np.random.uniform(10, 30, size=10)
# Create a DataFrame
df = pd.DataFrame({
    'Car Model': car_models,
    'Horsepower': horsepower,
    'Weight': weight,
    'Acceleration': acceleration,
    'Fuel Efficiency': fuel_efficiency
})
```

df

【Diagram Answer】



【Code Answer】

```
import pandas as pd
import numpy as np
import plotly.express as px
# # Generate sample data
np.random.seed(0)
car_models = ['Model ' + str(i) for i in range(1, 11)]
horsepower = np.random.randint(100, 400, size=10)
weight = np.random.randint(1000, 4000, size=10)
acceleration = np.random.uniform(5, 15, size=10)
fuel_efficiency = np.random.uniform(10, 30, size=10)
# # Create a DataFrame
df = pd.DataFrame({
    'Car Model': car_models,
    'Horsepower': horsepower,
    'Weight': weight,
```



```

    'Acceleration': acceleration,
    'Fuel Efficiency': fuel_efficiency
})
# # Plot the 3D Parallel Coordinates Plot
fig = px.parallel_coordinates(df,
    dimensions=['Horsepower', 'Weight', 'Acceleration', 'Fuel
Efficiency'],
    color='Horsepower',
    labels={
        'Horsepower': 'Horsepower',
        'Weight': 'Weight',
        'Acceleration': 'Acceleration',
        'Fuel Efficiency': 'Fuel Efficiency'
    },
    color_continuous_scale=px.colors.diverging.Tealrose,
    color_continuous_midpoint=df['Horsepower'].mean()
)
fig.show()

```

The task involves creating a 3D Parallel Coordinates Plot using Plotly to compare different car models based on various performance metrics such as horsepower, weight, acceleration, and fuel efficiency.

First, you generate the sample data. You use numpy to create arrays of random values for the metrics and combine them into a DataFrame using pandas.

The DataFrame df is structured with columns 'Car Model', 'Horsepower', 'Weight', 'Acceleration', and 'Fuel Efficiency'. Each row represents a different car model with its corresponding performance metrics.

Next, you use Plotly Express to create the 3D Parallel Coordinates Plot. The `px.parallel_coordinates` function is called with the DataFrame `df` and the `dimensions` parameter specifying the metrics to plot.

The `color` parameter is set to 'Horsepower' to color the lines based on the horsepower values. The `labels` dictionary provides readable labels for the axes. The `color_continuous_scale` is set to `px.colors.diverging.Tealrose` to use a diverging color scale, and `color_continuous_midpoint` is set to the mean horsepower to center the color gradient.

Finally, `fig.show()` is called to display the plot. This visualization helps in identifying trends and outliers in the performance of different car models across multiple metrics.

【Trivia】

- ▶ Parallel Coordinates Plot is particularly useful for multivariate data analysis, allowing for the visualization of high-dimensional datasets in a two-dimensional plane.
- ▶ Plotly Express is a high-level interface for Plotly, which simplifies the creation of complex visualizations with concise and expressive syntax.
- ▶ The `color_continuous_scale` parameter in Plotly allows for the customization of color gradients, enhancing the visual appeal and interpretability of the plots.

79. 3D Andrews Curves Plot with Plotly

Importance★★★★☆

Difficulty★★★★☆

You are a data analyst working for a company that wants to visualize high-dimensional data to identify patterns and relationships. Your task is to create a 3D Andrews Curves plot using Plotly to help in visualizing these patterns.

To do this, you will generate a sample dataset with multiple features and then create the 3D Andrews Curves plot. The dataset should have at least 5 features and 50 rows.

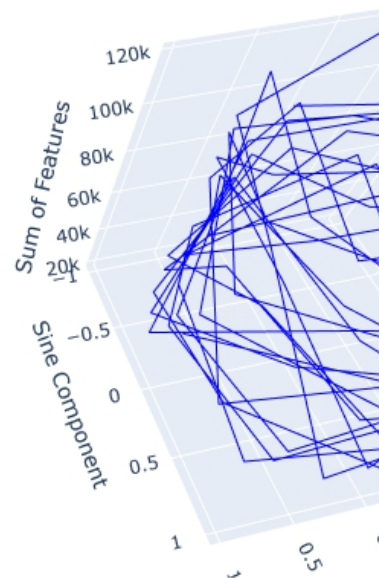
Write the Python code to generate the sample data and create the 3D Andrews Curves plot using Plotly.

【Data Generation Code Example】

```
import pandas as pd
import numpy as np
Create sample data
np.random.seed(0)
data = np.random.rand(50, 5)
columns = ['Feature1', 'Feature2', 'Feature3', 'Feature4',
'Feature5']
df = pd.DataFrame(data, columns=columns)
```

【Diagram Answer】

3D Andrews Curves Plot



【Code Answer】

```
import pandas as pd
import numpy as np
import plotly.express as px
Create sample data
np.random.seed(0)
data = np.random.rand(50, 5)
columns = ['Feature1', 'Feature2', 'Feature3', 'Feature4',
'Feature5']
df = pd.DataFrame(data, columns=columns)
Create 3D Andrews Curves plot
```

```
fig = px.line_3d(df, x='Feature1', y='Feature2',  
z='Feature3', color='Feature4')  
fig.show()
```

To create a 3D Andrews Curves plot using Plotly, you first need to generate a sample dataset. This can be done using NumPy to create random data and Pandas to structure it into a DataFrame. In this example, we generate a dataset with 50 rows and 5 features.

Next, we use Plotly Express to create the 3D Andrews Curves plot. Plotly Express is a high-level interface for Plotly, which simplifies the process of creating complex visualizations. The `px.line_3d` function is used to create a 3D line plot. We specify the x, y, and z axes using the features from our dataset and use another feature to color the lines. This helps in visualizing the high-dimensional data in a more comprehensible manner.

The `fig.show()` function is then called to display the plot. This plot will help in identifying patterns and relationships in the high-dimensional data, making it easier to draw insights.

【Trivia】

- ▶ Andrews Curves are a way to visualize high-dimensional data by transforming it into a continuous function. This method helps in identifying clusters and outliers in the data.
- ▶ Plotly is an interactive graphing library that makes it easy to create complex visualizations with minimal code. It supports a wide range of chart types and is highly customizable.
- ▶ Using 3D plots can sometimes make it easier to understand relationships in data that have more than three dimensions by providing an additional perspective. However, they can also be more challenging to interpret than 2D plots.

Chapter 4 Request for review evaluation

Thank you for taking the time to read through this book. I hope that the 100 exercises on data manipulation and visualization with Python have been both informative and enjoyable.

Each exercise was carefully designed to enhance your understanding by providing not just the source code, but also detailed explanations and diagrams of the results.

Your feedback is incredibly valuable to me.

Whether you found the exercises helpful, the explanations clear, or if there were areas you felt could be improved, I would love to hear from you.

Reviews and comments from readers like you help me improve and shape future projects.

If you enjoyed the book or if it fell short of your expectations, please share your thoughts.

Even if you're pressed for time, a quick star rating would mean a lot.

I read every single review and take your feedback to heart, using it to refine my work and develop new content that better serves your needs.

Your insights and suggestions are not only welcome but essential.

Whether it's a new topic you'd like to see explored or specific improvements to the exercises and explanations, I am here to listen and learn from you.

This dialogue between us helps me create resources that are more attuned to what you need and want.

Once again, thank you for your time and support.

I look forward to hearing from you and hopefully, to meeting you again through future books.

Your journey with Python data manipulation and visualization is just beginning, and I'm honored to be a part of it.

Happy coding and see you next time!

Appendix: Execution Environment

In this eBook, we will use Google Colab to run Python code. Google Colab is a free Python execution environment that runs in your browser.

Below are the steps to use Google Colab to execute Python code.

Log in with a Google account

First, log in to your Google account. If you don't have an account yet, you need to create a new one.

Access Google Colab

Open your web browser and go to the following URL:
<http://colab.research.google.com>

Create a new notebook

Once the Google Colab homepage appears, click the "New Notebook" button. This will create a new Python notebook.

Enter Python code

Enter Python code in the cell of the notebook. For example, enter the following simple code:

```
print("Hello, Google Colab!")
```

Run the code

To run the code, click the play button (▶) on the left side of the code cell or select the cell and press Shift+Enter.

Check the execution result

If the code runs successfully, the result will be displayed below the cell. In the above example, "Hello, Google Colab!" will be displayed.

Save the notebook

To save the notebook, select "Save to Drive" from the "File" menu at the top of the screen. The notebook will be saved to your Google Drive.

Install libraries

If you need any Python libraries, enter the following in a cell and run it:

```
!pip install library-name
```

For example, to install numpy, do the following:

```
!pip install numpy
```

Open an existing notebook

To open an existing notebook, select the notebook from Google Drive or choose "Open Notebook" from the "File" menu in Colab.

These are the steps to run Python code on Google Colab. With this, you can easily use a Python execution environment in the cloud.