ANDERSON SAGE

# MACHINE LEARNING SIMPLIFIED

An introduction to Supervised and Unsupervised Learning

# Machine Learning Simplified

## An introduction to Supervised and Unsupervised Learning

**Jose George**

**Copyright ©**

All rights reserved. Except for the quotation of short passages for the purposes of criticism and review, no part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher or a license from the Copyright Licensing Agency Limited.

Printed in the United States of America

**KJ Publishing**

USA

## CONTENTS


**CHAPTER ONE**

What is Machine Learning
Components of Machine Learning
Clay Analogy for Machine Learning
What are machine learning models?
Model Training

**CHAPTER TWO**

Introduction to the Five Machine Learning Steps
Step One: Define the Problem.
Supervised and Unsupervised Learning
Step Two: Build a Dataset
Step Three: Model Training
Step Four: Model Evaluation
Step Five: Model Inference
Introduction to Examples
Example One: House Price Prediction
Example Two: Book Genre Exploration
Example Three: Spill Detection from Video

**CHAPTER THREE**

Machine Learning with AWS
AWS Account Requirements
Reinforcement Learning with AWS DeepRacer
Putting Your Spin on AWS DeepRacer:
Introduction to Generative AI
Generative AI Models
Generative AI with AWS DeepComposer
GANs with AWS DeepComposer
Training Methodology
AR-CNN with AWS DeepComposer
Build a Custom GAN Model (Optional): Part 1

**CHAPTER FOUR**

Software Engineering Practices, Part I
Clean and Modular Code
Refactoring Code
Writing Clean Code
Writing Modular Code
Efficient Code
Documentation
Inline Comments
Docstrings
Multi-line docstring
Version Control In Data Science
Model versioning

**CHAPTER FIVE**

Software Engineering Practices, Part 2
Testing
Logging

**CHAPTER SIX**

Introduction to Object-Oriented Programming
Procedural versus object-oriented programming
Class, object, method, and attribute
OOP Syntax
Function versus method
Notes about OOP
Commenting object-oriented code
A Gaussian class

**Introduction**

Welcome to introduction to machining learning.

Machine learning is creating rapid and exciting changes across all levels of society.

- It is the engine behind the recent advancements in industries such as autonomous vehicles.
- It allows for more accurate and rapid translation of the text into hundreds of languages.
- It powers the AI assistants you might find in your home.
- It can help improve worker safety.
- It can speed up drug design

Machine learning is a complex subject area. Our goal in this lesson is to introduce you to some of the most common terms and ideas used in machine learning. I will then walk you through the different steps involved in machine learning and finish with a series of examples that use machine learning to solve real-world situations.

## Outline

This lesson is divided into the following sections:

- First, we'll discuss what machine learning is, common terminology, and common components involved in creating a machine learning project.
- Next, we'll step into the shoes of a machine learning practitioner. Machine learning involves using trained models to generate predictions and detect patterns from data. To understand the process, we'll break down the different steps involved and examine a common process that applies to the majority of machine learning projects.
- Finally, we'll take you through three examples using the steps we described to solve real-life scenarios that might be faced by machine learning practitioners.

### Section Objective

By the end of this section, you can do the following:

- Differentiate between supervised and unsupervised learning
- Identify problems that can be solved with machine learning concepts
- Describe commonly used algorithm including login regression, linear regression, and k-means
- Describe how model training and testing works
- Evaluate the performance of a machine learning model using metrics.

# CHAPTER ONE

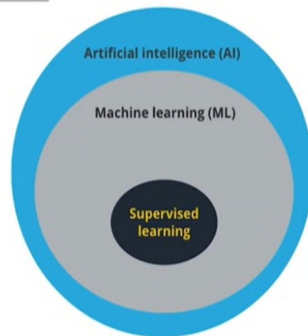
## What is Machine Learning

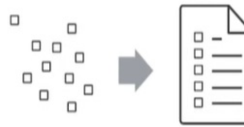
Machine learning (ML) is a modern software development technique and a type of artificial intelligence (AI) that enables computers to solve problems by using examples of real-world data. It allows computers to automatically learn and improve from experience without being explicitly programmed to do so.

Machine learning is part of the broader field of artificial intelligence. This field is concerned with the capability of machines to perform activities using human-like intelligence. Within machine learning there are several different kinds of tasks or techniques:

- **Supervised learning**, is a type of machine learning technique in which every training sample from the dataset has a corresponding label or output value associated with it. As a result, the algorithm learns to predict labels or output values. You can use supervised learning to do things like, predict the selling price of a house, or classify objects in an image. We will learn more about supervised learning in this lesson.

Supervised Learning

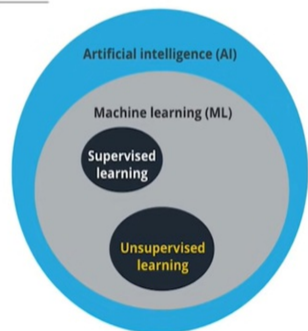


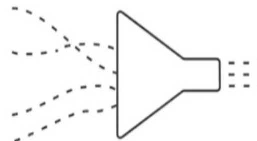



Example-driven training; every data point has a corresponding **label**

- In **unsupervised learning**, there are no labels for the training data. The algorithm tried to learn underlying patterns or distributions that govern the data. We will explore this in-depth in this lesson.

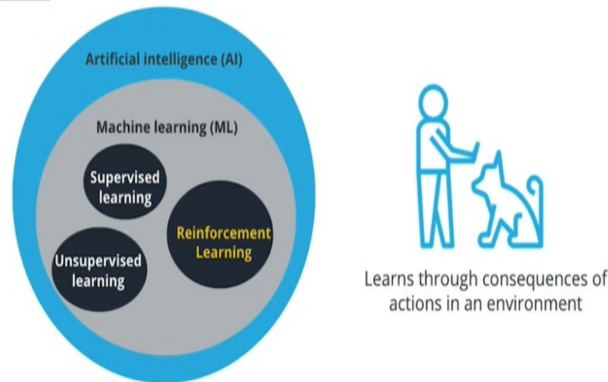
Unsupervised Learning



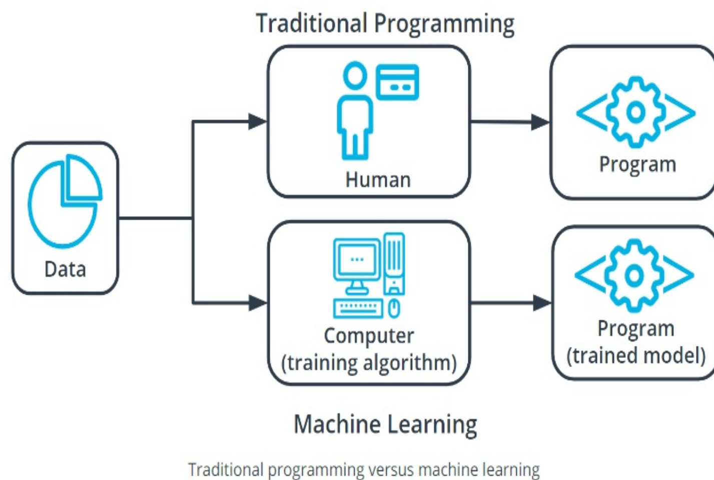



Patterns are discovered in unlabeled data

- In **reinforcement learning**, the algorithm figures out which actions to take in a situation to maximize a reward (in the form of a number) on the way to reaching a specific goal. This is a completely different approach than supervised and unsupervised learning. We will dive deep into this in the next lesson.




**Different between machine learning and traditional programming-based approaches**




Traditional programming versus machine learning

In traditional problem-solving with software, a person analyzes a problem and engineers a solution in code to solve that problem. For many real-world problems, this process can be laborious (or even impossible) because a correct solution would need to consider a vast number of edge cases.

Imagine, for example, the challenging task of writing a program that can detect if a cat is present in an image. Solving this in the traditional way would require careful attention to details like varying lighting conditions,

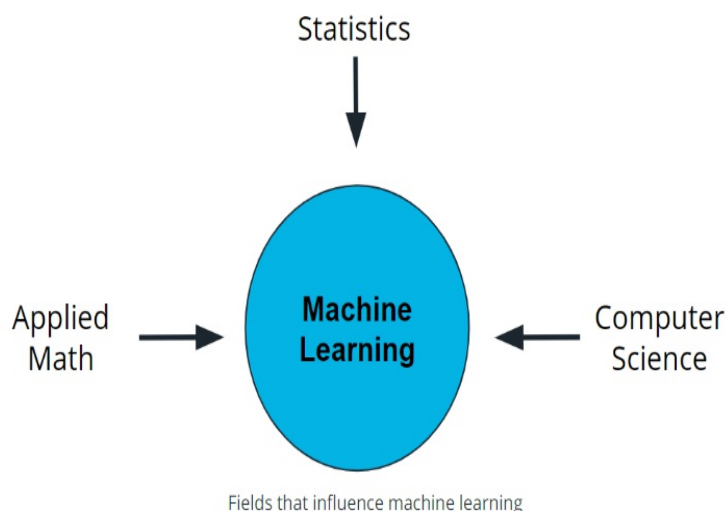
different types of cats, and various poses a cat might be in.

In machine learning, the problem solver abstracts away part of their solution as a flexible component called a model, and uses a special program called a model training algorithm to adjust that model to real-world data. The result is a trained model which can be used to predict outcomes that are not part of the data set used to train it.

In a way, machine learning automates some of the statistical reasoning and pattern-matching the problem solver would traditionally do.

The overall goal is to use a model created by a model training algorithm to generate predictions or find patterns in data that can be used to solve a problem.

**Understanding Terminology**



Fields that influence machine learning

Machine learning is a new field created at the intersection of statistics,

applied math, and computer science. Because of the rapid and recent growth of machine learning, each of these fields might use slightly different formal definitions of the same terms.

## Terminology

Machine learning, or ML, is a modern software development technique that enables computers to solve problems by using examples of real-world data.

In supervised learning, every training sample from the dataset has a corresponding label or output value associated with it. As a result, the algorithm learns to predict labels or output values.

In reinforcement learning, the algorithm figures out which actions to take in a situation to maximize a reward (in the form of a number) on the way to reaching a specific goal.

In unsupervised learning, there are no labels for the training data. A machine learning algorithm tries to learn the underlying patterns or distributions that govern the data.

# Components of Machine Learning

#### Models

In machine learning, all tasks are resolved with three primary components, namely:

- A machine learning model
- A model training algorithm
- A model inference algorithm

# Clay Analogy for Machine Learning

You can understand the relationships between these components by imagining the stages of crafting a teapot from a lump of clay.

- First, you start with a block of raw clay. At this stage, the clay can be molded into many different forms and be used to serve many different purposes. You decide to use this lump of clay to make a teapot.
- So how do you create this teapot? You inspect and analyze the raw clay and decide how to change it to make it look more like the teapot you have in mind.
- Next, you mold the clay to make it look more like the teapot that is your goal.

Congratulations! You've completed your teapot. You've inspected the materials, evaluated how to change them to reach your goal, and made the changes, and the teapot is now ready for your enjoyment.

# What are machine learning models?

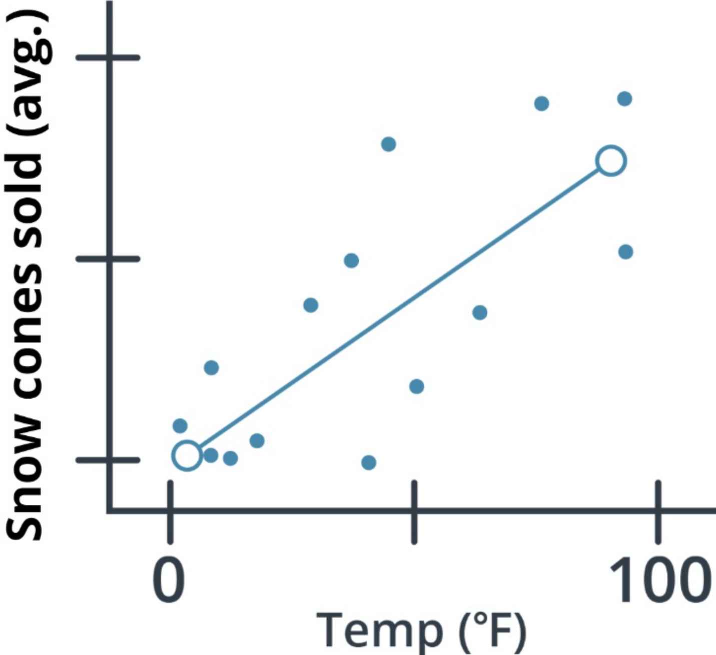
A machine learning model, like a piece of clay, can be molded into many different forms and serve many different purposes. A more technical definition would be that a machine learning model is a block of code or framework that can be modified to solve different but related problems based on the data provided.

**Important**

*A model is an extremely generic program (or block of code), made specific by the data used to train it. It is used to solve different problems.*

***Example 1***

Imagine you own a snow cone cart, and you have some data about the average number of snow cones sold per day based on the high temperature. You want to better understand this relationship to make sure you have enough inventory on hand for those high sales days.



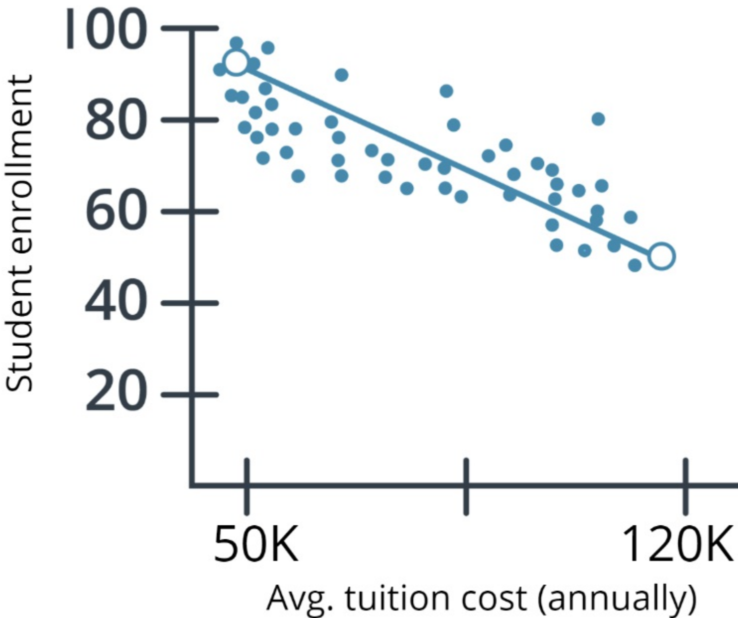
Snow cones sold regression chart

In the graph above, you can see one example of a model, a linear regression model (indicated by the solid line). You can see that, based on the data provided, the model predicts that as the high temperate for the day increases so do the average number of snow cones sold. Sweet!

***Example 2***

Let's look at a different example that uses the same linear regression model, but with different data and to answer completely different questions.

Imagine that you work in higher education and you want to better understand the relationship between the cost of enrollment and the number of students attending college. In this example, our model predicts that as the cost of tuition increases the number of people attending college is likely to decrease.




Average tuition regression chart

Using the same linear regression model (indicated by the solid line), you can see that the number of people attending college does go down as the cost increases.

*Both examples showcase that a model is a generic program made specific by the data used to train it.*

# Model Training

How are model training algorithms used to train a model?

In the preceding section, we talked about two key pieces of information: a model and data. In this section, we show you how those two pieces of information are used to create a trained model. This process is called *model training*.

**Model training algorithms work through an interactive process**

Let's revisit our clay teapot analogy. We've gotten our piece of clay, and now we want to make a teapot. Let's look at the algorithm for molding clay and how it resembles a machine learning algorithm:

- **Think about the changes that need to be made**. The first thing you would do is inspect the raw clay and think about what changes can be made to make it look more like a teapot. Similarly, a model training algorithm uses the model to process data and then compares the results against some end goal, such

as our clay teapot.

- **Make those changes**. Now, you mold the clay to make it look more like a teapot. Similarly, a model training algorithm gently nudges specific parts of the model in a direction that brings the model closer to achieving the goal.
- **Repeat**. By iterating over these steps over and over, you get closer and closer to what you want until you determine that you're close enough that you can stop.

Think about the changes that need to be made

Make those changes

### Model Inference: Using Your Trained Model

Now you have our completed teapot. You inspected the clay, evaluated the changes that needed to be made, and made them, and now the teapot is ready for you to use. Enjoy your tea!

*So what does this mean from a machine learning perspective*? We are ready to use the model inference algorithm to generate predictions using the trained model. This process is often referred to as **model inference**.

A finished teapot

**Quiz**

Which of the following are the primary components used in machine learning? (Select Multiple Boxes)

A. A model integrity algorithm
B. A machine learning model
C. A model training algorithm
D. A preparation algorithm
E. A model inference algorithm

Think back to the clay teapot analogy. Is it true or false that you always need to have an idea of what your're making when you are handling your raw block of clay?

A. True
B. False

Ans: B

**Terminology**

A **model** is an extremely generic program, made specific by the data used to train it.

**Model training algorithms** work through an interactive process where the current model iteration is analyzed to determine what changes can be made to get closer to the goal. Those changes are made and the iteration continues until the model is evaluated to meet the goals.

**Model inference** is when the trained model is used to generate predictions.

## CHAPTER TWO

# Introduction to the Five Machine Learning Steps

**Major Steps in the Machine Learning Process**

In the preceding diagram, you can see an outline of the major steps of the machine learning process. Regardless of the specific model or training algorithm used, machine learning practitioners practice a common workflow to accomplish machine learning tasks.

These steps are iterative. In practice, that means that at each step along the process, you review how the process is going. Are things operating as you expected? If not, go back and revisit your current step or previous steps to try and identify the breakdown.

Steps of machine learning

The rest of this book is designed around these very important steps. Check through them again here and get ready to dive deep into each of them.

- Step One: Define the problem
- Step Two: Build the dataset
- Step Three: Train the model
- Step Four: Evaluate the model

- Step Five: Inference (Use the model)

# Step One: Define the Problem.

**How do You Start a Machine Learning Task?**

- Define a very specific task.
  - Think back to the snow cone sales example. Now imagine that you own a frozen treats store and you sell snow cones along with many other products. You wonder, "‘How do I increase sales?" It's a valid question, but it's the **opposite** of a very specific task. The following examples demonstrate how a machine learning practitioner might attempt to answer that question.
    - “Does adding a $1.00 charge for sprinkles on a hot fudge sundae increase the sales of hot fudge sundaes?”
    - “Does adding a $0.50 charge for organic flavors in your snow cone increase the sales of snow cones?”
- Identify the machine learning task we might use to solve this problem.
  - This helps you better understand the data you need for a project.

**What is a Machine Learning Task?**

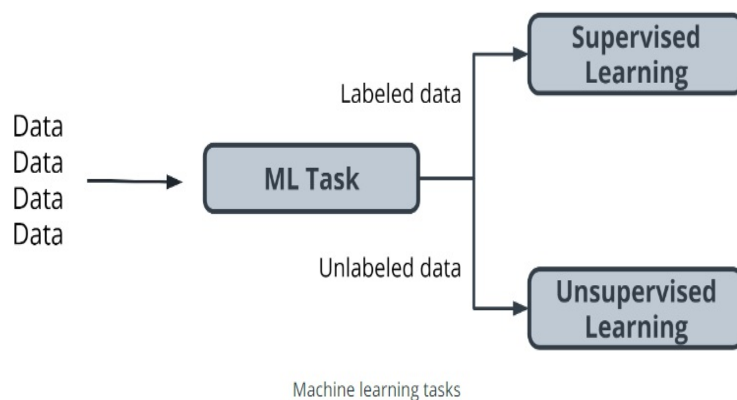
All model training algorithms, and the models themselves, take data as their input. Their outputs can be very different and are classified into a few different groups based on the task they are designed to solve. Often, we use the kind of data required to train a model as part of defining a machine-learning task.

In this lesson, we will focus on two common machine-learning tasks:

- Supervised learning
- Unsupervised learning

# Supervised and Unsupervised Learning

The presence or absence of labelling in your data is often used to identify a machine learning task.

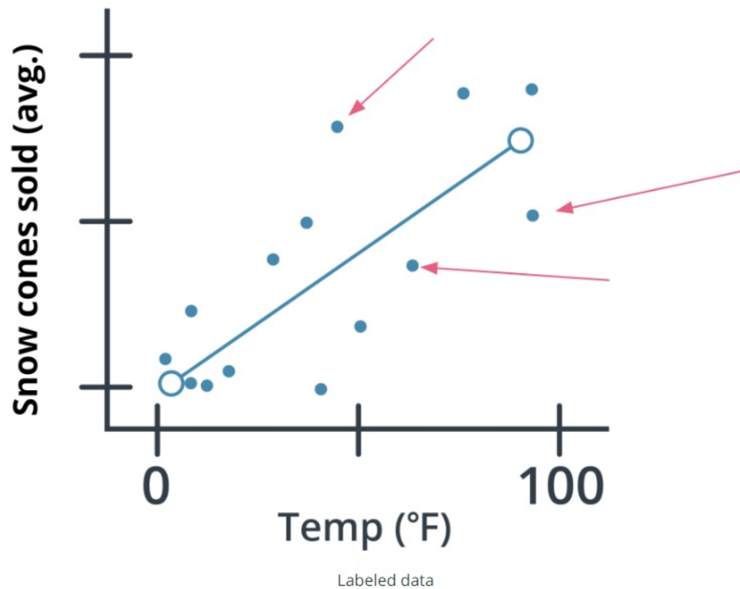


Machine learning tasks

**Supervised tasks**

A task is supervised if you are using labeled data. We use the term labeled to refer to data that already contains the solutions, called labels.

***For example***: *Predicting the number of snow cones sold based on the temperatures is an example of supervised learning.*



Labeled data

In the preceding graph, the data contains both a temperature and the number of snow cones sold. Both components are used to generate the linear regression shown on the graph. Our goal was to predict the number of snow cones sold, and we feed that value into the model. We are providing the model with labeled data and therefore, we are performing a supervised machine learning task.

**Unsupervised tasks**

A task is considered to be unsupervised if you are using unlabeled data. This means you don't need to provide the model with any kind of label or solution while the model is being trained.

Let's take a look at unlabeled data.

- Take a look at the preceding picture. Did you notice the tree in the picture? What you just did, when you noticed the object in the picture and identified it as a tree, is called labeling the picture. Unlike you, a computer just sees that image as a matrix of pixels of varying intensity.
- Since this image does not have the labeling in its original data, it is considered unlabeled.

**How do we classify tasks when we don't have a label?**

Unsupervised learning involves using data that doesn't have a label. One common task is called clustering. Clustering helps to determine if there are any naturally occurring groupings in the data.

Let's look at an example of how clustering in unlabeled data works.

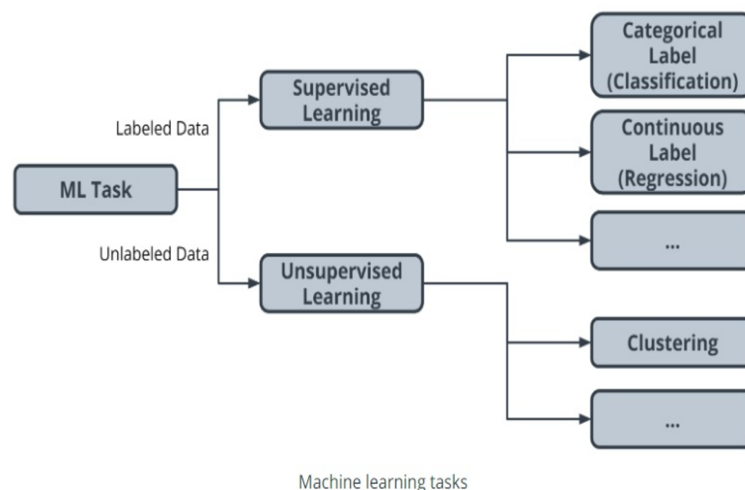
**Identifying book micro-genres with unsupervised learning**

Imagine that you work for a company that recommends books to readers.

The assumption: You are fairly confident that micro-genres exist, and that there is one called Teen Vampire Romance. Because you don’t know which

micro-genres exist, you can't use **supervised learning** techniques.

This is where the unsupervised learning clustering technique might be able to detect some groupings in the data. The words and phrases used in the book description might provide some guidance on a book's micro-genre.



Machine learning tasks

Initially, we divided tasks based on the presence or absence of labeled data while training our model. Often, tasks are further defined by the type of label which is present.

In **supervised learning**, there are two main identifiers you will see in machine learning:

- A **categorical** label has a discrete set of possible values. In a machine learning problem in which you want to identify the type of flower based on a picture, you would train your model using images that have been labeled with the categories of flower you would want to identify. Furthermore, when you work with categorical labels, you often carry out classification tasks*, which

are part of the supervised learning family.

- A **continuous** (regression) label does not have a discrete set of possible values, which often means you are working with numerical data. In the snow cone sales example, we are trying to predict the number* of snow cones sold. Here, our label is a number that could, in theory, be any value.

In unsupervised learning, clustering is just one example. There are many other options, such as deep learning.

**Quiz**

Match the task with it’s corresponding data type

i. Supervised learning
ii. Unsupervised learning

| Data Used | Machine learning task |
| --- | --- |
| Labeled data | |
| | |

**Terminology**

- **Clustering**. Unsupervised learning task that helps to determine if there are any naturally occurring groupings in the data.
- A **categorical label** has a discrete set of possible values, such as "is a cat" and "is not a cat."
- A **continuous (regression) label** does not have a discrete set of possible values, which means possibly an unlimited number of possibilities.

- **Discrete**: A term taken from statistics referring to an outcome taking on only a finite number of values (such as days of the week).
- A **label** refers to data that already contains the solution.
- Using **unlabeled** data means you don't need to provide the model with any kind of label or solution while the model is being trained.

**Additional Reading**

- The AWS Machine Learning blog is a great resource for learning more about projects in machine learning.
- You can use Amazon SageMaker to calculate new stats in Major League Baseball.
- You can also find an article on Flagging suspicious healthcare claims with Amazon SageMaker on the AWS Machine Learning blog.
- What kinds of questions and problems are good for machine learning?

**Quiz: Define the Problem**

1. Which of the following problem statements fit the definition of a regression-based task?
   A. I want to detect when my cat jumps on the dinner table, so I set up a camera and write a program to determine if my cat is in the frame or is not in the frame.
   B. I want to determine the expected reading time for online news articles, so I collect data on my reading time for a week and write a browser plugin to use that data to predict the reading time for new articles.
   C. I believe my customers fall into one of many customer segments, but I don't know what those segments are in advance. After

asking for permission, I collect a bunch of data on their actions when they use my product and try to determine if there are any collections of users that behave in similar ways

D. I work for a show company and want to provide a service to help parents predict their children's shoe sizes for any particular age. Within this system, I represent shoe size as a continuum of values and then round to the nearest shoe size.

Answer: B, D

2. As a machine learning practitioner, you're working with stakeholders on music streaming app. Your supervisor asks, "How can we increase the average number of minutes a customer spends listening on our app"?

This is a broad question (too broad) with many different potential factors affecting how long a customer might spend listening to music.

How might you change the scope or redefine the question to be better suited, and more concise, for a machine learning task?

A. Will changing the frequency of when we start playing ad affect how long a customer listens to music on our service.
B. Will creating customer playlist encourage customers to listen to music longer?
C. Will creating artist interviews about their songs increase how long our customers spend listening to music?

Answer: All of the Above

# Step Two: Build a Dataset

The next step in the machine learning process is to build a dataset that can be used to solve your machine learning-based problem. Understanding the data needed helps you select better models and algorithms so you can build

more effective solutions.

The most important step of the machine learning process

Working with data is perhaps the most overlooked—yet most important—step of the machine learning process. In 2017, an O'Reilly study showed that machine learning practitioners spend 80% of their time working with their data.

## The Four Aspects of Working with Data

Steps of working with data

You can take an entire class just on working with, understanding, and processing data for machine learning applications. Good, high-quality data is essential for any kind of machine learning project. Let's explore some of the common aspects of working with data.

## Data collection

Data collection can be as straightforward as running the appropriate SQL queries or as complicated as building custom web scraper applications to collect data for your project. You might even have to run a model over your data to generate needed labels. Here is the fundamental question:

*Does the data you've collected match the machine learning task and problem you have defined?*

### Data inspection

The quality of your data will ultimately be the largest factor that affects how well you can expect your model to perform. As you inspect your data, look for:

- Outliers
- Missing or incomplete values
- Data that needs to be transformed or preprocessed so it's in the correct format to be used by your model

### Summary statistics

Models can assume how your data is structured.

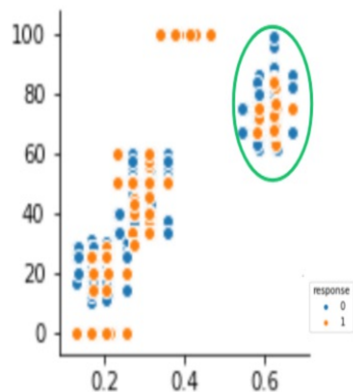
Now that you have some data in hand it is a good best practice to check that your data is in line with the underlying assumptions of your chosen machine learning model.

With many statistical tools, you can calculate things like the mean, inner-quartile range (IQR), and standard deviation. These tools can give you insight into the scope, scale, and shape of the dataset.

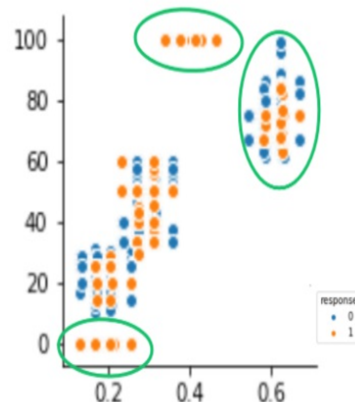
### Data visualization

You can use data visualization to see outliers and trends in your data and to help stakeholders understand your data.

Look at the following two graphs. In the first graph, some data seems to have clustered into different groups. In the second graph, some data points might be outliers.



Some of the data seems to cluster in groups



Some of the data points seem to be outliers

## Terminology

Impute is a common term referring to different statistical tools which can be used to calculate missing values from your dataset.

Outliers are data points that are significantly different from others in the same sample.

## Additional reading

In machine learning, you use several statistical-based tools to better understand your data. The sklearn library has many examples and tutorials, such as this example demonstrating outlier detection on a real dataset.

**Quiz: Build A Dataset**

Use this series of True / False questions to test your knowledge of different parts of building your dataset. Some questions are asking you to think just a little bit deeper about some of the content you just learned.

1. True or false: Your data requirements will not change based on the machine learning task you are using.
   A. True
   B. False

   Answer: B
2. True or false: models are universal, so date is not relevant.
   A. True
   B. False

   Answer: B
3. True or false: data needs to be formatted so that is compatible with the model and model training algorithm you plan to use
   A. True
   B. False

   Answer: A
4. True or false: Data visualizations are they only way to identify outliers in your data.
   A. True
   B. False

   Answer: B
5. True or false: After you start using your model(performing inference), you don’t need to check the new data that it received.
   A. True
   B. False

   Answer: B

# Step Three: Model Training

**Splitting your Dataset**

The first step in model training is to randomly split the dataset. This allows you to keep some data hidden during training, so that data can be used to evaluate your model before you put it into production. Specifically, you do this to test against the bias-variance trade-off. If you're interested in learning more, see the Further learning and reading section.

Splitting your dataset gives you two sets of data:

- **Training dataset**: The data on which the model will be trained. Most of your data will be here. Many developers estimate about 80%.
- **Test dataset**: The data withheld from the model during training, which is used to test how well your model will generalize to new data.

**Model Training Terminology**

*The model training algorithm iteratively updates a model's parameters to minimize some loss function.*

**Let's define those two terms:**

- Model parameters: Model parameters are settings or configurations the training algorithm can update to change how the model behaves. Depending on the context, you'll also hear other more specific terms used to describe model parameters

such as weights and biases. Weights, which are values that change as the model learns, are more specific to neural networks.

- Loss function: A loss function is used to codify the model's distance from this goal. For example, if you were trying to predict a number of snow cone sales based on the day's weather, you would care about making predictions that are as accurate as possible. So you might define a loss function to be "the average distance between your model's predicted number of snow cone sales and the correct number." You can see in the snow cone example this is the difference between the two purple dots.

**Putting it All Together**

The end-to-end training process is

- Feed the training data into the model.
- Compute the loss function on the results.
- Update the model parameters in a direction that reduces loss.

You continue to cycle through these steps until you reach a predefined stop condition. This might be based on a training time, the number of training cycles, or an even more intelligent or application-aware mechanism.

**Advice From the Experts**

Remember the following advice when training your model.

- Practitioners often use machine learning frameworks that already have working implementations of models and model training algorithms. You could implement these from scratch, but you probably won't need to do so unless you're developing new models or algorithms.

- Practitioners use a process called model selection to determine which model or models to use. The list of established models is constantly growing, and even seasoned machine learning practitioners may try many different types of models while solving a problem with machine learning.
- Hyperparameters are settings on the model which are not changed during training but can affect how quickly or how reliably the model trains, such as the number of clusters the model should identify.
- Be prepared to iterate.

*Pragmatic problem solving with machine learning is rarely an exact science, and you might have assumptions about your data or problem which turn out to be false. Don't get discouraged. Instead, foster a habit of trying new things, measuring success, and comparing results across iterations.*

**Extended Learning**

This information hasn't been covered in the above video but is provided for the advanced reader.

**Linear models**

One of the most common models covered in introductory coursework, linear models simply describe the relationship between a set of input numbers and a set of output numbers through a linear function (think of $y = mx + b$ or a line on a x vs y chart).

Classification tasks often use a strongly related logistic model, which adds an

additional transformation mapping the output of the linear function to the range [0, 1], interpreted as “probability of being in the target class.” Linear models are fast to train and give you a great baseline against which to compare more complex models. A lot of media buzz is given to more complex models, but for most new problems, consider starting with a simple model.

**Tree-based models**

Tree-based models are probably the second most common model type covered in introductory coursework. They learn to categorize or regress by building an extremely large structure of nested if/else blocks, splitting the world into different regions at each if/else block. Training determines exactly where these splits happen and what value is assigned at each leaf region.

For example, if you’re trying to determine if a light sensor is in sunlight or shadow, you might train tree of depth 1 with the final learned configuration being something like if (sensor_value > 0.698), then return 1; else return 0;. The tree-based model XGBoost is commonly used as an off-the-shelf implementation for this kind of model and includes enhancements beyond what is discussed here. Try tree-based models to quickly get a baseline before moving on to more complex models.

**Deep learning models**

Extremely popular and powerful, deep learning is a modern approach based

around a conceptual model of how the human brain functions. The model (also called a neural network) is composed of collections of neurons (very simple computational units) connected together by weights (mathematical representations of how much information to allow to flow from one neuron to the next). The process of training involves finding values for each weight.

Various neural network structures have been determined for modeling different kinds of problems or processing different kinds of data.

A short (but not complete!) list of noteworthy examples includes:

- **FFNN**: The most straightforward way of structuring a neural network, the Feed Forward Neural Network (FFNN) structures neurons in a series of layers, with each neuron in a layer containing weights to all neurons in the previous layer.
- **CNN**: Convolutional Neural Networks (CNN) represent nested filters over grid-organized data. They are by far the most commonly used type of model when processing images.
- **RNN/LSTM**: Recurrent Neural Networks (RNN) and the related Long Short-Term Memory (LSTM) model types are structured to effectively represent for loops in traditional computing, collecting state while iterating over some object. They can be used for processing sequences of data.
- **Transformer**: A more modern replacement for RNN/LSTMs, the transformer architecture enables training over larger datasets involving sequences of data.

**Machine Learning Using Python Libraries**

For more classical models (linear, tree-based) as well as a set of common ML-related tools, take a look at scikit-learn. The web documentation for this library is also organized for those getting familiar with space and can be a great place to get familiar with some extremely useful tools and techniques.

For deep learning, mxnet, tensorflow, andpytorch are the three most common libraries. For the purposes of the majority of machine learning needs, each of these is feature-paired and equivalent.

**Terminology**

Hyperparameters are settings on the model which are not changed during training but can affect how quickly or how reliably the model trains, such as the number of clusters the model should identify.

A **loss function** is used to codify the model's distance from this goal

**Training dataset**: The data on which the model will be trained. Most of your data will be here.

**Test dataset**: The data withheld from the model during training, which is used to test how well your model will generalize to new data.

Model parameters are settings or configurations the training algorithm can update to change how the model behaves.

**Additional reading**

- The Wikipedia entry on the bias-variance trade-off can help you understand more about this common machine learning concept.
- In this AWS Machine Learning blog post, you can see how to train

a machine-learning algorithm to predict the impact of weather on air quality using Amazon SageMaker.

# Step Four: Model Evaluation

After you have collected your data and trained a model, you can start to evaluate how well your model is performing. The metrics used for evaluation are likely to be very specific to the problem you have defined. As you grow in your understanding of machine learning, you will be able to explore a wide variety of metrics that can enable you to evaluate effectively.

**Using Model Accuracy**

Model accuracy is a fairly common evaluation metric. Accuracy is the fraction of predictions a model gets right.

Here's an example:

Petal length to determine species

Imagine that you built a model to identify a flower as one of two common species based on measurable details like petal length. You want to know

how often your model predicts the correct species. This would require you to look at your model's accuracy.

### Extended Learning

This information hasn't been covered in the above video but is provided for the advanced reader.

### Using Log Loss

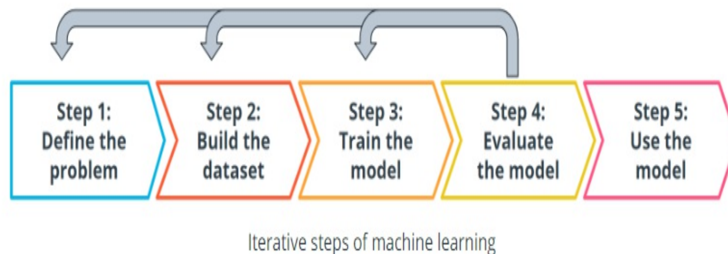
Log loss seeks to calculate how uncertain your model is about the predictions it is generating. In this context, uncertainty refers to how likely a model thinks the predictions being generated are to be correct.

For example, let's say you're trying to predict how likely a customer is to buy either a jacket or t-shirt.

Log loss could be used to understand your model's uncertainty about a given prediction. In a single instance, your model could predict with 5% certainty that a customer is going to buy a t-shirt. In another instance, your model could predict with 80% certainty that a customer is going to buy a t-shirt. Log loss enables you to measure how strongly the model believes that its prediction is accurate.

In both cases, the model predicts that a customer will buy a t-shirt, but the model's certainty about that prediction can change.

**Remember: This Process is Iterative**



Iterative steps of machine learning

Every step we have gone through is highly iterative and can be changed or re-scoped during the course of a project. At each step, you might find that you need to go back and reevaluate some assumptions you had in previous steps. Don't worry! This ambiguity is normal.

**Terminology**

**Log loss** seeks to calculate how uncertain your model is about the predictions it is generating.

**Model Accuracy** is the fraction of predictions a model gets right.

**Additional reading**

The tools used for model evaluation are often tailored to a specific use case, so it's difficult to generalize rules for choosing them. The following articles provide use cases and examples of specific metrics in use.

- This healthcare-based example, which automates the prediction

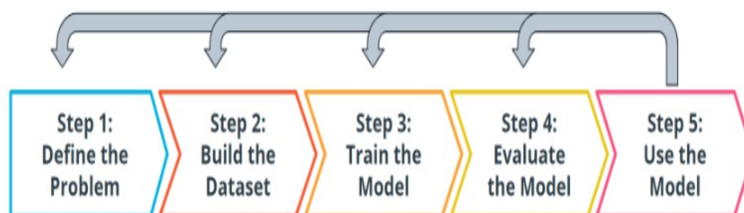
of spinal pathology conditions, demonstrates how important it is to avoid false positive and false negative predictions using the tree-based xgboost model.
- The popular open-source library sklearn provides information about common metrics and how to use them.
- This entry from the AWS Machine Learning blog demonstrates the importance of choosing the correct model evaluation metrics for making accurate energy consumption estimates using Amazon Forecast.

# Step Five: Model Inference

Congratulations! You're ready to deploy your model.

Once you have trained your model, have evaluated its effectiveness, and are satisfied with the results, you're ready to generate predictions on real-world problems using unseen data in the field. In machine learning, this process is often called inference.

**Iterative Process**




Iteration of the entire machine learning process

Even after you deploy your model, you're always monitoring to make sure your model is producing the kinds of results that you expect. There may be

times when you reinvestigate the data, modify some of the parameters in your model training algorithm, or even change the model type used for training.

# Introduction to Examples

Through the remainder of the lesson, we will be walking through 3 different case study examples of machine learning tasks actually solving problems in the real world.

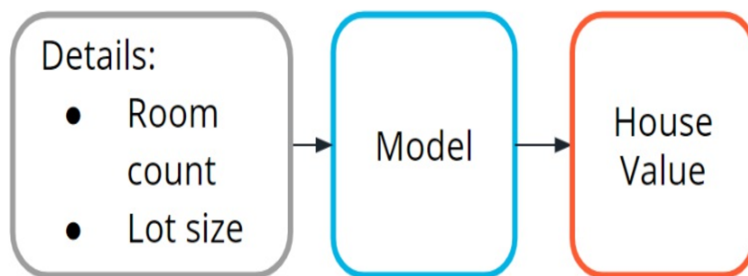
- **Supervised learning**
  - Using machine learning to predict housing prices in a neighborhood based on lot size and number of bedrooms
- **Unsupervised learning**
  - Using machine learning to isolate micro-genres of books by analyzing the wording on the back cover description.
- **Deep neural network**
  - While this type of task is beyond the scope of this lesson, we wanted to show you the power and versatility of modern machine learning. You will see how it can be used to analyze raw images from lab video footage from security cameras, trying to detect chemical spills.

# Example One: House Price Prediction

House price prediction is one of the most common examples used to introduce machine learning.

Traditionally, real estate appraisers use many quantifiable details about a home (such as number of rooms, lot size, and year of construction) to help them estimate the value of a house.

You detect this relationship and believe that you could use machine learning to predict home prices.

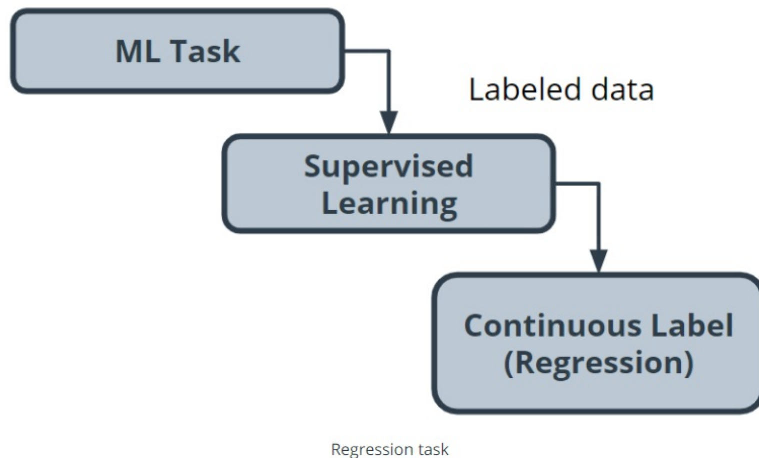



Machine language models to determine house values

**Step One: Define the Problem**

*Can we estimate the price of a house based on lot size or the number of bedrooms?*

You access the sale prices for recently sold homes or have them appraised. Since you have this data, this is a supervised learning task. You want to predict a continuous numeric value, so this task is also a regression task.




Regression task

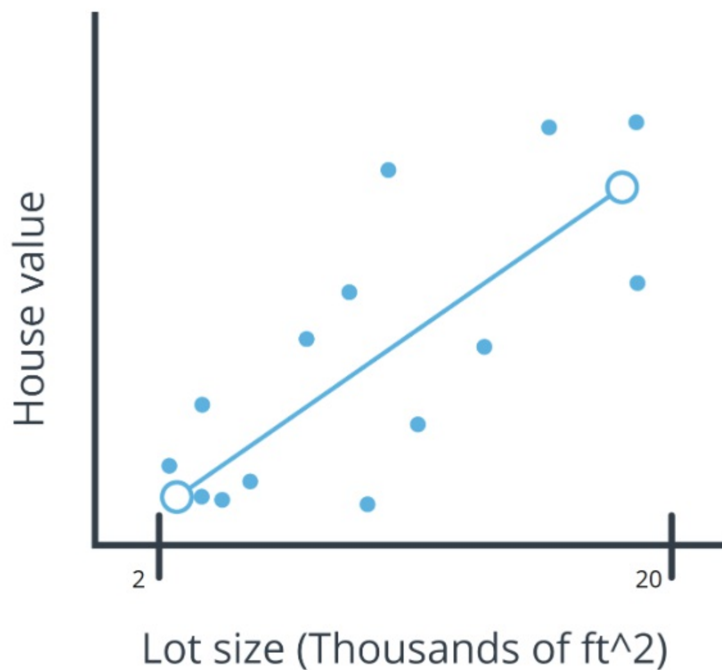
**Step Two: Building a Dataset**

- **Data collection**: You collect numerous examples of homes sold in your neighborhood within the past year, and pay a real estate appraiser to appraise the homes whose selling price is not known.
- **Data exploration**: You confirm that all of your data is numerical because most machine learning models operate on sequences of numbers. If there is textual data, you need to transform it into numbers. You'll see this in the next example.
- **Data cleaning**: Look for things such as missing information or outliers, such as the 10-room mansion. Several techniques can be used to handle outliers, but you can also just remove those from your dataset.

| No of Rooms | Lot Size ($ft^2$) | House Value ($) |
|---|---|---|
| 4 | 10,454 | 339,900 |
| | 9,147 | 239,000 |
| 3 | 10,890 | 250,000 |
| | ~~25,877~~ | ~~877,000~~ |

*Data Cleaning: removing outlier values*

- Data visualization: You can plot home values against each of your input variables to look for trends in your data. In the following chart, you see that when lot size increases, the house value increases.




Regression line of a model

**Step Three: Model Training**

Prior to actually training your model, you need to split your data. The standard practice is to put 80% of your dataset into a training dataset and 20% into a test dataset.

## Linear model selection

As you see in the preceding chart, when lot size increases, home values

increase too. This relationship is simple enough that a linear model can be used to represent this relationship.

A linear model across a single input variable can be represented as a line. It becomes a plane for two variables, and then a hyperplane for more than two variables. The intuition, as a line with a constant slope, doesn't change.

## Using a Python library

The Python scikit-learn library has tools that can handle the implementation of the model training algorithm for you.

### Step Four: Evaluation

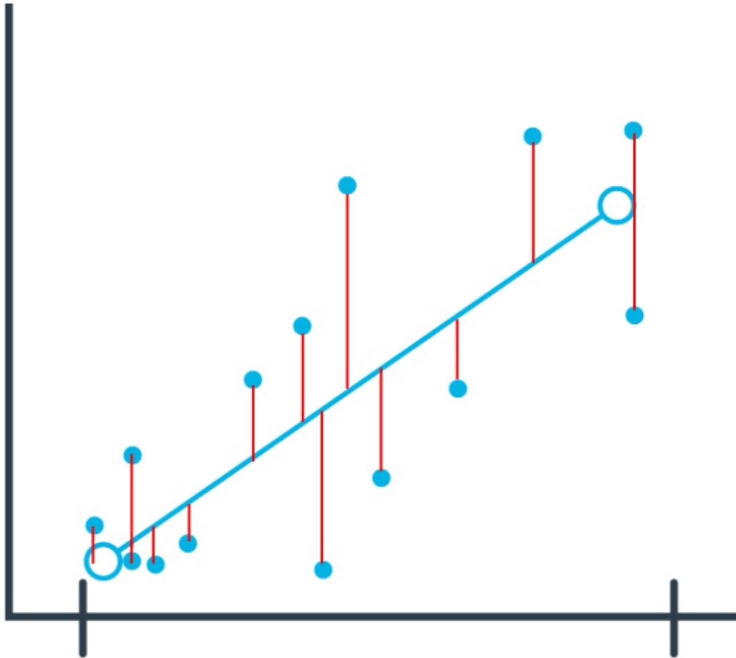
One of the most common evaluation metrics in a regression scenario is called root mean square or RMS. The math is beyond the scope of this lesson, but RMS can be thought of roughly as the "average error" across your test dataset, so you want this value to be low.

$$RMS = \sqrt{\frac{1}{n}\sum_{i} x_i^2}$$

The math behind RMS

In the following chart, you can see where the data points are in relation to the blue line. You want the data points to be as close to the "average" line as possible, which would mean less net error.

You compute the root mean square between your model's prediction for a data point in your test dataset and the true value from your data. This actual calculation is beyond the scope of this lesson, but it's good to understand the process at a high level.



**Interpreting Results**

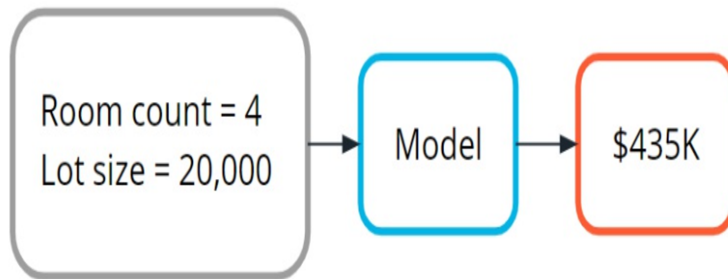
In general, as your model improves, you see a better RMS result. You may still not be confident about whether the specific value you've computed is good or bad.

Many machine learning engineers manually count how many predictions were off by a threshold (for example, $50,000 in this house pricing problem) to help determine and verify the model's accuracy.

#### Step Five: Inference: Try out your model

Now you are ready to put your model into action. As you can see in the

following image, this means seeing how well it predicts with new data not seen during model training.




**Terminology**

- **Continuous**: Floating-point values with an infinite range of possible values. The opposite of categorical or discrete values, which take on a limited number of possible values.
- **Hyperplane**: A mathematical term for a surface that contains more than two planes.
- **Plane**: A mathematical term for a flat surface (like a piece of paper) on which two points can be joined by a straight line.
- Regression: A common task in supervised machine learning.

**Additional reading**

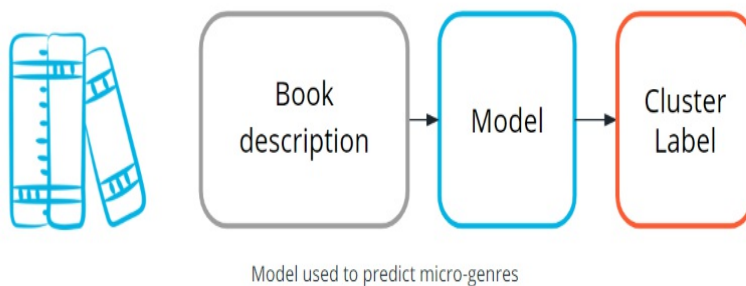
The Machine Learning Mastery blog is a fantastic resource for learning more about machine learning. The following example blog posts dive deeper into training regression-based machine learning models.

- How to Develop Ridge Regression Models in Python offers another approach to solving the problem in the example from this lesson.
- Regression is a popular machine learning task, and you can use several different model evaluation metrics with it.

# Example Two: Book Genre Exploration

In this video, you saw how the machine learning process can be applied to an unsupervised machine learning task that uses book description text to identify different micro-genres.

#### Step One: Define the Problem




Model used to predict micro-genres

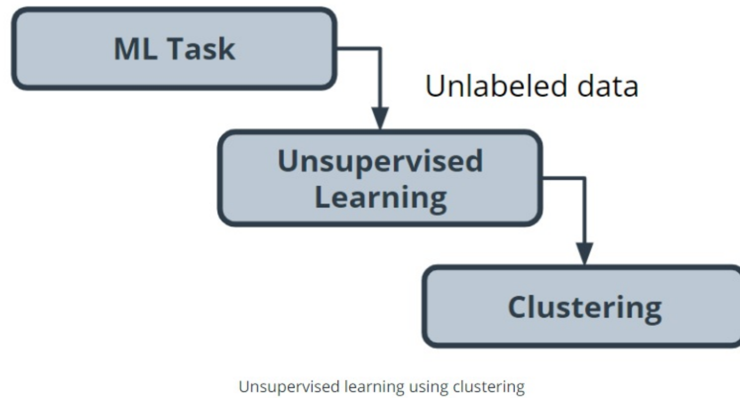
*Find clusters of similar books based on the presence of common words in the book descriptions.*

You do editorial work for a book recommendation company, and you want to write an article on the largest book trends of the year. You believe that a trend called "micro-genres" exists, and you have confidence that you can use the book description text to identify these micro-genres.

By using an unsupervised machine learning technique called clustering, you can test your hypothesis that the book description text can be used to identify these "hidden" micro-genres.

Earlier in this lesson, you were introduced to the idea of unsupervised

learning. This machine learning task is especially useful when your data is not labeled.



Unsupervised learning using clustering

#### Step Two: Build your Dataset

To test the hypothesis, you gather book description text for 800 romance books published in the current year.

Data exploration, cleaning and preprocessing

For this project, you believe capitalization and verb tense will not matter, and therefore you remove capitals and convert all verbs to the same tense using a Python library built for processing human language. You also remove punctuation and words you don't think have useful meaning, like 'a' and 'the'. The machine learning community refers to these words as stop words.

Before you can train the model, you need to do some data preprocessing, called data vectorization, to convert text into numbers.

You transform this book description text into what is called a bag of wordsrepresentation shown in the following image so that it is understandable by machine learning models.

How the bag of words representation works is beyond the scope of this course. If you are interested in learning more, see the **Additional Reading** section at the bottom of the section.

**"Little did he know, she was secretly a vampire."**

**['little', 'does', 'he', 'know', 'she', 'is', 'secretly', 'vampire']**

Bag of Words

[0, 0, 1, 0, 1, ...]

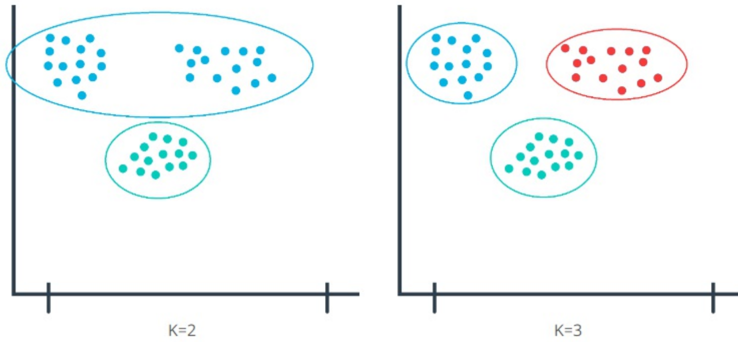
### Step Three: Train the Model

Now you are ready to train your model.

You pick a common cluster-finding model called k-means. In this model, you can change a model parameter, k, to be equal to how many clusters the model will try to find in your dataset.

Your data is unlabeled: you don't how many microgenres might exist. So you train your model multiple times using different values for k each time.

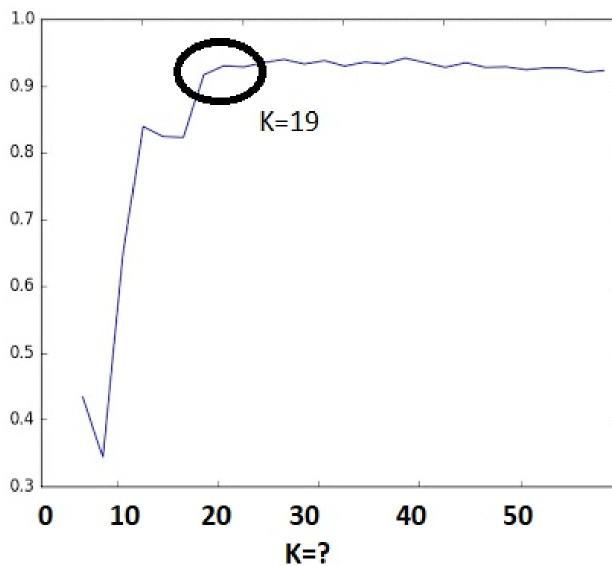
What does this even mean? In the following graphs, you can see examples of when k=2 and when k=3.




During the model evaluation phase, you plan on using a metric to find which value for k is most appropriate.

#### Step Four: Model Evaluation

In machine learning, numerous statistical metrics or methods are available to evaluate a model. In this use case, the silhouette coefficient is a good choice. This metric describes how well your data was clustered by the model. To find the optimal number of clusters, you plot the silhouette coefficient as shown in the following image below. You find the optimal value is when k=19.




Optimum number (k=19) of clusters

Often, machine learning practitioners do a manual evaluation of the model's findings.

You find one cluster that contains a large collection of books you can categorize as “paranormal teen romance.” This trend is known in your industry, and therefore you feel somewhat confident in your machine learning approach. You don’t know if every cluster is going to be as cohesive as this, but you decide to use this model to see if you can find anything interesting about which to write an article.

#### Step Five: Inference (Use the Model)

As you inspect the different clusters found when k=19, you find a surprisingly large cluster of books. Here's an example from fictionalized cluster #7.

| Cluster Label | Book Description |
|---|---|
| 7 | “Susan's crush just moved away..” |
| 7 | “Can Alice and Bob keep their relationship together three hundred miles apart?” |
| 7 | “When Hank's fiance George got offered a new job in New York...” |

Clustered data

As you inspect the preceding table, you can see that most of these text snippets are indicating that the characters are in some kind of long-distance relationship. You see a few other self-consistent clusters and feel you now have enough useful data to begin writing an article on unexpected modern

romance microgenres.

## Terminology

- **Bag of words**: A technique used to extract features from the text. It counts how many times a word appears in a document (corpus), and then transforms that information into a dataset.
- **Data vectorization**: A process that converts non-numeric data into a numerical format so that it can be used by a machine learning model.
- **Silhouette coefficient**: A score from -1 to 1 describing the clusters found during modeling. A score near zero indicates overlapping clusters, and scores less than zero indicate data points assigned to incorrect clusters. A score approaching 1 indicates successful identification of discrete non-overlapping clusters.
- **Stop words**: A list of words removed by natural language processing tools when building your dataset. There is no single universal list of stop words used by all-natural language processing tools.

## Additional reading

Machine Learning Mastery is a great resource for finding examples of machine learning projects.

The How to Develop a Deep Learning Bag-of-Words Model for Sentiment Analysis (Text Classification) blog post provides an example using a bag of words–based approach pair with a deep learning model.

# Example Three: Spill Detection from Video

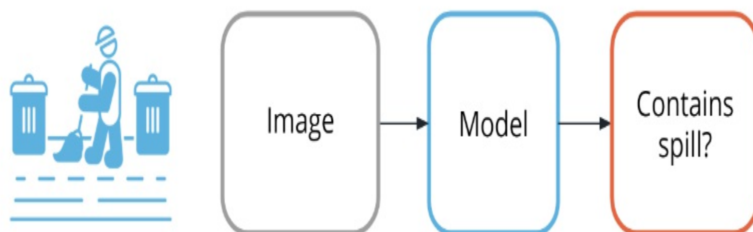
In the previous two examples, we used classical methods like linear models and k-means to solve machine learning tasks. In this example, we'll use a more modern model type.

*Note*: This example uses a neural network. The algorithm for how a neural network works is beyond the scope of this lesson. However, there is still value in seeing how machine learning applies in this case.

#### Step One: Defining the Problem

Imagine you run a company that offers specialized on-site janitorial services. A client, an industrial chemical plant, requires a fast response for spills and other health hazards. You realize if you could automatically detect spills using the plant's surveillance system, you could mobilize your janitorial team faster.

**Machine learning could be a valuable tool to solve this problem.**




Detecting spills with machine learning

#### Step Two: Model Training (and selection)

This task is a supervised classification task, as shown in the following image. As shown in the image above, your goal will be to predict if each image belongs to one of the following classes:

- Contains spill
- Does not contain spill

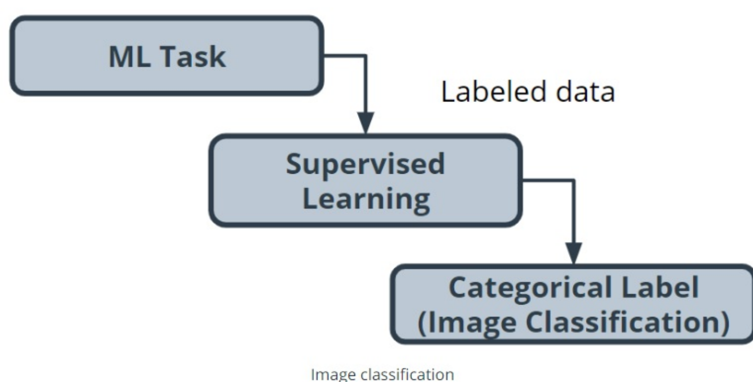


Image classification

**Step Two: Building a Dataset**

- **Collecting**
  - Using historical data, as well as safely staged spills, you quickly build a collection of images that contain both spills and non-spills in multiple lighting conditions and environments.
- **Exploring and cleaning**
  - You go through all the photos to ensure the spill is clearly in the shot. There are Python tools and other techniques available to improve image quality, which you can use later if you determine a need to iterate.
- Data vectorization (converting to numbers)
  - Many models require numerical data, so all your image data needs to be transformed into a numerical format. Python tools can help you do this automatically.
  - In the following image, you can see how each pixel in the image on the left can be represented in the image on the right by a number between 0 and 1, with 0 being completely black and 1 being completely white.

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1
1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1
1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1
1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1
0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1
0 0 0 0 0 0 1 1 1 1 1 0 1 1 1 1
0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Chemical spill image

Numeric representation of chemical spill image

**Split the data**

- You split your image data into a training dataset and a test dataset.

**Step Three: Model Training**

Traditionally, solving this problem would require hand-engineering features on top of the underlying pixels (for example, locations of prominent edges and corners in the image), and then training a model on these features.

Today, deep neural networks are the most common tool used for solving this kind of problem. Many deep neural network models are structured to learn the features on top of the underlying pixels so you don't have to learn them. You'll have a chance to take a deeper look at this in the next lesson, so we'll keep things high-level for now.

**CNN (convolutional neural network)**

Neural networks are beyond the scope of this lesson, but you can think of them as a collection of very simple models connected together. These simple models are called neurons, and the connections between these

models are trainable model parameters called weights.

Convolutional neural networks are a special type of neural network particularly good at processing images.

#### Step Four: Model Evaluation

As you saw in the last example, there are many different statistical metrics you can use to evaluate your model. As you gain more experience in machine learning, you will learn how to research which metrics can help you evaluate your model most effectively. Here's a list of common metrics:

| Accuracy | False positive rate | Precision |
|---|---|---|
| Confusion matrix | False negative rate | Recall |
| F1 Score | Log Loss | ROC curve |
| | Negative predictive value | Specificity |

In cases such as this, accuracy might not be the best evaluation mechanism.

**Why not**? You realize the model will see the 'Does not contain spill' class almost all the time, so any model that just predicts “no-spill” most of the time will seem pretty accurate.

*What you really care about is an evaluation tool that rarely misses a real spill.*

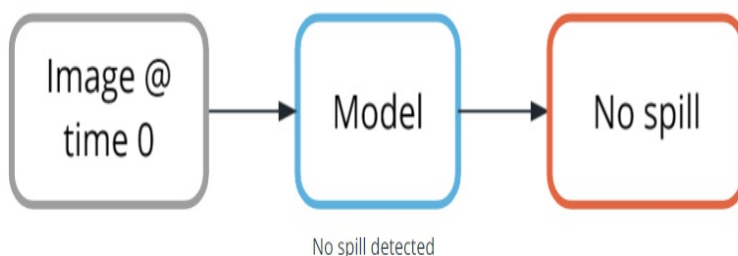
After doing some internet sleuthing, you realize this is a common problem and that Precision and Recall will be effective. You can think of precision as answering the question, "Of all predictions of a spill, how many were right?" and recall as answering the question, "Of all actual spills, how many did we detect?"

Manual evaluation plays an important role. You are unsure if your staged spills are sufficiently realistic compared to actual spills. To get a better sense how well your model performs with actual spills, you find additional examples from historical records. This allows you to confirm that your model is performing satisfactorily.

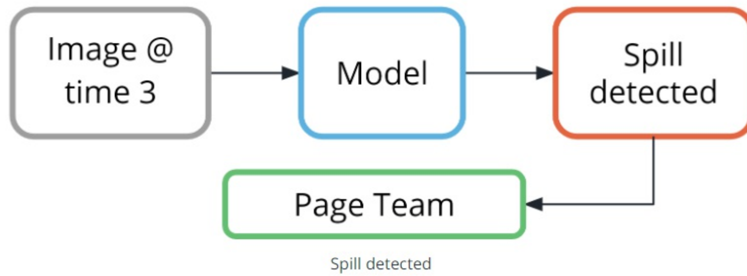
**Step Five: Model Inference**

The model can be deployed on a system that enables you to run machine learning workloads such as AWS Panorama.

Thankfully, most of the time, the results will be from the class 'Does not contain spill.




But, when the class **'Contains spill'** is detected, a simple paging system could alert the team to respond.




### Terminology

**Convolutional neural networks(CNN)** are a special type of neural network particularly good at processing images.

**Neural networks**: a collection of very simple models connected together.

- These simple models are called neurons
- the connections between these models are trainable model parameters called weights.

### Additional reading

As you continue your machine learning journey, you will start to recognize problems that are excellent candidates for machine learning.

The AWS Machine Learning Blog is a great resource for finding more examples of machine learning projects.

- In the Protecting people from hazardous areas through virtual boundaries with Computer Vision blog post, you can see a more detailed example of the deep learning process described in this lesson.

### Lesson Review

Congratulations on making it through the lesson. Let's review what you learning.

- In the first part of the lesson, we talked about what machine learning actually is, introduced you to some of the most common terms and ideas used in machine learning, and identified the common components involved in machine learning projects.
- We learned that machine learning involves using trained models to generate predictions and detect patterns from data. We looked behind the scenes to see what is really happening. We also broke down the different steps or tasks involved in machine learning.
- We looked at three machine learning examples to demonstrate how each works to solve real-world situations.
    - A supervised learning task in which you used machine learning to predict housing prices for homes in your neighborhood, based on the lot size and the number of bedrooms.
    - An unsupervised learning task in which you used machine learning to find interesting collections of books in a book dataset, based on the descriptive words in the book description text.
    - Using a deep neural network to detect chemical spills in a lab from video and images.

**Learning Objectives**

If you read through studied the images, and completed all the quizzes, then you should've mastered the learning objectives for the lesson. You should recognize all of these by now. Please read through and check off each as you go through them.

- Differentiate between supervised learning and unsupervised learning

- Identify problems that can be solved with machine learning
- Describe commonly used algorithms including linear regression, logistic regression, and k-means
- Describe how model training and testing works
- Evaluate the performance of a machine learning model using matrices.

# CHAPTER THREE

# Machine Learning with AWS

## Why AWS?

The AWS machine learning mission is to put machine learning in the hands of every developer.

- AWS offers the broadest and deepest set of artificial intelligence (AI) and machine learning (ML) services with unmatched flexibility.
- You can accelerate your adoption of machine learning with AWS SageMaker. Models that previously took months to build and required specialized expertise can now be built in weeks or even days.
- AWS offers the most comprehensive cloud offering optimized for machine learning.
- More machine learning happens at AWS than anywhere else.

## AWS Machine Learning offerings

When it comes to AI services, you don't necessarily need to code ML solutions from scratch. AWS pre-trained AI services provide ready-made intelligence for your applications and workflows. You can just take the services and apply them from your own use cases.

Amazon has different health AI offerings, which requires no machine learning knowledge. One example of Amazon health AI offering is Amazon Transcribe Medical, which turns medical speech into text. Conversations

between health care providers and patients provide the foundations of a patient's diagnosis and treatment. Its very important that they information is accurate. Amazon Transcribe Medical is an automatic speech recognition, short-form ASR service.

**AWS AI services**

By using AWS pre-trained AI services, you can apply ready-made intelligence to a wide range of applications such as personalized recommendations, modernizing your contact center, improving safety and security, and increasing customer engagement.

**Industry-specific solutions**

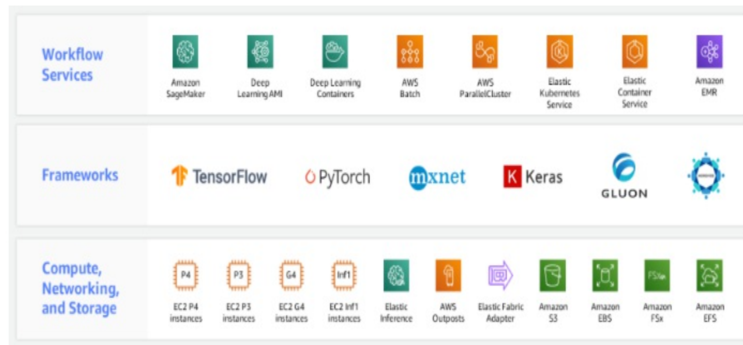
With no knowledge in machine learning needed, add intelligence to a wide range of applications in different industries including healthcare and manufacturing.

**AWS Machine Learning services**

With AWS, you can build, train, and deploy your models fast. Amazon SageMaker is a fully managed service that removes complexity from ML workflows so every developer and data scientist can deploy machine learning for a wide range of use cases.

**ML infrastructure and frameworks**

AWS Workflow services make it easier for you to manage and scale your underlying ML infrastructure.




ML infrastructure and frameworks

**Getting started**

In addition to educational resources such as AWS Training and Certification, AWS has created a portfolio of educational devices to help put new machine learning techniques into the hands of developers in unique and fun ways, with **AWS DeepRacer** and **AWS DeepComposer**.

**AWS DeepRacer**: An autonomous race car designed to test reinforcement learning models by racing on a physical track

**AWS DeepComposer**: A composing device powered by generative AI that creates a melody that transforms into a completely original song

**AWS ML Training and Certification**: Curriculum used to train Amazon developers.

**AWS DeepRacer**
Race in global leagues with a fully autonomous race car and learn **reinforcement learning**.

**AWS DeepComposer**
Learn **generative artificial intelligence (AI)** by collaborating with AI to create music.

AWS educational devices

# AWS Account Requirements

**An AWS account is required**

To complete the exercises in this course, you need an **AWS Account ID**.

However, you are not required to complete the exercises in this lesson in order to graduate from the foundations course. The exercises included in this lesson are **OPTIONAL ONLY**, so if you cannot create an AWS account, please simply read the exercise modules to understand how these tools are meant to work.

To set up a new AWS Account ID, follow the directions in How do I create and activate a new Amazon Web Services account?

You are required to provide a payment method when you create the account. To learn about which services are available at no cost, see the AWS Free Tier documentation.

**Will these exercises cost anything?**

This lesson contains many demos and exercises. You **do not** need to purchase any AWS devices to complete the lesson. However, please carefully read the following list of AWS services you may need in order to follow the demos and complete the exercises.

**Train your reinforcement learning model with AWS DeepRacer**

- To get started with AWS DeepRacer, you receive **10 free hours** to train or evaluate models and **5GB of free storage** during your first month. This is enough to train your first time-trial model, evaluate it, tune it, and then enter it into the AWS DeepRacer League. **This offer is valid for 30 days after you have used the service for the first time.**
- Beyond 10 hours of training and evaluation, you pay for training, evaluating, and storing your machine learning models. Charges are based on the amount of time you train and evaluate a new model and the size of the model stored. To learn more about AWS DeepRacer pricing, see the AWS DeepRacer Pricing

**Generate music using AWS DeepComposer**

- To get started, **AWS DeepComposer** provides a 12-month Free Tier for first-time users. With the Free Tier, you can perform up to 500 inference jobs translating to 500 pieces of music using the **AWS DeepComposer** Music studio. You can use one of these instances to complete the exercise at no cost. To learn more about **AWS DeepComposer** costs, see the **AWS DeepComposer** Pricing.

**Build a custom generative AI model (GAN) using Amazon SageMaker**

**(optional)**

- Amazon SageMaker is a separate service and has its own service pricing and billing tier. To train the custom generative AI model, the instructor uses an instance type that is not covered in the Amazon SageMaker free tier. If you want to code along with the instructor and train your own custom model, you may incur a cost. Please note, that creating your own custom model is completely optional. You are not required to do this exercise to complete the course. To learn more about SageMaker costs, see the Amazon SageMaker Pricing.

**Please confirm that you have taken the necessary steps to complete this lesson.**

- I have an AWS account ID
- I understand that to finish this course, I do not need to purchase AWS DeepRacer or AWS DeepComposer devices.
- I understand that for 30 days after I first use AWS DeepRacer, I have 10 free hours of model training and evaluation to finish the demo and exercise in the AWS DeepRacer section.
- I understand that the AWS Free Tier provides me with **500 AWS DeepComposer inferences jobs** (500 pieces of music) at no cost.
- I understand that the “Build a custom GAN” demo is optional and I can watch the demo rather that completing it on my own, as I may incur a charge for completing the exercise.
- I understand that I may liable for any charge incurred on my AWS account.

### Reinforcement Learning and Its Applications

This section introduces you to a type of machine learning (ML) called

reinforcement learning (RL). You'll hear about its real-world applications and learn basic concepts using AWS DeepRacer as an example. By the end of the section, you will be able to create, train, and evaluate a reinforcement learning model in the AWS DeepRacer console.

In reinforcement learning (RL), an agent is trained to achieve a goal based on the feedback it receives as it interacts with an environment. It collects a number as a reward for each action it takes. Actions that help the agent achieve its goal are incentivized with higher numbers. Unhelpful actions result in a low reward or no reward.

With a learning objective of maximizing total cumulative reward, over time, the agent learns, through trial and error, to map gainful actions to situations. The better trained the agent, the more efficiently it chooses actions that accomplish its goal.

**Reinforcement Learning Applications**

Reinforcement learning is used in a variety of fields to solve real-world problems. It’s particularly useful for addressing sequential problems with long-term goals. Let’s take a look at some examples.

- RL is great at **playing games**:
  - Go (board game) was mastered by the AlphaGo Zero software.
  - Atari classic video games are commonly used as a learning tool for creating and testing RL software.
  - StarCraft II, the real-time strategy video game, was mastered by the AlphaStar software.
- RL is used in **video game level design**:

    - Video game level design determines how complex each stage of a game is and directly affects how boring, frustrating, or fun it is to play that game.
    - Video game companies create an agent that plays the game over and over again to collect data that can be visualized on graphs.
    - This visual data gives designers a quick way to assess how easy or difficult it is for a player to make progress, which enables them to find that “just right” balance between boredom and frustration faster.
- RL is used in **wind energy optimization**:
    - RL models can also be used to power robotics in physical devices.
    - When multiple turbines work together in a wind farm, the turbines in the front, which receive the wind first, can cause poor wind conditions for the turbines behind them. This is called wake turbulence and it reduces the amount of energy that is captured and converted into electrical power.
    - Wind energy organizations around the world use reinforcement learning to test solutions. Their models respond to changing wind conditions by changing the angle of the turbine blades. When the upstream turbines slow down it helps the downstream turbines capture more energy.
- Other examples of real-world RL include:
    - **Industrial robotics**
    - **Fraud detection**
    - **Stock trading**
    - **Autonomous driving**

Some examples of real-world RL include: Industrial robotics, fraud detection, stock trading, and autonomous driving

**New Terms**

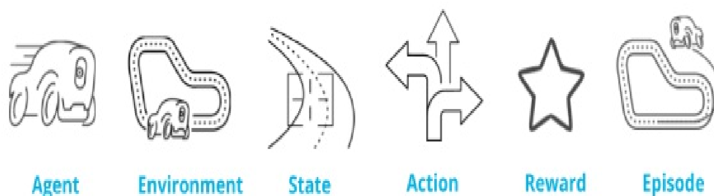
- **Agent**: The piece of software you are training is called an agent. It makes decisions in an environment to reach a goal.
- **Environment**: The environment is the surrounding area with which the agent interacts.
- **Reward**: Feedback is given to an agent for each action it takes in a given state. This feedback is a numerical reward.
- **Action**: For every state, an agent needs to take an action toward achieving its goal.

# Reinforcement Learning with AWS DeepRacer

**Reinforcement Learning Concepts**

In this section, we'll learn some basic reinforcement learning terms and concepts using **AWS DeepRacer** as an example.

This section introduces six basic reinforcement learning terms and provides an example for each in the context of AWS DeepRacer.




*Basic RL terms: Agent, environment, state, action, reward, and episode*

**Agent**

- The piece of software you are training is called an agent.
- It makes decisions in an environment to reach a goal.
- In AWS DeepRacer, the agent is the AWS DeepRacer car and its goal is to finish * laps around the track as fast as it can while, in some cases, avoiding obstacles.

**Environment**

- The environment is the surrounding area within which our agent interacts.
- For AWS DeepRacer, this is a track in our simulator or in real life.

**State**

- The state is defined by the current position within the environment that is visible, or known, to an agent.
- In AWS DeepRacer's case, each state is an image captured by its camera.
- The car's initial state is the starting line of the track and its terminal state is when the car finishes a lap, bumps into an obstacle, or drives off the track.

**Action**

- For every state, an agent needs to take an action toward achieving its goal.
- An AWS DeepRacer car approaching a turn can choose to accelerate or brake and turn left, right, or go straight.

**Reward**

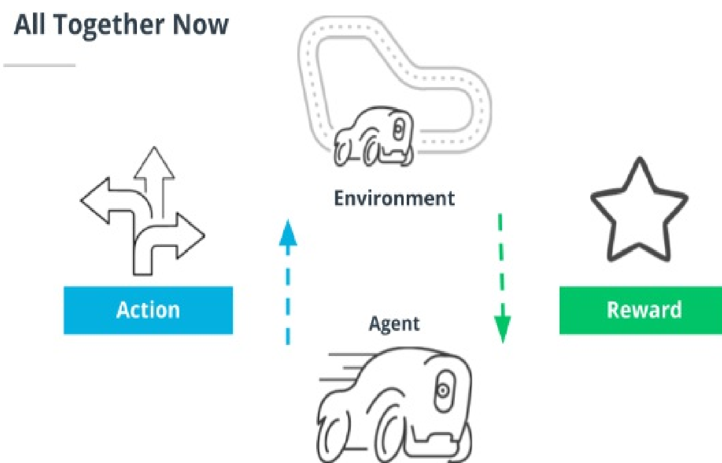
- Feedback is given to an agent for each action it takes in a given state.

- This feedback is a numerical reward.
- A reward function is an incentive plan that assigns scores as rewards to different zones on the track.

**Episode**

- An episode represents a period of trial and error when an agent makes decisions and gets feedback from its environment.
- For AWS DeepRacer, an episode begins at the initial state, when the car leaves the starting position, and ends at the terminal state, when it finishes a lap, bumps into an obstacle, or drives off the track.

In a reinforcement learning model, an agent learns in an interactive real-time environment by trial and error using feedback from its own actions. Feedback is given in the form of rewards.




*In a reinforcement learning model, an agent learns in an interactive real-time environment by trial and error using feedback from its own actions. Feedback is given in the form of rewards.*

# Putting Your Spin on AWS DeepRacer:

**The Practitioner's Role in RL**

AWS DeepRacer may be autonomous, but you still have an important role to play in the success of your model. In this section, we introduce the training algorithm, action space, hyperparameters, and reward function and discuss how your ideas make a difference.

- *An algorithm* is a set of instructions that tells a computer what to do. ML is special because it enables computers to learn without being explicitly programmed to do so.
- *The training algorithm* defines your model's learning objective, which is to maximize total cumulative reward. Different algorithms have different strategies for going about this.
  - A soft actor critic (SAC) embraces exploration and is data-efficient, but can lack stability.
  - A proximal policy optimization (PPO) is stable but data-hungry.
- An action space is the set of all valid actions, or choices, available to an agent as it interacts with an environment.

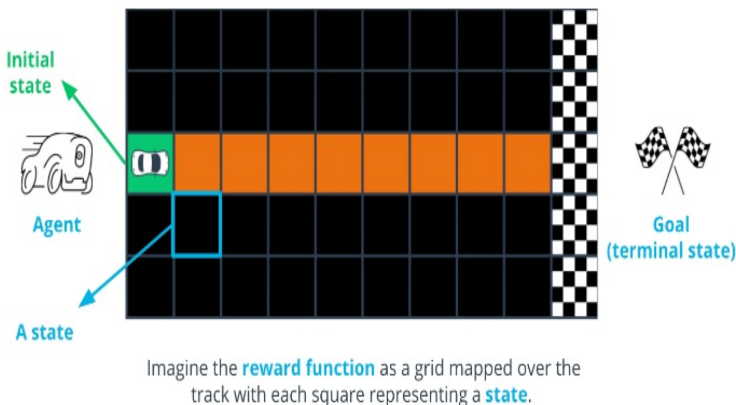
Discrete action space represents all of an agent's possible actions for each state in a finite set of steering angle and throttle value combinations.

Continuous action space allows the agent to select an action from a range of values that you define for each sta te.

- Hyperparameters are variables that control the performance of your agent during training. There is a variety of different categories with which to experiment. Change the values to increase or decrease the influence of different parts of your model.
  - For example, the learning rate is a hyperparameter that controls how many new experiences are counted in learning at each step. A higher learning rate results in faster training but may reduce the model's quality.
- The reward function's purpose is to encourage the agent to reach its goal. Figuring out how to reward which actions is one of your most important jobs.

**Putting Reinforcement Learning into Action with AWS DeepRacer**

This video put the concepts we've learned into action by imagining the reward function as a grid mapped over the race track in AWS DeepRacer’s training environment, and visualizing it as metrics plotted on a graph. It also introduced the trade-off between exploration and exploitation, an important challenge unique to this type of machine learning.

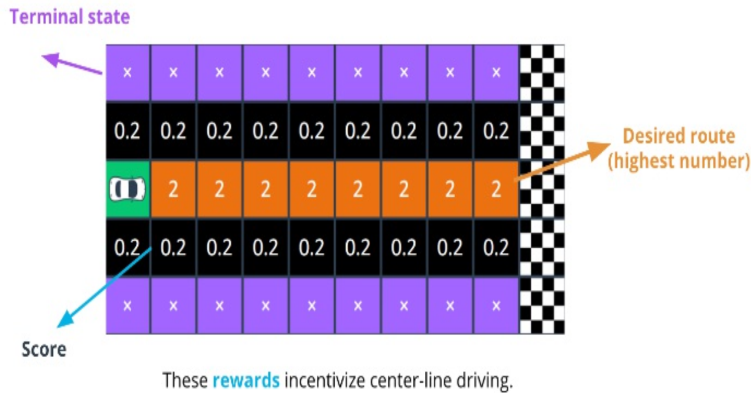


Imagine the reward function as a grid mapped over the track with each square representing a state.

Each square is a state. The green square is the starting position, or initial state, and the finish line is the goal, or terminal state.

Key points to remember about **reward functions**:

- Each state on the grid is assigned a score by your reward function. You incentivize behavior that supports your car’s goal of completing fast laps by giving the highest numbers to the parts of the track on which you want it to drive.
- The reward function is the actual code you'll write to help your agent determine if the action it just took was good or bad, and how good or bad it was.




The squares containing exes are the track edges and defined as terminal states, which tell your car it has gone off track.

Key points to remember about **exploration versus exploitation:**

- When a car first starts out, it explores by wandering in random directions. However, the more training an agent gets, the more it learns about an environment. This experience helps it become more confident about the actions it chooses.
- Exploitation means the car begins to exploit or use information from previous experiences to help it reach its goal. Different training algorithms utilize exploration and exploitation differently.

Key points to remember about the **reward graph**:

- While training your car in the AWS DeepRacer console, your training metrics are displayed on a reward graph.
- Plotting the total reward from each episode allows you to see how the model performs over time. The more reward your car gets, the better your model performs.

Key points to remember about **AWS DeepRacer**:

- AWS DeepRacer is a combination of a physical car and a virtual simulator in the AWS Console, the AWS DeepRacer League, and

community races.
- An AWS DeepRacer device is not required to start learning: you can start now in the AWS console. The 3D simulator in the AWS console is where training and evaluation take place.

**New Terms**

- Exploration versus exploitation: An agent should exploit known information from previous experiences to achieve higher cumulative rewards, but it also needs to explore to gain new experiences that can be used in choosing the best actions in the future.

**Additional Reading**

- If you are interested in more tips, workshops, classes, and other resources for improving your model, you'll find a wealth of resources on the AWS DeepRacer Pit Stop page.
- For detailed step-by-step instructions and troubleshooting support, see the AWS DeepRacer Developer Documentation.
- If you're interested in reading more posts on a range of DeepRacer topics as well as staying up to date on the newest releases, check out the AWS Discussion Forums.
- If you're interested in connecting with a thriving global community of reinforcement learning racing enthusiasts, join the AWS DeepRacer Slack community.
- If you're interested in tinkering with DeepRacer's open-source device software and collaborating with robotics innovators, check out our AWS DeepRacer GitHub Organization.

**Exercise: Interpret the reward graph of your first AWS DeepRacer model**

**Instructions**

Train a model in the AWS DeepRacer console and interpret its reward graph.

## Part 1: Train a reinforcement learning model using the AWS DeepRacer console

Practice the knowledge you've learned by training your first reinforcement learning model using the AWS DeepRacer console.

- If this is your first time using AWS DeepRacer, choose **Get started** from the service landing page, or choose Get started with reinforcement learning from the main navigation pane.
- On the **Get started with reinforcement learning** page, under Step 2: Create a model and race, choose Create model. Alternatively, on the AWS DeepRacer home page, choose Your models from the main navigation pane to open the Your models page. On the Your models page, choose Create model.
- On the **Create model page**, under Environment simulation, choose a track as a virtual environment to train your AWS DeepRacer agent. Then, choose Next. For your first run, choose a track with a simple shape and smooth turns. In later iterations, you can choose more complex tracks to progressively improve your models. To train a model for a particular racing event, choose the track most similar to the event track.
- On the **Create model** page, choose **Next**.
- On the **Create Model** page, under Race type, choose a training type. For your first run, choose Time trial. The agent with the default sensor configuration with a single-lens camera is suitable for this type of racing without modifications.
- On the **Create model** page, under **Training algorithm and hyperparameters**, choose the **Soft Actor Critic (SAC)** or **Proximal**

**Policy Optimization (PPO)** algorithm. In the AWS DeepRacer console, SAC models must be trained in continuous action spaces. PPO models can be trained in either continuous or discrete action spaces.

- On the **Create model** page, under Training algorithm and hyperparameters, use the default hyperparameter values as is. Later on, to improve training performance, expand the hyperparameters and experiment with modifying the default hyperparameter values.
- On the Create model page, under Agent, choose The Original DeepRacer or The Original DeepRacer (continuous action space) for your first model. If you use Soft Actor Critic (SAC) as your training algorithm, we filter your cars so that you can conveniently choose from a selection of compatible continuous action space agents.
- On the Create model page, choose Next.
- On the Create model page, under Reward function, use the default reward function example as is for your first model. Later on, you can choose Reward function examples to select another example function and then choose Use code to accept the selected reward function.
- On the Create model page, under Stop conditions, leave the default Maximum time value as is or set a new value to terminate the training job to help prevent long-running (and possible run-away) training jobs. When experimenting in the early phase of training, you should start with a small value for this parameter and then progressively train for longer amounts of time.
- On the Create model page, choose Create model to start creating the model and provisioning the training job instance.
- After the submission, watch your training job being initialized and then run. The initialization process takes about 6 minutes to change status from Initializing to In progress.
- Watch the Reward graph and Simulation video stream to observe the progress of your training job. You can choose the refresh button next to Reward graph periodically to refresh the Reward graph until

the training job is complete.

Note: The training job is running on the AWS Cloud, so you don't need to keep the AWS DeepRacer console open during training. However, you can come back to the console to check on your model at any point while the job is in progress.

**Part 2: Inspect your reward graph to assess your training progress**

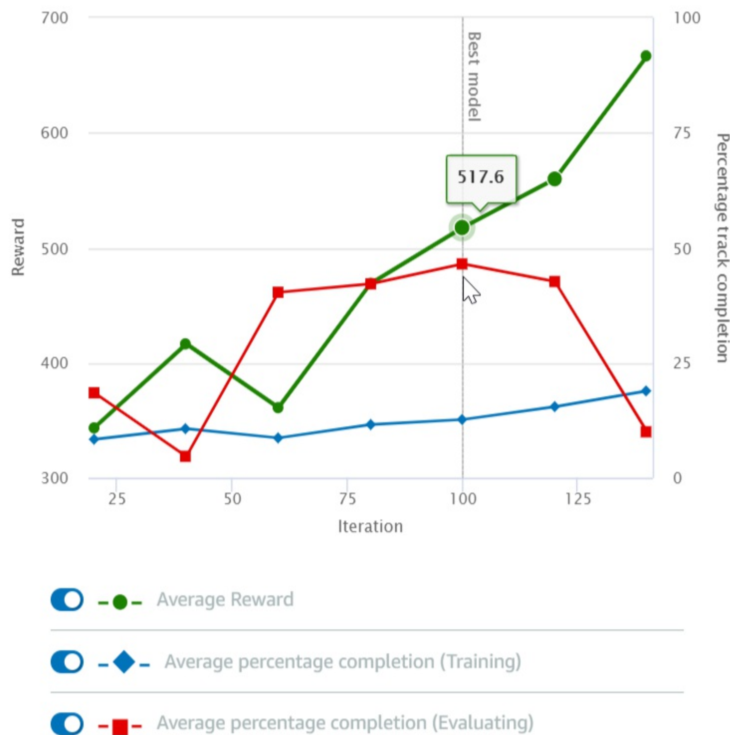
As you train and evaluate your first model, you'll want to get a sense of its quality. To do this, inspect your reward graph.

**Find the following on your reward graph:**

- Average reward
- Average percentage completion (training)
- Average percentage completion (evaluation)
- Best model line
- Reward primary y-axis
- Percentage track completion secondary y-axis
- Iteration x-axis

Review the solution to this exercise for ideas on how to interpret it.

Reward graph Info




As you train and evaluate your first model, you'll want to get a sense of its quality. To do this, inspect your reward graph.

### Exercise Solution

To get a sense of how well your training is going, watch the reward graph. Here is a list of its parts and what they do:

- **Average reward**
  - This graph represents the average reward the agent earns during a training iteration. The average is calculated by averaging the reward earned across all episodes in the training iteration. An episode begins at the starting line and ends when the agent completes one loop around the track or at the place the vehicle

left the track or collided with an object. Toggle the switch to hide this data.

- **Average percentage completion (training)**
    - The training graph represents the average percentage of the track completed by the agent in all training episodes in the current training. It shows the performance of the vehicle while experience is being gathered.
- **Average percentage completion (evaluation)**
    - While the model is being updated, the performance of the existing model is evaluated. The evaluation graph line is the average percentage of the track completed by the agent in all episodes run during the evaluation period.
- **Best model line**
    - This line allows you to see which of your model iterations had the highest average progress during the evaluation. The checkpoint for this iteration will be stored. A checkpoint is a snapshot of a model that is captured after each training (policy-updating) iteration.
- **Reward primary y-axis**
    - This shows the reward earned during a training iteration. To read the exact value of a reward, hover your mouse over the data point on the graph.
    - Percentage track completion secondary y-axis
    - This shows you the percentage of the track the agent completed during a training iteration.
- **Iteration x-axis**
    - This shows the number of iterations completed during your training job.

## Reward Graph Interpretation

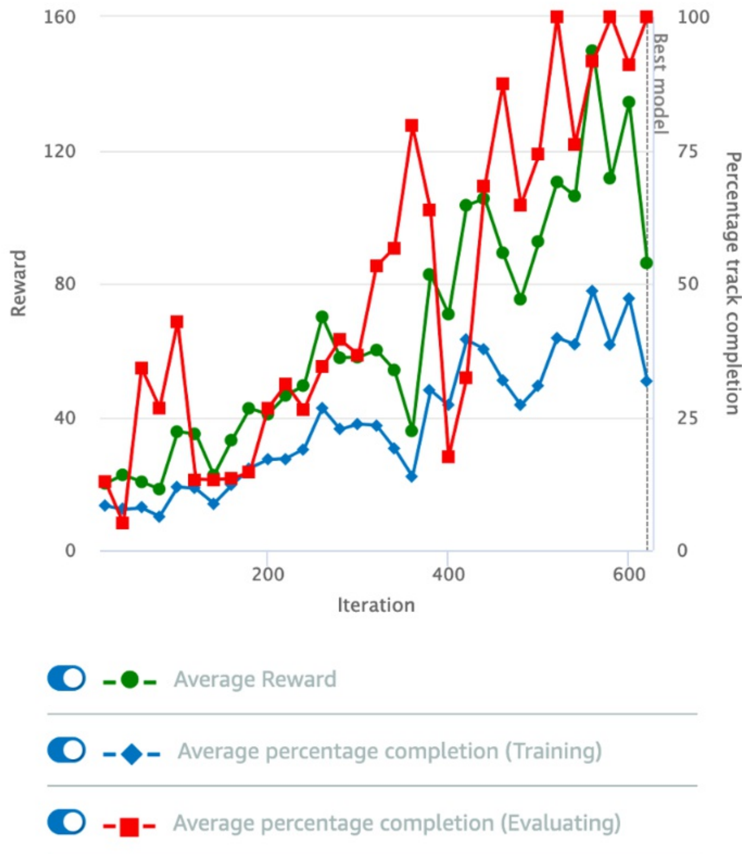
The following four examples give you a sense of how to interpret the success of your model based on the reward graph. Learning to read these graphs is as much of an art as it is a science and takes time, but reviewing the following four examples will give you a start.

## Needs more training

In the following example, we see there have only been 600 iterations, and the graphs are still going up. We see the evaluation completion percentage has just reached 100%, which is a good sign but isn't fully consistent yet, and the training completion graph still has a ways to go. This reward function

and model are showing promise, but need more training time.

Reward graph Info

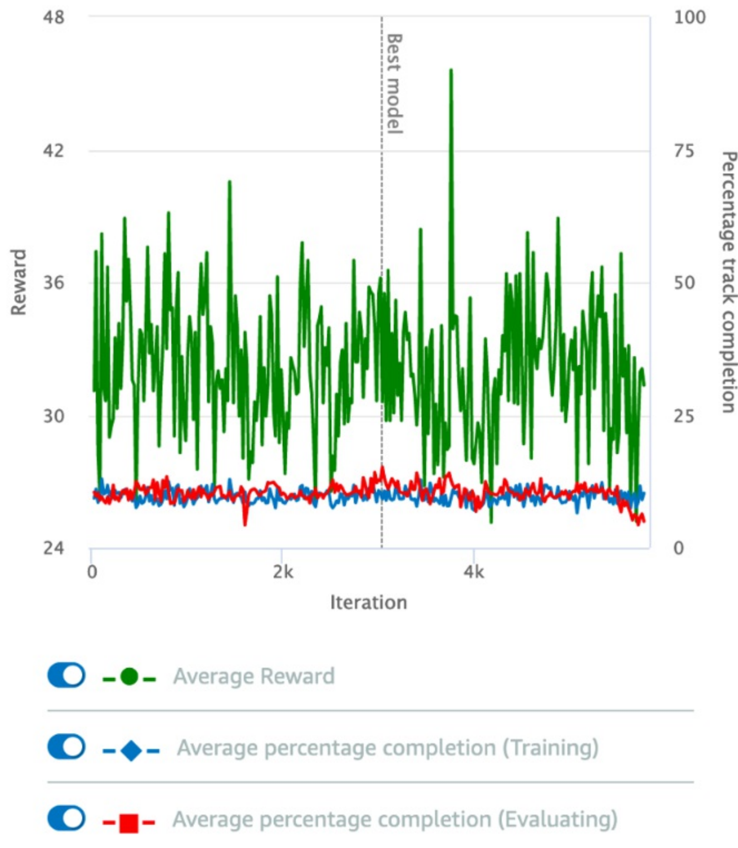



*Needs more training*

**No improvement**

In the next example, we can see that the percentage of track completions haven’t gone above around 15 percent and it's been training for quite some time—probably around 6000 iterations or so. This is not a good sign! Consider throwing this model and reward function away and trying a different strategy.

Reward graph Info




*No improvement*

## A well-trained model

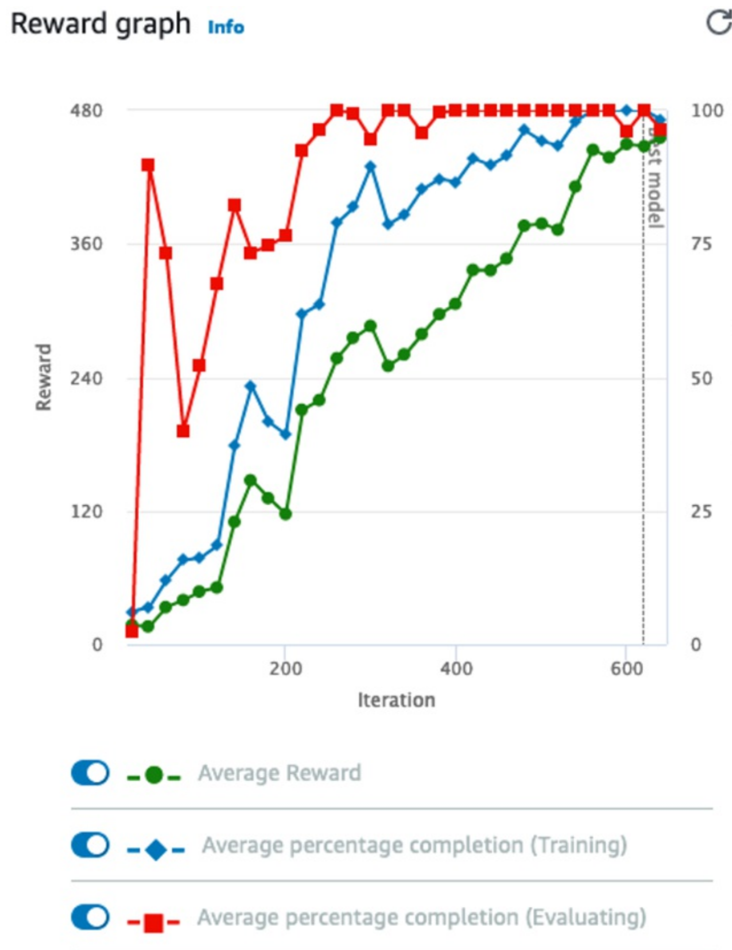
In the following example graph, we see the evaluation percentage completion reached 100% a while ago, and the training percentage reached 100% roughly 100 or so iterations ago. At this point, the model is well trained. Training it further might lead to the model becoming overfit to this track.

## Avoid overfitting

Overfitting or overtraining is a really important concept in machine learning.

With AWS DeepRacer, this can become an issue when a model is trained on a specific track for too long. A good model should be able to make decisions based on the features of the road, such as the sidelines and centerlines, and be able to drive on just about any track.

An overtrained model, on the other hand, learns to navigate using landmarks specific to an individual track. For example, the agent turns a certain direction when it sees uniquely shaped grass in the background or a specific angle the corner of the wall makes. The resulting model will run beautifully on that specific track, but perform badly on a different virtual track, or even on the same track in a physical environment due to slight variations in angles, textures, and lighting.

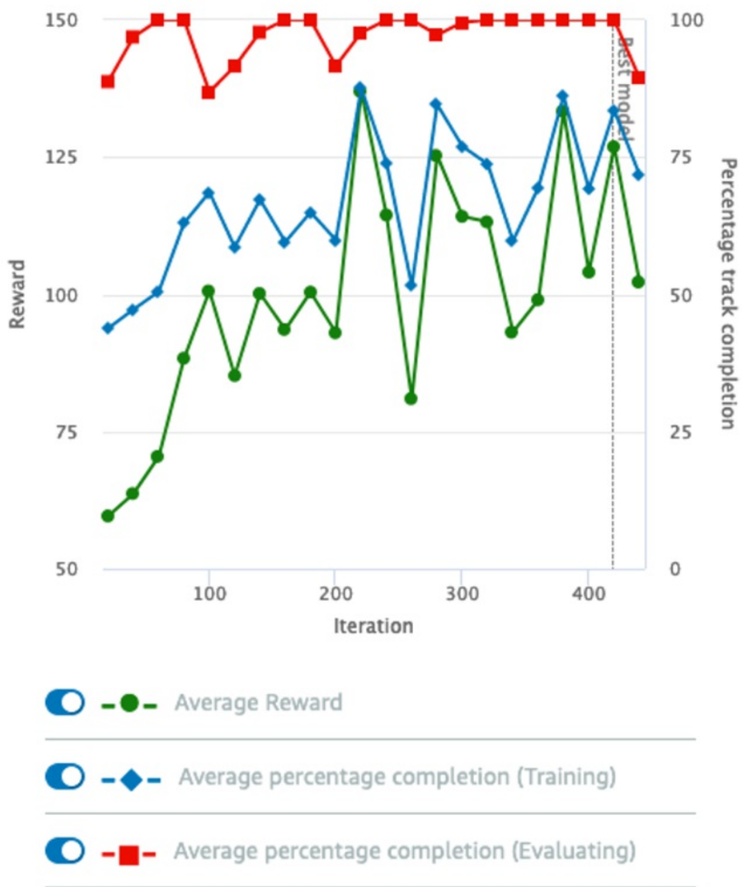


**Adjust hyperparameters**

The AWS DeepRacer console's default hyperparameters are quite effective, but occasionally you may consider adjusting the training hyperparameters. The hyperparameters are variables that essentially act as settings for the training algorithm that control the performance of your agent during training. We learned, for example, that the learning rate controls how many new experiences are counted in learning at each step.

In this reward graph example, the training completion graph and the reward graph are swinging high and low. This might suggest an inability to converge, which may be helped by adjusting the learning rate. Imagine if the current weight for a given node is .03, and the optimal weight should be .035, but your learning rate was set to .01. The next training iteration would then swing past optimal to .04, and the following iteration would swing under it to .03 again. If you suspect this, you can reduce the learning rate to .001. A lower learning rate makes learning take longer but can help increase the quality of your model.

Reward graph Info




*Adjust hyperparameters*

**Good Job and Good Luck!**

Remember: training experience helps both model and reinforcement learning practitioners become a better team. Enter your model in the monthly AWS DeepRacer League races for chances to win prizes and glory while improving your machine learning development skills!

# Introduction to Generative AI

**Generative AI and Its Applications**

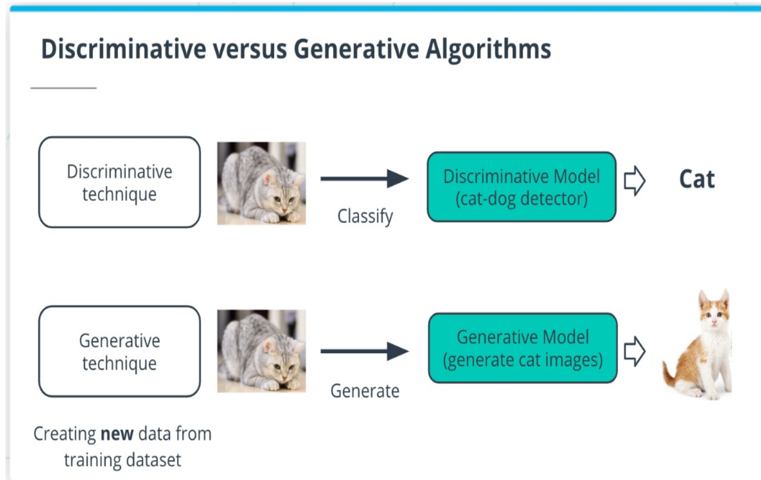
Generative AI is one of the biggest recent advancements in artificial intelligence because of its ability to create new things.

Until recently, the majority of machine learning applications were powered by discriminative models. A discriminative model aims to answer the question, "If I'm looking at some data, how can I best classify this data or predict a value?" For example, we could use discriminative models to detect if a camera was pointed at a cat.

As we train this model over a collection of images (some of which contain cats and others which do not), we expect the model to find patterns in images which help make this prediction.

A generative model aims to answer the question,"Have I seen data like this before?" In our image classification example, we might still use a generative model by framing the problem in terms of whether an image with the label "cat" is more similar to data you've seen before than an image with the label "no cat."

However, generative models can be used to support a second use case. The patterns learned in generative models can be used to create brand new examples of data which look similar to the data it seen before.




*Discriminative versus Generative algorithms*

# Generative AI Models

In this lesson, you will learn how to create three popular types of generative models: generative adversarial networks (GANs), general autoregressive models, and transformer-based models. Each of these is accessible through AWS DeepComposer to give you hands-on experience with using these techniques to generate new examples of music.

### Autoregressive models

Autoregressive convolutional neural networks (AR-CNNs) are used to study systems that evolve over time and assume that the likelihood of some data depends only on what has happened in the past. It's a useful way of looking at many systems, from weather prediction to stock prediction.

### Generative adversarial networks (GANs)

Generative adversarial networks (GANs), are a machine learning model

format that involves pitting two networks against each other to generate new content. The training algorithm swaps back and forth between training a generator network (responsible for producing new data) and a discriminator network (responsible for measuring how closely the generator network's data represents the training dataset).

**Transformer-based models**

Transformer-based models are most often used to study data with some sequential structure (such as the sequence of words in a sentence). Transformer-based methods are now a common modern tool for modeling natural language.

We won't cover this approach in this course but you can learn more about transformers and how AWS DeepComposer uses transformers in AWS DeepComposer learning capsules.

# Generative AI with AWS DeepComposer

### What is AWS DeepComposer?

AWS DeepComposer gives you a creative and easy way to get started with machine learning (ML), specifically generative AI. It consists of a USB keyboard that connects to your computer to input melody and the AWS DeepComposer console, which includes AWS DeepComposer Music studio to generate music, learning capsules to dive deep into generative AI models,

and AWS DeepComposer Chartbusters challenges to showcase your ML skills.

*AWS DeepComposer*

*AWS DeepComposer keyboard*

You don't need an AWS DeepComposer keyboard to finish this course. You can import your own MIDI file, use one of the provided sample melodies, or use the virtual keyboard in the AWS DeepComposer Music studio.

**AWS DeepComposer music studio**

To generate, create, and edit compositions with AWS DeepComposer, you use the AWS DeepComposer Music studio. To get started, you need an input track and a trained model.

For the input track, you can use a sample track, record a custom track, or import a track.

*Input track*

For the ML technique, you can use either a sample model or a custom model.

Each AWS DeepComposer Music studio experience supports three different generative AI techniques: generative adversarial networks (GANs), autoregressive convolutional neural network (AR-CNNs), and transformers.

- Use the GAN technique to create accompaniment tracks.
- Use the AR-CNN technique to modify notes in your input track.
- Use the transformers technique to extend your input track by up to 30 seconds.

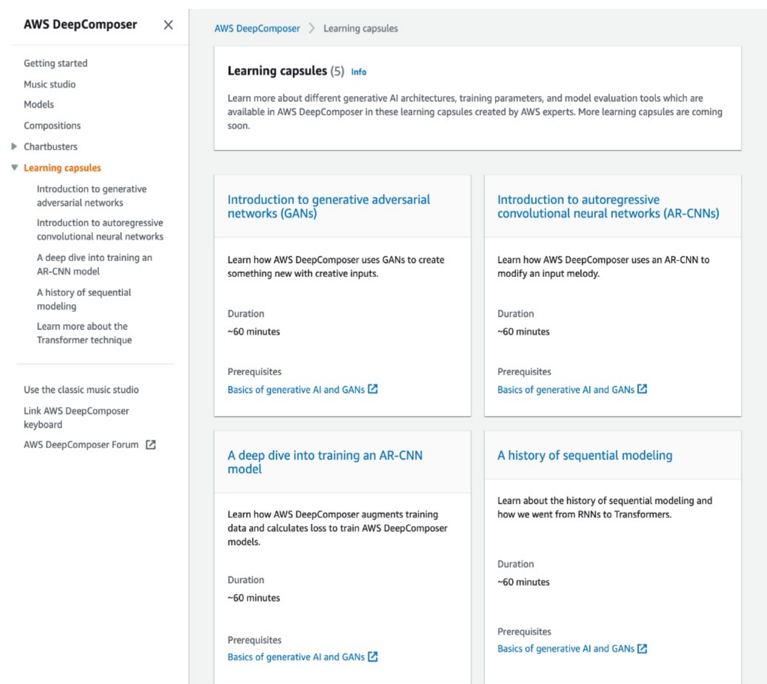
*ML models*

## Demo: AWS DeepComposer

In this demo, you went through the AWS DeepComposer console where you can learn about deep learning, input your music, and train deep learning models to create new music.

## AWS DeepComposer learning capsules

To learn the details behind generative AI and ML techniques used in AWS DeepComposer you can use easy-to-consume, bite-sized learning capsules in the AWS DeepComposer console.




*AWS DeepComposer learning capsules*

## AWS DeepComposer Chartbusters challenges

Chartbusters is a global challenge where you can use AWS DeepComposer to create original compositions and compete in monthly challenges to showcase your machine learning and generative AI skills.

You don't need to participate in this challenge to finish this course, but the course teaches everything you need to win in both challenges we launched this season. Regardless of your background in music or ML, you can find a competition just right for you.

You can choose between two different challenges this season:

- In the Basic challenge, “Melody-Go-Round”, you can use any machine learning technique in the AWS DeepComposer Music studio to create new compositions.
- In the Advanced challenge, “Melody Harvest”, you train a custom generative AI model using Amazon SageMaker.

# GANs with AWS DeepComposer

We’ll begin our journey of popular generative models in AWS DeepComposer with generative adversarial networks or GANs. Within an AWS DeepComposer GAN, models are used to solve a creative task: adding accompaniments that match the style of an input track you provide. Listen to the input melody and the output composition created by the AWS DeepComposer GAN model:

- Input melody
- Output melody

**What are GANs?**

A GAN is a type of generative machine learning model which pits two neural networks against each other to generate new content: a generator and a discriminator.

- A generator is a neural network that learns to create new data resembling the source data on which it was trained.
- A discriminator is another neural network trained to differentiate between real and synthetic data.

The generator and the discriminator are trained in alternating cycles. The generator learns to produce more and more realistic data while the discriminator iteratively gets better at learning to differentiate real data from the newly created data.

**Collaboration between an orchestra and its conductor**

A simple metaphor of an orchestra and its conductor can be used to understand a GAN. The orchestra trains, practices, and tries to generate polished music, and then the conductor works with them, as both judge and coach. The conductor judges the quality of the output and at the same time provides feedback to achieve a specific style. The more they work together, the better the orchestra can perform.

The GAN models that AWS DeepComposer uses work in a similar fashion. There are two competing networks working together to learn how to generate musical compositions in distinctive styles.

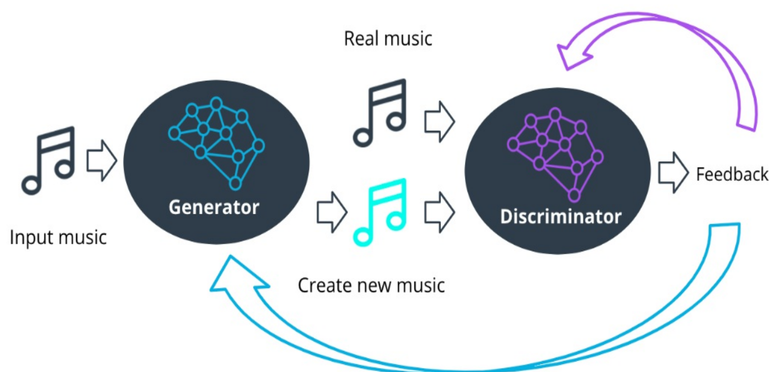
A GAN's generator produces new music as the orchestra does. And the

discriminator judges whether the music generator creates is realistic and provides feedback on how to make its data more realistic, just as a conductor provides feedback to make an orchestra sound better.

*An orchestra and its conductor*

# Training Methodology

Let’s dig one level deeper by looking at how GANs are trained and used within AWS DeepComposer. During training, the generator and discriminator work in a tight loop as depicted in the following image.




*A schema representing a GAN model used within AWS DeepComposer*

**Note**: While this figure shows the generator taking input on the left, GANs in general can also generate new data without any input.

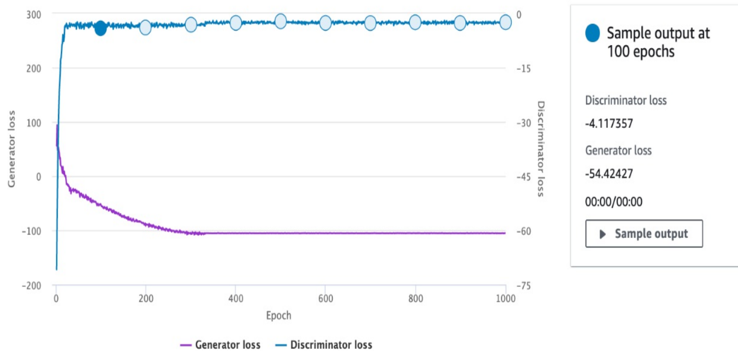
**Generator**

- The generator takes in a batch of single-track piano rolls (melody) as the input and generates a batch of multi-track piano rolls as the output by adding accompaniments to each of the input music tracks.
- The discriminator then takes these generated music tracks and predicts how far they deviate from the real data present in the training dataset. This deviation is called the generator loss. This feedback from the discriminator is used by the generator to incrementally get better at creating realistic output.

**Discriminator**

- As the generator gets better at creating music accompaniments, it begins fooling the discriminator. So, the discriminator needs to be retrained as well. The discriminator measures the discriminator loss to evaluate how well it is differentiating between real and fake data.

Beginning with the discriminator on the first iteration, we **alternate training these two networks** until we reach some stop condition; for example, the algorithm has seen the entire dataset a certain number of times or the generator and discriminator loss reach some plateau (as shown in the following image).




*Discriminator loss and generator loss reach a plateau*

**New Terms**

- **Generator**: A neural network that learns to create new data resembling the source data on which it was trained.
- **Discriminator**: A neural network trained to differentiate between real and synthetic data.
- **Generator loss**: Measures how far the output data deviates from the real data present in the training dataset.
- **Discriminator loss**: Evaluates how well the discriminator differentiates between real and fake data.

# AR-CNN with AWS DeepComposer

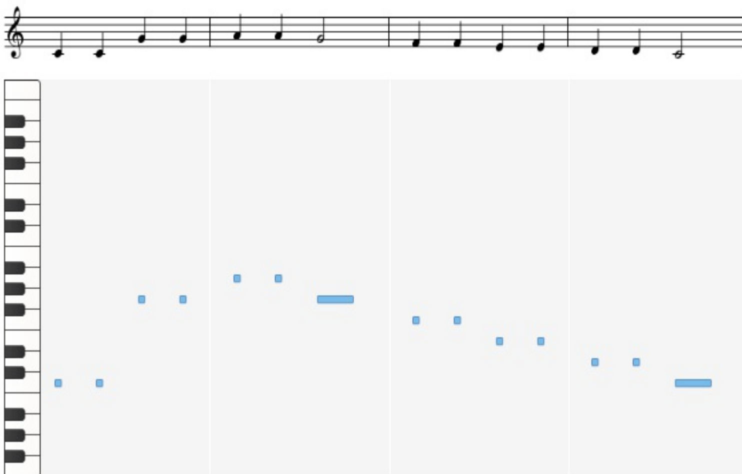
Our next popular generative model is the autoregressive convolutional neural network (AR-CNN). Autoregressive convolutional neural networks make iterative changes over time to create new data.

To better understand how the AR-CNN model works, let’s first discuss how music is represented so it is machine-readable.

**Image-based representation**

Nearly all machine learning algorithms operate on data as numbers or sequences of numbers. In AWS DeepComposer, the input tracks are represented as a piano roll**. *In each two-dimensional piano roll, time is on the horizontal axis and pitch* is on the vertical axis. You might notice this representation looks similar to an image.

The AR-CNN model uses a piano roll image to represent the audio files from the dataset. You can see an example in the following image where on top is a musical score and below is a piano roll image of that same score.



*Musical score and piano roll*

**How the AR-CNN Model Works**

When a note is either added or removed from your input track during inference, we call it an edit event. To train the AR-CNN model to predict when notes need to be added or removed from your input track (edit

event), the model iteratively updates the input track to sounds more like the training dataset. During training, the model is also challenged to detect differences between an original piano roll and a newly modified piano roll.

**New Terms**

- **Piano roll**: A two-dimensional piano roll matrix that represents input tracks. Time is on the horizontal axis and pitch is on the vertical axis.
- **Edit event**: When a note is either added or removed from your input track during inference.

**Demo: Create Music with AWS DeepComposer**

Below you find a video demonstrating how you can use AWS DeepComposer to experiment with GANs and AR-CNN models.

**Important**

To get you started, AWS DeepComposer provides a 12-month Free Tier for first-time users. With the Free Tier, you can perform up to 500 inference jobs, translating to 500 pieces of music, using the AWS DeepComposer Music studio. You can use one of these instances to complete the exercise at no cost. For more information, please read the AWS account requirements page.

In the demo, you have learned how to create music using AWS Deepcomposer.

You will need a music track to get started. There are several ways to do it. You can record your own using the AWS keyboard device or the virtual keyboard provided in the console. Or you can input a MIDI file or choose a provided music track.

Once the music track is inputted, choose "Continue" to create a model. The models you can choose are AR-CNN, GAN, and transformers. Each of them has a slightly different function. After choosing a model, you can then adjust the parameters used to train the model.

Once you are done with model creation, you can select "Continue" to listen and improve your output melody. To edit the melody, you can either drag or extend notes directly on the piano roll or adjust the model parameters and train it again. Keep tuning your melody until you are happy with it then click "Continue" to finish the composition.

If you want to enhance your music further with another generative model, you can do it too. Simply choose a model under the "Next step" section and create a new model to enhance your music.

Congratulations on creating your first piece of music using AWS DeepComposer! Now you can download the melody or submit it to a competition. Hope you enjoy the journey of creating music with AWS DeepComposer.

**Exercise: Generate music with AWS DeepComposer**

You have seen how the instructor generates a piece of music in AWS DeepComposer. Now, it's your turn to create your very own piece of music.

To finish this exercise, you should complete the following steps.

- Open the AWS DeepComposer console.
- In the navigation pane, choose Music studio, then choose Start composing.
- On the Input track page, record a melody using the virtual keyboard, import a MIDI file, or choose an input track. On the ML technique page, choose AR-CNN. On the Inference output page, you can do the following:
  - Change the AR-CNN parameters, choose Enhance again, and then choose Play to hear how your track has changed. Repeat until you like the outcome.
  - Choose Edit melody to modify and change the notes that were added during inference.

Choose Continue to finish creating your composition.

You can then choose Share composition, Register, or Sign in to Soundcloud and submit to the "Melody-Go-Round" competition. Participation in the competition is optional.

If you get stuck, you can check out the demo videos on Demo: Create Music with AWS DeepComposer page.

# Build a Custom GAN Model (Optional): Part 1

To create the custom GAN, you will need to use an instance type that is not covered in the Amazon SageMaker free tier. You may incur a cost if you want to build a custom GAN.

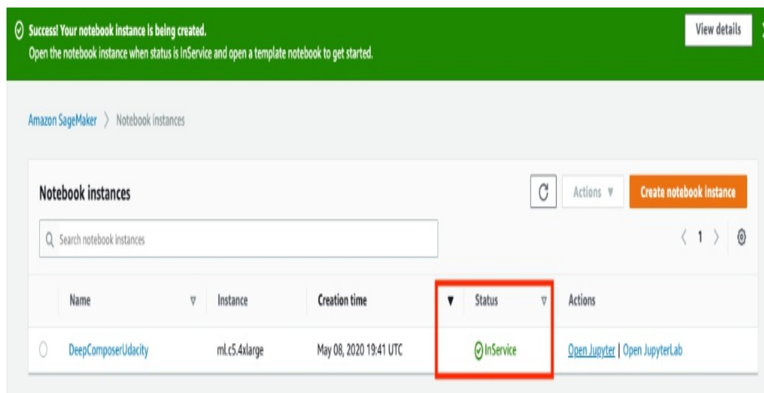
You can learn more about SageMaker costs in the Amazon SageMaker pricing documentation.

Important: This is an optional exercise. You will not need to complete this exercise in order to graduate from the foundation's course. If you are unable to create an AWS account, please move to the next module.

**Setting Up the AWS DeepComposer Notebook**

- Go to the AWS Management Console and search for Amazon SageMaker.
- Once inside the SageMaker console, look to the left-hand menu and select Notebook Instances.
- Next, choose Create notebook instance.
- In the Notebook instance setting section, give the notebook a name, for example, DeepComposerUdacity.
- Based on the kind of CPU, GPU and memory you need the next step is to select an instance type. For our purposes, we'll configure a ml.c5.4xlarge
- Leave the Elastic Inference defaulted to none.
- In the Permissions and encryption section, create a new IAM role using all of the defaults.
- When you see that the role was created successfully, navigate down a little way to the Git repositories section

- Select Clone a public Git repository to this notebook instance only
- Copy and paste the public URL into the Git repository URL section: https://github.com/aws-samples/aws-deepcomposer-samples
- Select Create notebook instance
- Give SageMaker a few minutes to provision the instance and clone the Git repository
- When the status reads "InService" you can open the Jupyter notebook.




**Exploring the Notebook**

Now that it's configured and ready to use, let's take a moment to investigate what's inside the notebook.

**Open the Notebook**

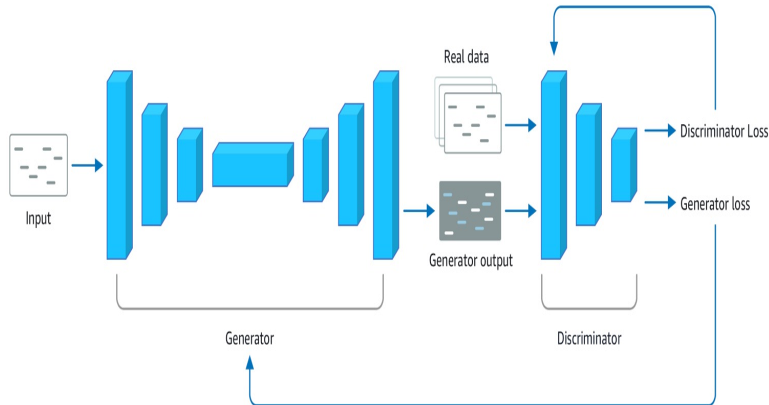
- Click Open Jupyter.
- When the notebook opens, click on "gan".
- When the lab opens click on GAN.ipynb.

**Review: Generative Adversarial Networks (GANs).**

GANs consist of two networks constantly competing with each other:

- Generator network that tries to generate data based on the data it was trained on.

- Discriminator network that is trained to differentiate between real data and data which is created by the generator.




*A diagram of generator and discriminator.*

**Set Up the Project**

- Run the first Dependencies cell to install the required packages
- Run the second Dependencies cell to import the dependencies
- Run the Configuration cell to define the configuration variables

Note: While executing the cell that installs dependency packages, you may see warning messages indicating that later versions of conda are available for certain packages. It is completely OK to ignore this message. It should not affect the execution of this notebook.

*Click Run or Shift-Enter in the cell*

**Good Coding Practices**

- Do not hard-code configuration variables
- Move configuration variables to a separate config file
- Use code comments to allow for easy code collaboration

**Data Preparation**

The next section of the notebook is where we'll prepare the data so it can train the generator network.

**Why Do We Need to Prepare Data?**

Data often comes from many places (like a website, IoT sensors, a hard drive, or physical paper) and it's usually not clean or in the same format. Before you can better understand your data, you need to make sure it's in the right format to be analyzed. Thankfully, there are library packages that can help! One such library is called NumPy, which was imported into our notebook.

**Piano Roll Format**

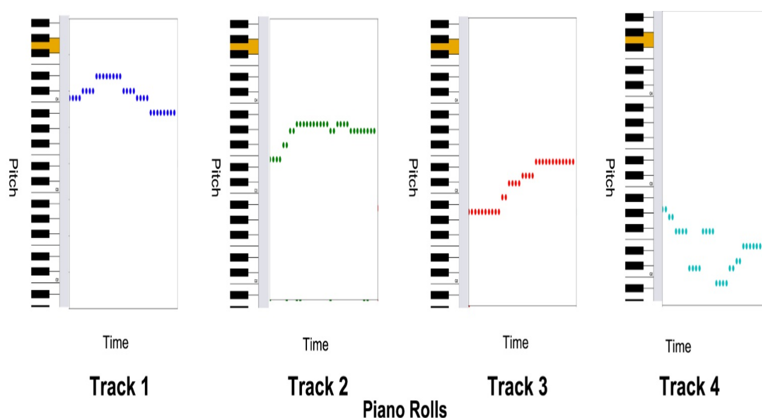
The data we are preparing today is music and it comes formatted in what's called a "piano roll". Think of a piano roll as a 2D image where the X-axis represents time and the Y-axis represents the pitch value. Using music as images allows us to leverage existing techniques within the computer vision domain.

Our data is stored as a *NumPy* Array, or grid of values. Our dataset comprises 229 samples of 4 tracks (all tracks are piano). Each sample is a 32 time-step snippet of a song, so our dataset has a shape of:

*(num_samples, time_steps, pitch_range, tracks)*

*or*

*(229, 32, 128, 4)*

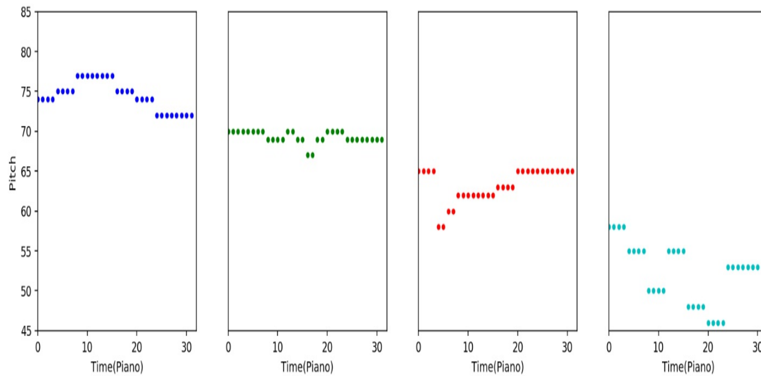



*Each Piano Roll Represents A Separate Piano Track in the Song*

**Load and View the Dataset**

- Run the next cell to play a song from the dataset.

- Run the next cell to load the dataset as a nympy array and output the shape of the data to confirm that it matches the (229, 32, 128, 4) shape we are expecting
- Run the next cell to see a graphical representation of the data.




*Graphical Representation of Model Data*

**Create a Tensorflow Dataset**

Much like there are different libraries to help with cleaning and formatting data, there are also different frameworks. Some frameworks are better suited for particular kinds of machine learning workloads and for this deep learning use case, we're going to use a Tensorflow framework with a Keras library.

We'll use the dataset object to feed batches of data into our model.

- Run the first Load Data cell to set parameters.
- Run the second Load Data cell to prepare the data.

**Build a Custom GAN Model (Optional): Part 2**

To create the custom GAN, you will need to use an instance type that is not covered in the Amazon SageMaker free tier. You may incur a cost if you

want to build a custom GAN.

You can learn more about SageMaker costs in the Amazon SageMaker pricing documentation.

**Important**: This is an optional exercise. You will not need to complete this exercise in order to graduate from the foundation's course. If you are unable to create an AWS account, please move to the next module.

**Model Architecture**

Before we can train our model, let's take a closer look at model architecture including how GAN networks interact with the batches of data we feed into the model, and how the networks communicate with each other.

**How the Model Works**

The model consists of two networks, a generator and a discriminator (critic). These two networks work in a tight loop:

- The generator takes in a batch of single-track piano rolls (melody) as the input and generates a batch of multi-track piano rolls as the output by adding accompaniments to each of the input music tracks.
- The discriminator evaluates the generated music tracks and predicts how far they deviate from the real data in the training dataset.
- The feedback from the discriminator is used by the generator to help it produce more realistic music the next time.
- As the generator gets better at creating better music and fooling the discriminator, the discriminator needs to be retrained by using music tracks just generated by the generator as fake inputs and an equivalent number of songs from the original dataset as the real

input.

- We alternate between training these two networks until the model converges and produces realistic music.

The discriminator is a binary classifier which means that it classifies inputs into two groups, e.g. “real” or “fake” data.

- Defining and Building Our Model
- Run the cell that defines the generator
- Run the cell that builds the generator
- Run the cell that defines the discriminator
- Run the cell that builds the discriminator

**Model Training and Loss Functions**

As the model tries to identify data as “real” or “fake”, it’s going to make errors. Any prediction different than the ground truth is referred to as an error.

The measure of the error in the prediction, given a set of weights, is called a loss function. Weights represent how important an associated feature is to determining the accuracy of a prediction.

Loss functions are an important element of training a machine learning model because they are used to update the weights after every iteration of your model. Updating weights after iterations optimizes the model making the errors smaller and smaller.

**Setting Up and Running the Model Training**

- Run the cell that defines the loss functions
- Run the cell to set up the optimizer

- Run the cell to define the generator step function
- Run the cell to define the discriminator step function
- Run the cell to load the melody samples
- Run the cell to set the parameters for the training
- Run the cell to train the model!!!!

Training and tuning models can take a very long time – weeks or even months sometimes. Our model will take around an hour to train.

**Model Evaluation**

Now that the model has finished training it's time to evaluate its results.

There are several evaluation metrics you can calculate for classification problems and typically these are decided in the beginning phases as you organize your workflow.

**You can:**

- Check to see if the losses for the networks are converging
- Look at commonly used musical metrics of the generated sample and compared them to the training dataset.

**Evaluating Our Training Results**

- Run the cell to restore the saved checkpoint. If you don't want to wait to complete the training you can use data from a pre-trained model by setting TRAIN = False in the cell.
- Run the cell to plot the losses.
- Run the cell to plot the metrics.

## Results and Inference

Finally, we are ready to hear what the model produced and visualize the piano roll output!

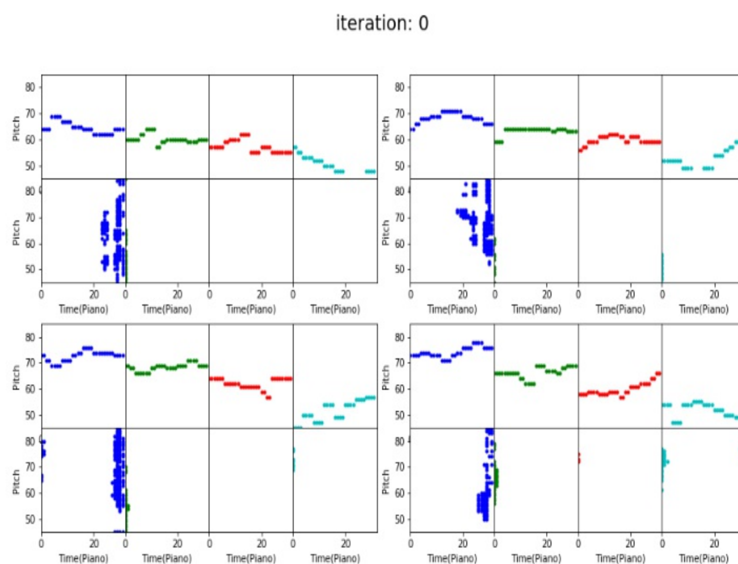
Once the model is trained and producing acceptable quality, it’s time to see how it does on data it hasn’t seen. We can test the model on these unknown inputs, using the results as a proxy for performance on future data.

## Evaluate the Generated Music

- In the first cell, enter 0 as the iteration number.

run the cell and play the music snippet.

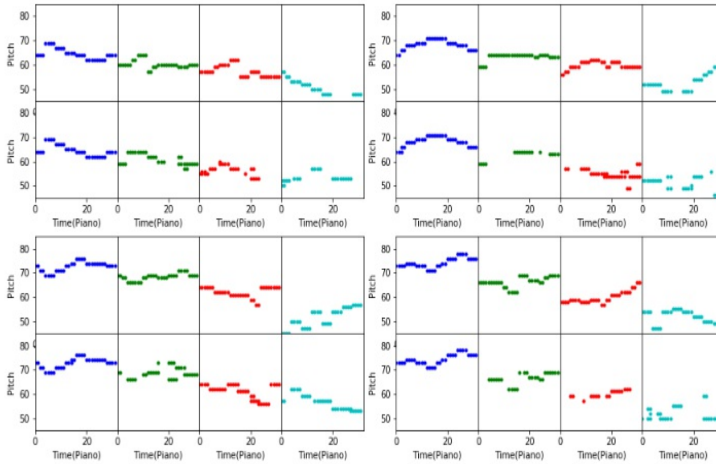
Or listen to this example snippet from iteration 0:




*Example Piano Roll at Iteration 0*

- In the first cell, enter 500 as the iteration number:run the cell and play the music snippet. Or listen to the example snippet at iteration 500.
- In the second cell, enter 500 as the iteration number:run the cell and display the piano roll.

iteration: 500

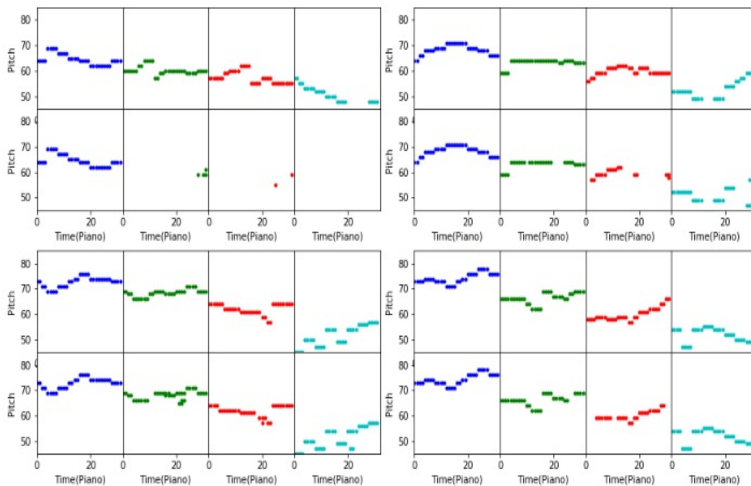


*Example Piano Roll at Iteration 500*

Play around with the iteration number and see how the output changes over time!

Here is an example snippet at iteration 950

And here is the piano roll:




*Example Piano Roll at Iteration 950*

Do you see or hear a quality difference between iteration 500 and iteration 950?

**Watch the Evolution of the Model!**

Run the next cell to create a video to see how the generated piano rolls change over time.

**Inference**

Now that the GAN has been trained we can run it on a custom input to generate music.

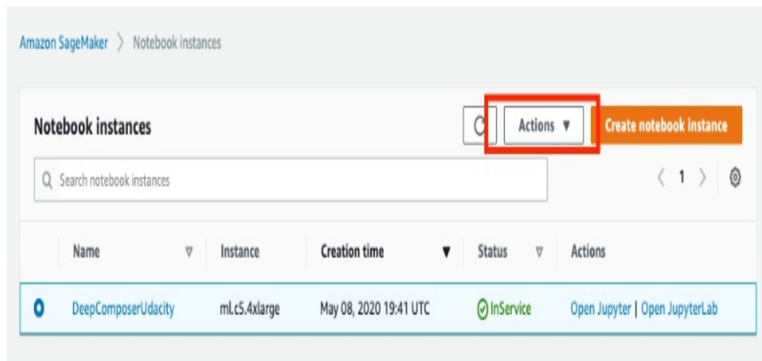
- Run the cell to generate a new song based on "Twinkle Twinkle Little Star". Or listen to the example of the generated music here:
- Run the next cell and play the generated music. Or listen to the example of the generated music here:

**Stop and Delete the Jupyter Notebook When You Are Finished!**

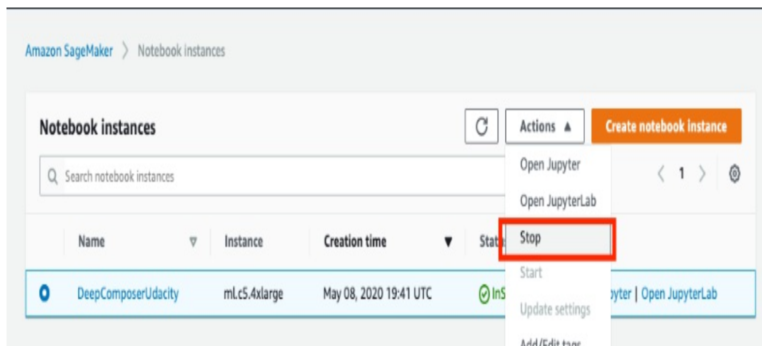
This project is not covered by the AWS Free Tier so ***your project will continue to accrue costs as long as it is running.***

Follow these steps to stop and delete the notebook.

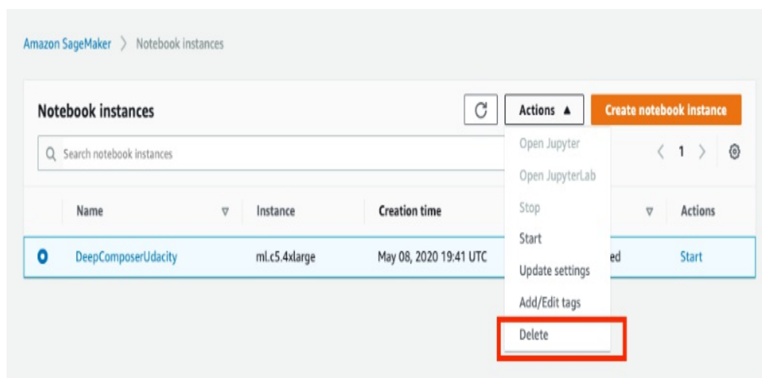
- Go back to the Amazon SageMaker console.
- Select the notebook and click Actions.




- Select Stop and wait for the instance to stop.




- Select Delete



## Recap

In this demo we learned how to setup a Jupyter notebook in Amazon SageMaker, reviewed a machine learning code, and what data preparation, model training, and model evaluation can look like in a notebook instance. While this was a fun use case for us to explore, the concepts and techniques can be applied to other machine learning projects like an object detector or a sentiment analysis on text.

## Glossary

- **Action**: For every state, an agent needs to take an action toward achieving its goal.
- **Agent**: The piece of software you are training is called an agent. It makes decisions in an environment to reach a goal.
- **Discriminator**: A neural network trained to differentiate between real and synthetic data.
- **Discriminator loss**: Evaluates how well the discriminator differentiates between real and fake data.
- **Edit event**: When a note is either added or removed from your input track during inference.
- **Environment**: The environment is the surrounding area within which the agent interacts.
- **Exploration versus exploitation**: An agent should exploit known information from previous experiences to achieve higher cumulative rewards, but it also needs to explore to gain new experiences that can be used in choosing the best actions in the future.
- **Generator**: A neural network that learns to create new data resembling the source data on which it was trained.
- **Generator loss**: Measures how far the output data deviates from the real data present in the training dataset.

- **Hidden layer**: A layer that occurs between the output and input layers. Hidden layers are tailored to a specific task.
- **Input layer**: The first layer in a neural network. This layer receives all data that passes through the neural network.
- **Output layer**: The last layer in a neural network. This layer is where the predictions are generated based on the information captured in the hidden layers.
- **Piano roll**: A two-dimensional piano roll matrix that represents input tracks. Time is on the horizontal axis and pitch is on the vertical axis.
- **Reward**: Feedback is given to an agent for each action it takes in a given state. This feedback is a numerical reward.

# CHAPTER FOUR

# Software Engineering Practices, Part I

In this lesson, you'll learn about the following software engineering practices and how they apply in data science.

- Writing clean and modular code
- Writing efficient code
- Code refactoring
- Adding meaningful documentation
- Using version control

In the lesson following this one (part 2), you'll also learn about the following software engineering practices:

- Testing
- Logging
- Code reviews

# Clean and Modular Code

- **Production code**: Software running on production servers to handle live users and data of the intended audience. Note that this is different from production-quality code, which describes code that meets expectations for production in reliability, efficiency, and other aspects. Ideally, all code in production meets these expectations, but this is not always the case.
- **Clean code**: Code that is readable, simple, and concise. Clean production-quality code is crucial for collaboration and maintainability in software development.
- **Modular code**: Code that is logically broken up into functions and modules. Modular production-quality code that makes your code more organized, efficient, and reusable.

- **Module**: A file. Modules allow code to be reused by encapsulating them into files that can be imported into other files.

# Refactoring Code

- *Refactoring*: Restructuring your code to improve its internal structure without changing its external functionality. This gives you a chance to clean and modularize your program after you've got it working.
- Since it isn't easy to write your best code while you're still trying to just get it working, allocating time to do this is essential to producing high-quality code. Despite the initial time and effort required, this really pays off by speeding up your development time in the long run.
- You become a much stronger programmer when you're constantly looking to improve your code. The more you refactor, the easier it will be to structure and write good code the first time.

**Why Refactor?**

- Reduce workload in the long run
- Easier to maintain code
- Reuse more of your code
- Become a better developer

# Writing Clean Code

**Writing clean code: Meaningful names**

*Use meaningful names.*

- *Be descriptive and imply type*: For booleans, you can prefix with is_ or has_ to make it clear it is a condition. You can also use parts of speech to imply types, like using verbs for functions and nouns for variables.
- *Be consistent but clearly differentiate*: age_list and age is easier to differentiate than ages and age.
- *Avoid abbreviations and single letters*: You can determine when to make these exceptions based on the audience for your code. If you work with other data scientists, certain variables may be common knowledge. While if you work with full stack engineers, it might be necessary to provide more descriptive names in these cases as well. (Exceptions include counters and common math variables.)
- *Long names aren't the same as descriptive names*: You should be descriptive, but only with relevant information. For example, good function names describe what they do well without including details about implementation or highly specific uses.

Try testing how effective your names are by asking a fellow programmer to guess the purpose of a function or variable based on its name, without looking at your code. Coming up with meaningful names often requires effort to get right.

**Writing clean code: Nice whitespace**

*Use whitespace properly.*

- Organize your code with consistent indentation: the standard is to use four spaces for each indent. You can make this a default in

your text editor.

- Separate sections with blank lines to keep your code well organized and readable.
- Try to limit your lines to around 79 characters, which is the guideline given in the PEP 8 style guide. In many good text editors, there is a setting to display a subtle line that indicates where the 79 character limit is.

For more guidelines, check out the code layout section of PEP 8 in the following notes.

# Writing Modular Code

Follow the tips below to write modular code.

**Tip**: *DRY (Don't Repeat Yourself)*

Don't repeat yourself! Modularization allows you to reuse parts of your code. Generalize and consolidate repeated code in functions or loops.

**Tip**: *Abstract out logic to improve readability*

Abstracting out code into a function not only makes it less repetitive, but also improves readability with descriptive function names. Although your code can become more readable when you abstract out logic into functions, it is possible to over-engineer this and have way too many modules, so use your judgement.

**Tip**: *Minimize the number of entities (functions, classes, modules, etc.)*

There are trade-offs to having function calls instead of inline logic. If you have broken up your code into an unnecessary amount of functions and modules, you'll have to jump around everywhere if you want to view the

implementation details for something that may be too small to be worth it. Creating more modules doesn't necessarily result in effective modularization.

**Tip**: *Functions should do one thing*

Each function you write should be focused on doing one thing. If a function is doing multiple things, it becomes more difficult to generalize and reuse. Generally, if there's an "and" in your function name, consider refactoring.

**Tip**: *Arbitrary variable names can be more effective in certain functions*

Arbitrary variable names in general functions can actually make the code more readable.

**Tip**: *Try to use fewer than three arguments per function*

Try to use no more than three arguments when possible. This is not a hard rule and there are times when it is more appropriate to use many parameters. But in many cases, it's more effective to use fewer arguments. Remember we are modularizing to simplify our code and make it more efficient. If your function has a lot of parameters, you may want to rethink how you are splitting this up.

# Efficient Code

Knowing how to write code that runs efficiently is another essential skill in software development. Optimizing code to be more efficient can mean making it:

- Execute faster
- Take up less space in memory/storage

The project on which you're working determines which of these is more important to optimize for your company or product. When you're performing lots of different transformations on large amounts of data, this can make orders of magnitudes of difference in performance.

# Documentation

- Documentation: Additional text or illustrated information that comes with or is embedded in the code of software.
- Documentation is helpful for clarifying complex parts of code, making your code easier to navigate, and quickly conveying how and why different components of your program are used.
- Several types of documentation can be added at different levels of your program:
  - **Inline comments** - line level
  - **Docstrings** - module and function level
  - **Project documentation** - project level

# Inline Comments

- Inline comments are text following hash symbols throughout your code. They are used to explain parts of your code, and really help future contributors understand your work.
- Comments often document the major steps of complex code. Readers may not have to understand the code to follow what it does if the comments explain it. However, others would argue that this is using comments to justify bad code, and that if code requires comments to follow, it is a sign refactoring is needed.
- Comments are valuable for explaining where code cannot. For

example, the history behind why a certain method was implemented a specific way. Sometimes an unconventional or seemingly arbitrary approach may be applied because of some obscure external variable causing side effects. These things are difficult to explain with code.

# Docstrings

Docstring, or documentation strings, are valuable pieces of documentation that explain the functionality of any function or module in your code. Ideally, each of your functions should always have a docstring.

Docstrings are surrounded by triple quotes. The first line of the docstring is a brief explanation of the function's purpose.

**One-line docstring**

```
def population_density(population, land_area):
 """Calculate the population density of an area."""
    return population / land_area
```

If you think that the function is complicated enough to warrant a longer description, you can add a more thorough paragraph after the one-line summary.

# Multi-line docstring

```
def population_density(population, land_area):
 """Calculate the population density of an area.


    Args:
    population: int. The population of the area
    land_area: int or float. This function is unit-agnostic, if you pass in values
in terms of square km or square miles the function will return a density in
those units.


    Returns:
    population_density: population/land_area. The population density of a
    particular area.
    """
    return population / land_area
```

The next element of a docstring is an explanation of the function's arguments. Here, you list the arguments, state their purpose, and state what types the arguments should be. Finally, it is common to provide some description of the output of the function. Every piece of the docstring is optional; however, doc strings are a part of good coding practice.

# Version Control In Data Science

If you need a refresher on using Git for version control, check out the course

linked in the extracurriculars. If you're ready, let's see how Git is used in real data science scenarios!

**Scenario #1**

Let's walk through the Git commands that go along with each step in the scenario you just observed in the video.

**Step 1: You have a local version of this repository on your laptop, and to get the latest stable version, you pull from the develop branch.**

*Switch to the develop branch*

*git checkout develop*

*Pull the latest changes in the develop branch*

*git pull*

**Step 2: When you start working on this demographic feature, you create a new branch called demographic, and start working on your code in this branch.**

*Create and switch to a new branch called demographic from the develop branch*

*git checkout -b demographic*

*Work on this new feature and commit as you go*

*git commit -m 'added gender recommendations'*

*git commit -m 'added location specific recommendations'*

*...*

**Step 3: However, in the middle of your work, you need to work on another feature. So you commit your changes on this demographic branch, and switch back to the develop branch.**

*Commit your changes before switching*

*git commit -m 'refactored demographic gender and location recommendations '*

*Switch to the develop branch*

*git checkout develop*

**Step 4: From this stable develop branch, you create another branch for a new feature called friend_groups.**

*Create and switch to a new branch called friend_groups from the develop branch*

*git checkout -b friend_groups*

**Step 5: After you finish your work on the friend_groups branch, you commit your changes, switch back to the development branch, merge it**

**back to the develop branch, and push this to the remote repository's develop branch.**

*Commit your changes before switching*

*git commit -m 'finalized friend_groups recommendations '*

*Switch to the develop branch*

*git checkout develop*

*Merge the friend_groups branch into the develop branch*

*git merge --no-ff friends_groups*

*Push to the remote repository*

*git push origin develop*

**Step 6: Now, you can switch back to the demographic branch to continue your progress on that feature.**

*Switch to the demographic branch*

*git checkout demographic*

**Scenario #2**

Let's walk through the Git commands that go along with each step in the

scenario you just observed in the video.

**Step 1: You check your commit history, seeing messages about the changes you made and how well the code performed.**

*View the log history*

*git log*

**Step 2: The model at this commit seemed to score the highest, so you decide to take a look.**

*Check out a commit*

*git checkout bc90f2cbc9dc4e802b46e7a153aa106dc9a88560*

After inspecting your code, you realize what modifications made it perform well, and use those for your model.

**Step 3: Now, you're confident merging your changes back into the development branch and pushing the updated recommendation engine.**

*Switch to the develop branch*

*git checkout develop*

*Merge the friend_groups branch into the develop branch*

*git merge --no-ff friend_groups*

*Push your changes to the remote repository*

*git push origin develop*

**Scenario #3**

Let's walk through the Git commands that go along with each step in the scenario you just observed in the video.

**Step 1: Andrew commits his changes to the documentation branch, switches to the development branch, and pulls down the latest changes from the cloud on this development branch, including the change I merged previously for the friends group feature.**

*Commit the changes on the documentation branch*

*git commit -m "standardized all docstrings in process.py"*

*Switch to the develop branch*

*git checkout develop*

*Pull the latest changes on the develop branch down*

*git pull*

**Step 2: Andrew merges his documentation branch into the develop branch on his local repository, and then pushes his changes up to update the**

**develop branch on the remote repository.**

*Merge the documentation branch into the develop branch*

*git merge --no-ff documentation*

*Push the changes up to the remote repository*

*git push origin develop*

**Step 3: After the team reviews your work and Andrew's work, they merge the updates from the development branch into the master branch. Then, they push the changes to the master branch on the remote repository. These changes are now in production.**

*Merge the develop branch into the master branch*

*git merge --no-ff develop*

*Push the changes up to the remote repository*

*git push origin master*

**Note on merge conflicts**

For the most part, Git makes merging changes between branches really simple. However, there are some cases where Git can become confused about how to combine two changes, and asks you for help. This is called a merge conflict.

Mostly commonly, this happens when two branches modify the same file.

For example, in this situation, let's say you deleted a line that Andrew modified on his branch. Git wouldn't know whether to delete the line or modify it. You need to tell Git which change to take, and some tools even allow you to edit the change manually. If it isn't straightforward, you may have to consult with the developer of the other branch to handle a merge conflict.

# Model versioning

In the previous example, you may have noticed that each commit was documented with a score for that model. This is one simple way to help you keep track of model versions. Version control in data science can be tricky, because there are many pieces involved that can be hard to track, such as large amounts of data, model versions, seeds, and hyperparameters.

The following resources offer useful methods and tools for managing model versions and large amounts of data. These are here for you to explore, but are not necessary to know now as you start your journey as a data scientist. On the job, you'll always be learning new skills, and many of them will be specific to the processes set in your company.

# CHAPTER FIVE

# Software Engineering Practices, Part 2

In part 2 of software engineering practices, you'll learn about the following practices of software engineering and how they apply in data science.

- Testing
- Logging
- Code reviews

# Testing

Testing your code is essential before deployment. It helps you catch errors and faulty conclusions before they make any major impact. Today, employers are looking for data scientists with the skills to properly prepare their code for an industry setting, which includes testing their code.

**Testing And Data Science**

Problems that could occur in data science aren’t always easily detectable; you might have values being encoded incorrectly, features being used inappropriately, or unexpected data breaking assumptions.

To catch these errors, you have to check for the quality and accuracy of your analysis in addition to the quality of your code. Proper testing is necessary to avoid unexpected surprises and have confidence in your results.

Test-driven development (TDD): A development process in which you write tests for tasks before you even write the code to implement those tasks.

Unit test: A type of test that covers a “unit” of code—usually a single

function—independently from the rest of the program.

### Unit tests

We want to test our functions in a way that is repeatable and automated. Ideally, we'd run a test program that runs all our unit tests and cleanly lets us know which ones failed and which ones succeeded. Fortunately, there are great tools available in Python that we can use to create effective unit tests!

### Unit test advantages and disadvantages

The advantage of unit tests is that they are isolated from the rest of your program, and thus, no dependencies are involved. They don't require access to databases, APIs, or other external sources of information. However, passing unit tests isn’t always enough to prove that our program is working successfully. To show that all the parts of our program work with each other properly, communicating and transferring data between them correctly, we use integration tests. In this lesson, we'll focus on unit tests; however, when you start building larger programs, you will want to use integration tests as well.

To learn more about integration testing and how integration tests relate to unit tests, see Integration Testing. That article contains other very useful links as well.

### Unit Testing Tools

To install pytest, run pip install -U pytest in your terminal. You can see more information on getting started here.

- Create a test file starting with *test_*.
- Define unit test functions that start with *test_* inside the test file.
- Enter *pytest* into your terminal in the directory of your test file and it detects these tests for you.

*test_* is the default; if you wish to change this, you can learn how in this pytest configuration.

In the test output, periods represent successful unit tests and Fs represent failed unit tests. Since all you see is which test functions failed, it's wise to have only one *assert* statement per test. Otherwise, you won't know exactly how many tests failed or which tests failed.

Your test won't be stopped by failed assert statements, but it will stop if you have syntax errors.

# Logging

Logging is valuable for understanding the events that occur while running your program. For example, if you run your model overnight and the results the following morning are not what you expect, log messages can help you understand more about the context in those results occurred. Let's learn about the qualities that make a log message effective.

**Log messages**

Logging is the process of recording messages to describe events that have occurred while running your software. Let's take a look at a few examples, and learn tips for writing good log messages.

**Tip: Be professional and clear**

*Bad: Hmmm... this isn't working???*

*Bad: idk.... :(*

*Good: Couldn't parse file.*

***Tip: Be concise and use normal capitalization***

*Bad: Start Product Recommendation Process*

*Bad: We have completed the steps necessary and will now proceed with the recommendation process for the records in our product database.*

*Good: Generating product recommendations.*

**Tip: Choose the appropriate level for logging**

Debug: Use this level for anything that happens in the program. Error: Use this level to record any error that occurs. Info: Use this level to record all actions that are user driven or system specific, such as regularly scheduled operations.

**Tip: Provide any useful information**

*Bad: Failed to read location data*

*Good: Failed to read location data: store_id 8324971*

**Questions to ask yourself when conducting a code review**

First, let's look over some of the questions we might ask ourselves while reviewing code. These are drawn from the concepts we've covered in these last two lessons.

**Is the code clean and modular?**

- Can I understand the code easily?
- Does it use meaningful names and whitespace?
- Is there duplicated code?
- Can I provide another layer of abstraction?
- Is each function and module necessary?
- Is each function or module too long?

**Is the code efficient?**

- Are there loops or other steps I can vectorize?
- Can I use better data structures to optimize any steps?
- Can I shorten the number of calculations needed for any steps?
- Can I use generators or multiprocessing to optimize any steps?

**Is the documentation effective?**

- Are inline comments concise and meaningful?
- Is there complex code that's missing documentation?
- Do functions use effective docstrings?
- Is the necessary project documentation provided?

**Is the code well tested?**

- Does the code high test coverage?

- Do tests check for interesting cases?
- Are the tests readable?
- Can the tests be made more efficient?

**Is the logging effective?**

- Are log messages clear, concise, and professional?
- Do they include all relevant and useful information?
- Do they use the appropriate logging level?

## Tips for conducting a code review

Now that we know what we're looking for, let's go over some tips on how to actually write your code review. When your coworker finishes up some code that they want to merge to the team's code base, they might send it to you for review. You provide feedback and suggestions, and then they may make changes and send it back to you. When you are happy with the code, you approve it and it gets merged to the team's code base.

As you may have noticed, with code reviews you are now dealing with people, not just computers. So it's important to be thoughtful of their ideas and efforts. You are in a team and there will be differences in preferences. The goal of code review isn't to make all code follow your personal preferences, but to ensure it meets a standard of quality for the whole team.

### Tip: Use a code linter

This isn't really a tip for code review, but it can save you lots of time in a

code review. Using a Python code linter like pylint can automatically check for coding standards and PEP 8 guidelines for you. It's also a good idea to agree on a style guide as a team to handle disagreements on code style, whether that's an existing style guide or one you create together incrementally as a team.

### Tip: Explain issues and make suggestions

Rather than commanding people to change their code a specific way because it's better, it will go a long way to explain to them the consequences of the current code and suggest changes to improve it. They will be much more receptive to your feedback if they understand your thought process and are accepting recommendations, rather than following commands. They also may have done it a certain way intentionally, and framing it as a suggestion promotes a constructive discussion, rather than opposition.

***BAD****: Make model evaluation code its own module - too repetitive.*

***BETTER****: Make the model evaluation code its own module. This will simplify models.py to be less repetitive and focus primarily on building models.*

***GOOD****: How about we consider making the model evaluation code its own module? This would simplify models.py to only include code for building models. Organizing these evaluations methods into separate functions would also allow us to reuse them with different models without repeating code.*

### Tip: Keep your comments objective

Try to avoid using the words "I" and "you" in your comments. You want to

avoid comments that sound personal to bring the attention of the review to the code and not to themselves.

***BAD****: I wouldn't groupby genre twice like you did here... Just compute it once and use that for your aggregations.*

***BAD****: You create this groupby dataframe twice here. Just compute it once, save it as groupby_genre and then use that to get your average prices and views.*

***GOOD****: Can we group by genre at the beginning of the function and then save that as a groupby object? We could then reference that object to get the average prices and views without computing groupby twice.*

**Tip: Provide code examples**

When providing a code review, you can save the author time and make it easy for them to act on your feedback by writing out your code suggestions. This shows you are willing to spend some extra time to review their code and help them out. It can also just be much quicker for you to demonstrate concepts through code rather than explanations.

Let's say you were reviewing code that included the following lines:

```
first_names = []
last_names = []
```

```
for name in enumerate(df.name):
   first, last = name.split(' ')
   first_names.append(first)
   last_names.append(last)


df['first_name'] = first_names
df['last_names'] = last_names
```

BAD: You can do this all in one step by using the pandas str.split method.

GOOD: We can actually simplify this step to the line below using the pandas str.split method. Found this on this stack overflow post: https://stackoverflow.com/questions/14745022/how-to-split-a-column-into-two-columns

```
df['first_name'], df['last_name'] = df['name'].str.split(' ', 1).str
```

# Introduction to Object-Oriented Programming

# Procedural versus object-oriented programming

**Lesson outline**

- Object-oriented programming syntax
  - Procedural vs. object-oriented programming
  - Classes, objects, methods and attributes
  - Coding a class
  - Magic methods
  - Inheritance
- Using object-oriented programming to make a Python package
  - Making a package
  - Tour of scikit-learn source code
  - Putting your package on PyPi

**Why object-oriented programming?**

Object-oriented programming has a few benefits over procedural programming, which is the programming style you most likely first learned. As you'll see in this lesson:

- Object-oriented programming allows you to create large, modular programs that can easily expand over time.
- Object-oriented programs hide the implementation from the end user.

Consider Python packages like Scikit-learn, pandas, and NumPy. These are all Python packages built with object-oriented programming. Scikit-learn, for example, is a relatively large and complex package built with object-oriented programming. This package has expanded over the years with new functionality and new algorithms.

When you train a machine learning algorithm with Scikit-learn, you don't have to know anything about how the algorithms work or how they were coded. You can focus directly on the modeling.

Here's an example taken from the Scikit-learn website:

```
from sklearn import svm
X = [[0, 0], [1, 1]]
y = [0, 1]
clf = svm.SVC()
clf.fit(X, y)
```

How does Scikit-learn train the SVM model? You don't need to know because the implementation is hidden with object-oriented programming. If the implementation changes, you (as a user of Scikit-learn) might not ever find out. Whether or not you should understand how SVM works is a different question.

In this lesson, you'll practice the fundamentals of object-oriented programming. By the end of the lesson, you'll have built a Python package using object-oriented programming.

**Objects are defined by characteristics and actions**

Here is a reminder of what is a characteristic and what is an action.

*Objects are defined by their characteristics and their actions*

**Characteristics and actions in English grammar**

You can also think about characteristics and actions is in terms of English grammar. A characteristic corresponds to a noun and an action corresponds to a verb.

Let's pick something from the real world: a dog. Some characteristics of the dog include the dog's weight, color, breed, and height. These are all nouns. Some actions a dog can take include to bark, to run, to bite, and to eat. These are all verbs.

# Class, object, method, and attribute

Object-oriented programming (OOP) vocabulary

- *Class*: A blueprint consisting of methods and attributes.

- *Object*: An instance of a class. It can help to think of objects as something in the real world like a yellow pencil, a small dog, or a blue shirt. However, as you'll see later in the lesson, objects can be more abstract.
- *Attribute*: A descriptor or characteristic. Examples would be color, length, size, etc. These attributes can take on specific values like blue, 3 inches, large, etc.
- *Method*: An action that a class or object could take.
- *OOP*: A commonly used abbreviation for object-oriented programming.
- *Encapsulation*: One of the fundamental ideas behind object-oriented programming is called encapsulation: you can combine functions and data all into a single entity. In object-oriented programming, this single entity is called a class. Encapsulation allows you to hide implementation details, much like how the scikit-learn package hides the implementation of machine learning algorithms.

In English, you might hear an attribute described as a property, description, feature, quality, trait, or characteristic. All of these are saying the same thing.

Here is a reminder of how a class, an object, attributes, and methods relate to each other.

*A class is a blueprint consisting of attributes and methods.*

# OOP Syntax

**Object-oriented programming syntax**

In this video, you'll see what a class and object look like in Python. In the next section, you'll have the chance to play around with the code. Finally, you'll write your own class.

# Function versus method

In the video above, at 1:44, the dialogue mistakenly calls init a function rather than a method. Why is init not a function?

A function and a method look very similar. They both use the def keyword. They also have inputs and return outputs. The difference is that a method is inside of a class whereas a function is outside of a class.

**What is self?**

If you instantiate two objects, how does Python differentiate between these two objects?

*shirt_one = Shirt('red', 'S', 'short-sleeve', 15)*

*shirt_two = Shirt('yellow', 'M', 'long-sleeve', 20)*

That's where self comes into play. If you call the change_price method on shirt_one, how does Python know to change the price of shirt_one and not of shirt_two?

*shirt_one.change_price(12)*

Behind the scenes, Python is calling the change_price method:

```
def change_price(self, new_price):
    self.price = new_price
```

Self tells Python where to look in the computer's memory for the shirt_one object. Then, Python changes the price of the shirt_one object. When you call the change_price method, shirt_one.change_price(12), self is implicitly passed in.

The word self is just a convention. You could actually use any other name as long as you are consisten, but you should use self to avoid confusing people.

**Exercise: Use the Shirt class**

*Shirt class exercise*

You've seen what a class looks like and how to instantiate an object. Now it's your turn to write code that instantiates a shirt object.

You need to download three files for this exercise. These files are located on this page in the Supporting materials section.

- *Shirt_exercise.ipynb contains explanations and instructions.*
- *Answer.py containing solution to the exercise.*
- *Tests.py tests for checking your code: You can run these tests using the last code cell at the bottom of the notebook.*

**Getting started**

Open the Shirt Exercise.ipynb notebook file using Jupyter Notebook and follow the instructions in the notebook to complete the exercise.

# Notes about OOP

**Set and get methods**

The last part of the video mentioned that accessing attributes in Python can be somewhat different than in other programming languages like Java and C++. This section goes into further detail.

The Shirt class has a method to change the price of the shirt: shirt_one.change_price(20). In Python, you can also change the values of an attribute with the following syntax:

*shirt_one.price = 10*

*shirt_one.price = 20*

*shirt_one.color = 'red'*

*shirt_one.size = 'M'*

*shirt_one.style = 'long_sleeve'*

This code accesses and changes the price, color, size, and style attributes directly. Accessing attributes directly would be frowned upon in many other languages, but not in Python. Instead, the general object-oriented programming convention is to use methods to access attributes or change attribute values. These methods are called set and get methods or setter and getter methods.

A get method is for obtaining an attribute value. A set method is for changing an attribute value. If you were writing a Shirt class, you could use the following code:

```
class Shirt:
    def __init__(self, shirt_color, shirt_size, shirt_style, shirt_price):
        self._price = shirt_price
```

```
    def get_price(self):
      return self._price


    def set_price(self, new_price):
      self._price = new_price
```

Instantiating and using an object might look like the following code:

```
shirt_one = Shirt('yellow', 'M', 'long-sleeve', 15)
print(shirt_one.get_price())
shirt_one.set_price(10)
```

In the class definition, the underscore in front of price is a somewhat controversial Python convention. In other languages like C++ or Java, price could be explicitly labeled as a private variable. This would prohibit an object from accessing the price attribute directly like shirt_one._price = 15. Unlike other languages, Python does not distinguish between private and public variables. Therefore, there is some controversy about using the underscore convention as well as get and set methods in Python. Why use get and set methods in Python when Python wasn't designed to use them?

At the same time, you'll find that some Python programmers develop object-oriented programs using get and set methods anyway. Following the Python convention, the underscore in front of price is to let a programmer know that price should only be accessed with get and set methods rather than

accessing price directly with shirt_one._price. However, a programmer could still access _price directly because there is nothing in the Python language to prevent the direct access.

To reiterate, a programmer could technically still do something like shirt_one._price = 10, and the code would work. But accessing price directly, in this case, would not be following the intent of how the Shirt class was designed.

One of the benefits of set and get methods is that, as previously mentioned in the course, you can hide the implementation from your user. Perhaps, originally, a variable was coded as a list and later became a dictionary. With set and get methods, you could easily change how that variable gets accessed. Without set and get methods, you'd have to go to every place in the code that accessed the variable directly and change the code.

You can read more about get and set methods in Python on this Python Tutorial site.

**Attributes**

There are some drawbacks to accessing attributes directly versus writing a method for accessing attributes.

In terms of object-oriented programming, the rules in Python are a bit looser than in other programming languages. As previously mentioned, in some languages, like C++, you can explicitly state whether or not an object should be allowed to change or access an attribute's values directly. Python does not have this option.

Why might it be better to change a value with a method instead of directly? Changing values via a method gives you more flexibility in the long-term. What if the units of measurement change, like if the store was originally meant to work in US dollars and now has to handle Euros? Here's an example:

**Example: Dollars versus Euros**

If you've changed attribute values directly, you'll have to go through your code and find all the places where US dollars were used, such as in the following:

```
shirt_one.price = 10 # US dollars
```

Then, you'll have to manually change them to Euros.

```
shirt_one.price = 8 # Euros
```

If you had used a method, then you would only have to change the method to convert from dollars to Euros.

```
def change_price(self, new_price):
   self.price = new_price * 0.81 # convert dollars to Euros
shirt_one.change_price(10)
```

For the purposes of this introduction to object-oriented programming, you don't need to worry about updating attributes directly versus with a method; however, if you decide to further your study of object-oriented programming, especially in another language such as C++ or Java, you'll have to take this into consideration.

**Modularized code**

Thus far in the lesson, all of the code has been in Jupyter Notebooks. For example, in the previous exercise, a code cell loaded the Shirt class, which gave you access to the shirt class throughout the rest of the notebook.

If you were developing a software program, you would want to modularize this code. You would put the Shirt class into its own Python script, which you might call shirt.py. In another Python script, you would import the Shirt class with a line like from shirt import Shirt.

For now, as you get used to OOP syntax, you'll be completing exercises in Jupyter Notebooks. Midway through the lesson, you'll modularize object-oriented code into separate files.

**Exercise: Use the Pants class**

Now that you've had some practice instantiating objects, it's time to write your own class from scratch.

This lesson has two parts.

- In the first part, you'll write a Pants class. This class is similar to

the Shirt class with a couple of changes. Then you'll practice instantiating Pants objects.

- In the second part, you'll write another class called SalesPerson. You'll also instantiate objects for the SalesPerson.

This exercise requires two files, which are located on this page in the Supporting Materials section.

- *exercise.ipynbcontains explanations and instructions.*
- *answer.py contains solution to the exercise.*

**Getting started**

Open the exercise.ipynb notebook file using Jupyter Notebook and follow the instructions in the notebook to complete the exercise

# Commenting object-oriented code

Did you notice anything special about the answer key in the previous exercise? The Pants class and the SalesPerson class contained docstrings! A docstring is a type of comment that describes how a Python module, function, class, or method works. Docstrings are not unique to object-oriented programming.

For this section of the course, you just need to remember to use docstrings and to comment your code. It will help you understand and maintain your code and even make you a better job candidate.

From this point on, please always comment your code. Use both inline comments and document-level comments as appropriate.

To learn more about docstrings, see Example Google Style Python Docstrings.

## Docstrings and object-oriented code

The following example shows a class with docstrings. Here are a few things to keep in mind:

- Make sure to indent your docstrings correctly or the code will not run. A docstring should be indented one indentation underneath the class or method being described.
- You don't have to define self in your method docstrings. It's understood that any method will have self as the first method input.

```
class Pants:
    """The Pants class represents an article of clothing sold in a store
    """


    def __init__(self, color, waist_size, length, price):
        """Method for initializing a Pants object


        Args:
            color (str)
            waist_size (int)
            length (int)
```

```
            price (float)

        Attributes:
            color (str): color of a pants object
            waist_size (str): waist size of a pants object
            length (str): length of a pants object
            price (float): price of a pants object
        """

        self.color = color
        self.waist_size = waist_size
        self.length = length
        self.price = price

    def change_price(self, new_price):
        """The change_price method changes the price attribute of a pants
object

        Args:
            new_price (float): the new price of the pants object
```

```
        Returns: None

        """
        self.price = new_price

    def discount(self, percentage):
        """The discount method outputs a discounted price of a pants object

        Args:
            percentage (float): a decimal representing the amount to discount

        Returns:
            float: the discounted price
        """
        return self.price * (1 - percentage)
```

# A Gaussian class

Resources for review

The example in the next part of the lesson assumes you are familiar with Gaussian and binomial distributions.

Here are a few formulas that might be helpful:

Gaussian distribution formulas

probability density function