AWS ML COURSE

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.

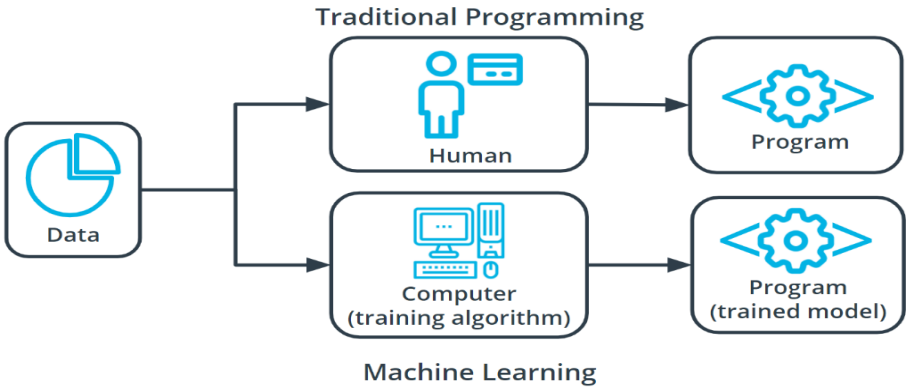
Course Overview

- **Lesson 2: Introduction to Machine Learning** – In this lesson, you will learn the fundamentals of supervised and unsupervised machine learning, including the process steps of solving machine learning problems, and explore some real world examples.
- **Lesson 3: Machine Learning with AWS** – In this lesson, you will learn about advanced machine learning techniques such as generative AI, reinforcement learning, and computer vision. You will also learn how to train these models with AWS AI/ML services.
- **Lesson 4: Software Engineering Practices, part 1** – In this lesson, you will learn how to write well-documented, modularized code.
- **Lesson 5: Software Engineering Practices, part 2** – In this lesson, you will learn how to test your code and log best practices.
- **Lesson 6: Object-Oriented Programming –** In this lesson, you will learn about this programming style and prepare to write your own Python package.

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.

- 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. We will explore this in-depth in this lesson.
- 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. We will explore this in-depth in this lesson.
- 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
- Using a **deep neural network** to detect chemical spills in a lab from video and images

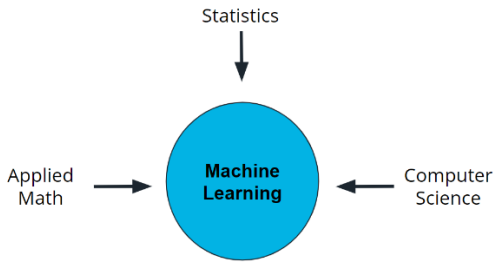


Traditional programming versus machine learning

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.

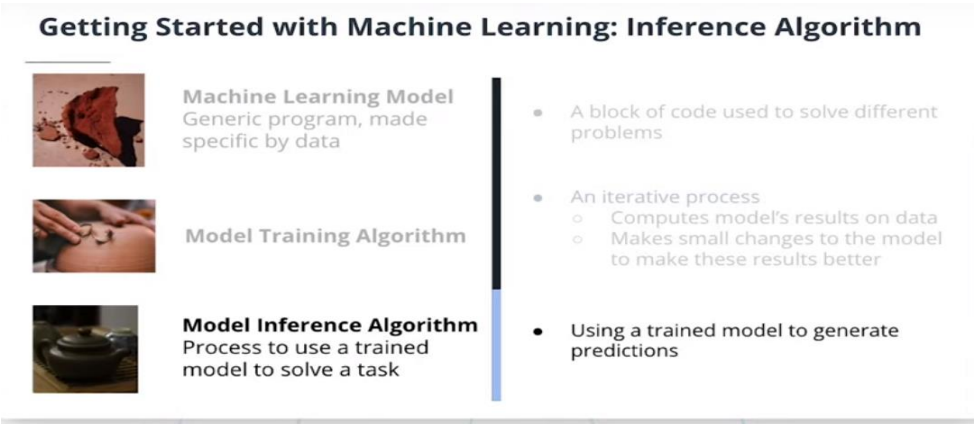
Understanding Terminology



Fields that influence machine learning

What are machine learning models?

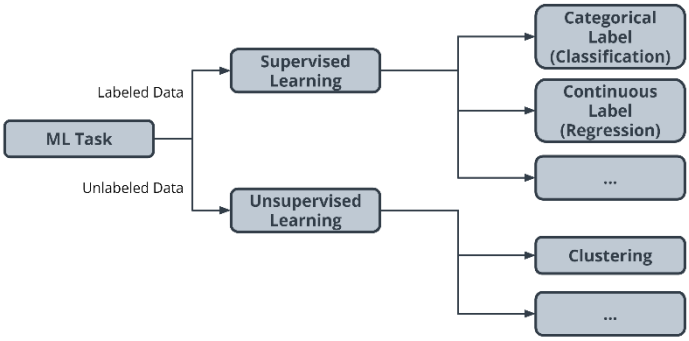
A machine learning model, like a piece of clay, can be molded into many different forms and serve many different purposes. A more technical definition would be that a machine learning model is a block of code or framework that can be modified to solve different but related problems based on the data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



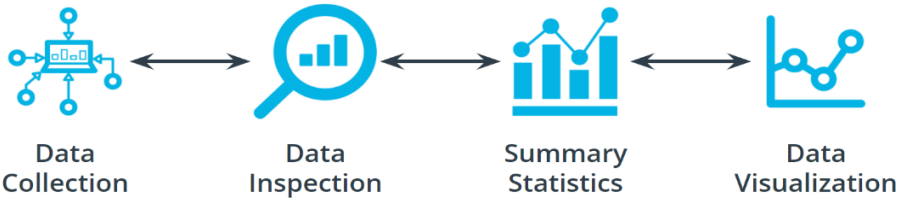
MACHINE LEARNING COMPONENT	DEFINITION
Machine learning model	Generic program, made specific by data
Model training algorithm	An iterative process fitting a generic model to specific data
Model inference algorithm	Process to use a trained model to solve a task

MAJOR STEPS OF PROBLEM IN ML MODEL

STEP - 1



STEP - 2



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.

Data inspection is quality of your data will ultimately be the largest factor that affects how well you can expect your model to perform.

Summary statistics is the Models can assume how your data is structured.

Data visualization to see outliers and trends in your data and to help stakeholders understand your data.

- *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.

Splitting your Dataset

- *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.

STEP - 3

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*: they 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 :* It 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

Extended Learning

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:

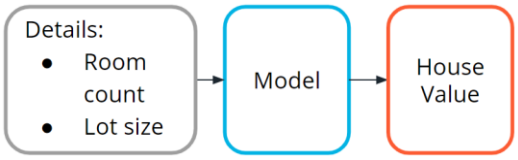
- **FFNN**: The most straightforward way of structuring a neural network, the Feed Forward Neural Network (FFNN) structures neurons in a series of layers, with each neuron in a layer containing weights to all neurons in the previous layer.
- **CNN**: Convolutional Neural Networks (CNN) represent nested 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.

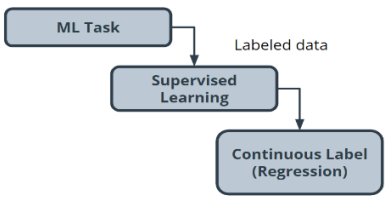
STEP – 4

Model accuracy is a fair evaluation metric. *Accuracy* is the fraction of predictions a model gets right.

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.
****

EXAMPLE 1 **House price prediction**



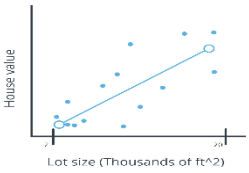


STEP One: Defining the problem

Step Two: Building a Dataset

- **Data collection**: You collect numerous examples of homes sold in your neighbourhood 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.
- **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.



# of Rooms	Lot Size (ft²)	House Value ($)
4	10,454	339,900
3	9,147	239,000
3	10,890	250,000
~~10~~	~~25,877~~	~~877,000~~

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.

Step Four: Evaluation

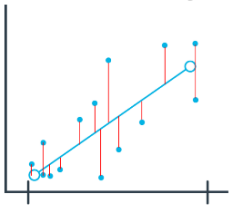
One of the most common evaluation metrics in a regression scenario is called *root mean square* or *RMS*. The math is beyond the scope of this lesson, but RMS can be thought of roughly as the "average error" across your test dataset, so you want this value to be low.

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

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.



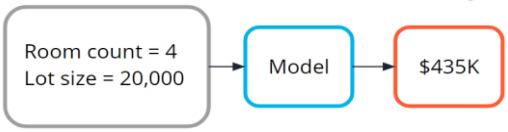
Interpreting Results

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.

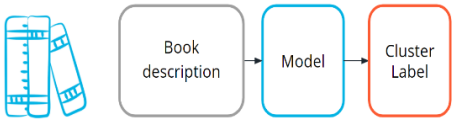
Step Five: Inference: Try out your model

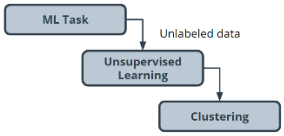
Now you are ready to put your model into action. As you can see in the following image, this means seeing how well it predicts with new data not seen during model training.



EXAMPLE 2 -Book Genre Exploration

Step One: Define the Problem



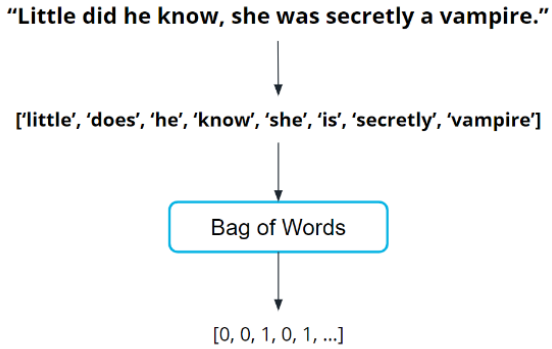


Step Two: Build your Dataset

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

Before you can train the model, you need to do some data pre-processing , called *data vectorization*, to convert text into numbers.

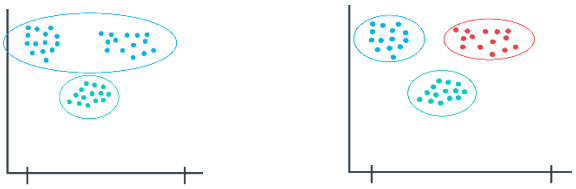
You transform this book description text into what is called a `bag of words` representation shown in the following image so that it is understandable by machine learning models.



Step Three: Train the 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.

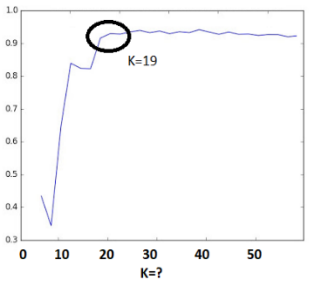
Your data is unlabelled you don't how many microgenres might exist. So you train your model multiple times using different values for `k` each time.



K=2 K=3

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`.



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..."

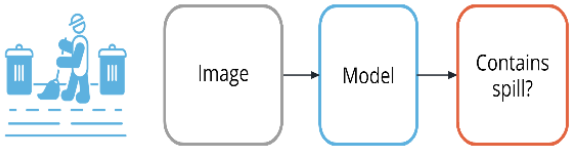
Clustered data

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.

****

EXAMPLE 3 Spill Detection from Video

Step One: Defining the Problem

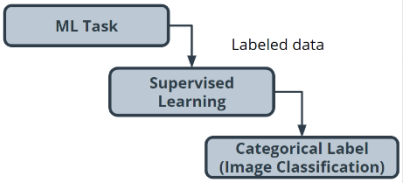


Detecting spills with machine learning

Step Two: Model Training (and selection)

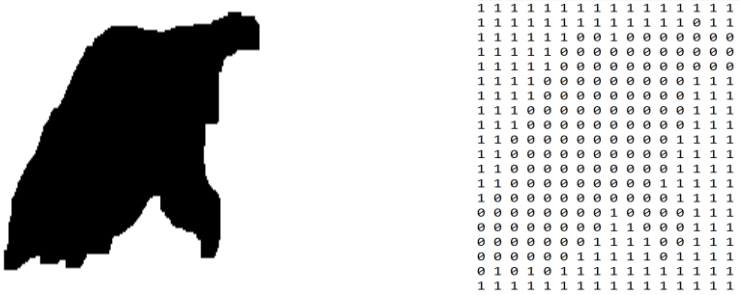
This task is a supervised classification task, as shown in the following image. As shown in the image above, your goal will be to predict if each image belongs to one of the following classes:

- Contains spill
- Does not contain spill



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.



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

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:

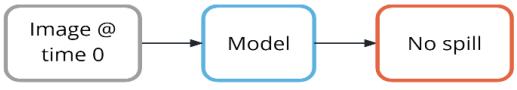
Accuracy	False positive rate	Precision
Confusion matrix	False negative rate	Recall
F1 Score	Log Loss	ROC curve
	Negative predictive value	Specificity

The common problem is 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

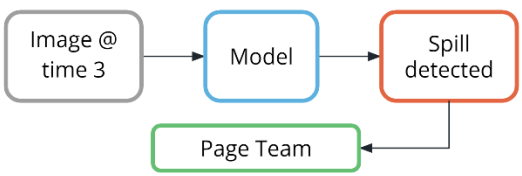
The model can be deployed on a system that enables you to run machine learning workloads such as AWS Panorama.

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



No spill detected

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



Spill detecte

GLOSSARY

1. **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.
2. A **categorical** label has a discrete set of possible values, such as "is a cat" and "is not a cat."
3. **Clustering**. Unsupervised learning task that helps to determine if there are any naturally occurring groupings in the data.
4. **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.
5. A **continuous (regression)** label does not have a discrete set of possible values, which means possibly an unlimited number of possibilities.
6. **Data vectorization**: A process that converts non-numeric data into a numerical format so that it can be used by a machine learning model.
7. **Discrete**: A term taken from statistics referring to an outcome taking on only a finite number of values (such as days of the week).
8. **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.
9. **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.
10. **Log loss** is used to calculate how uncertain your model is about the predictions it is generating.
11. **Hyperplane**: A mathematical term for a surface that contains more than two planes.
12. **Impute** is a common term referring to different statistical tools which can be used to calculate missing values from your dataset.
13. **label** refers to data that already contains the solution.
14. **loss function** is used to codify the model's distance from this goal
15. **Machine learning**, or ML, is a modern software development technique that enables computers to solve problems by using examples of real-world data.
16. **Model accuracy** is the fraction of predictions a model gets right. Discrete: A term taken from statistics referring to an outcome taking on only a finite number of values (such as days of the week). 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.
17. **Model inference** is when the trained model is used to generate predictions.
18. model is an extremely generic program, made specific by the data used to train it.
19. **Model parameters** are settings or configurations the training algorithm can update to change how the model behaves.

20. **Model training algorithms** work through an interactive process where the current model iteration is analysed 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.
21. **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**.
22. **Outliers** are data points that are significantly different from others in the same sample.
23. **Plane**: A mathematical term for a flat surface (like a piece of paper) on which two points can be joined by a straight line.
24. **Regression**: A common task in supervised machine learning.
25. 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.
26. **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.
27. **Silhouette coefficien**t: A score from -1 to 1 describing the clusters found during modelling. A score near zero indicates overlapping clusters, and scores less than zero indicate data points assigned to incorrect clusters. A
28. **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.
29. 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.
30. **Test dataset**: The data withheld from the model during training, which is used to test how well your model will generalize to new data.
31. **Training dataset**: The data on which the model will be trained. Most of your data will be here.
32. **Transformer**: A more modern replacement for RNN/LSTMs, the transformer architecture enables training over larger datasets involving sequences of data.
33. In **unlabelled data**, you don't need to provide the model with any kind of label or solution while the model is being trained.
34. 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.

Computer vision (CV) has many real-world applications. We cover examples of image classification, object detection, semantic segmentation, and activity recognition.

- ***Image classification*** is the most common application of computer vision in use today. Image classification can be used to answer questions like *What's in this image?* This type of task has applications in *text detection* or *optical character recognition (OCR)* and *content moderation*.
- ***Object detection*** is closely related to image classification, but it allows users to gather more granular detail about an image. For example, rather than just knowing whether an object is present in an image, a user might want to know if there are *multiple instances of the same object* present in an image, or if *objects from different classes* appear in the same image.
- ***Semantic segmentation*** is another common application of computer vision that takes a pixel-by-pixel approach. Instead of just identifying whether an object is present or not, it tries to identify down the pixel level which part of the image is part of the object.
- ***Activity recognition*** is an application of computer vision that is based around videos rather than just images. Video has the added dimension of time and, therefore, models are able to detect changes that occur over time.
- **Input Layer**: The first layer in a neural network. This layer receives all data that passes through the neural network.
- **Hidden Layer**: A layer that occurs between the output and input layers. Hidden layers are tailored to a specific task.
- **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.

AWS Deep Lens

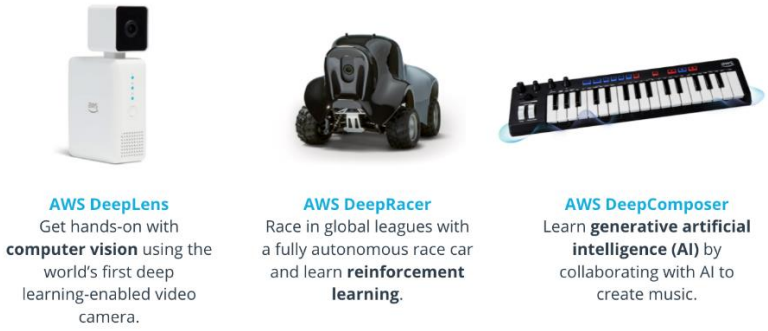
AWS Deep Lens allows you to create and deploy end-to-end computer vision–based applications. The following video provides a brief introduction to how AWS Deep Lens works and how it uses other AWS services.

AWS Deep Lens is a **deep learning–enabled camera** that allows you to deploy trained models directly to the device. You can either use sample templates and recipes or train your own model. AWS Deep Lens is integrated with several AWS machine learning services and can perform local inference against deployed models provisioned from the AWS Cloud. It enables you to learn and explore the latest artificial intelligence (AI) tools and techniques for developing computer vision applications based on a deep learning model.

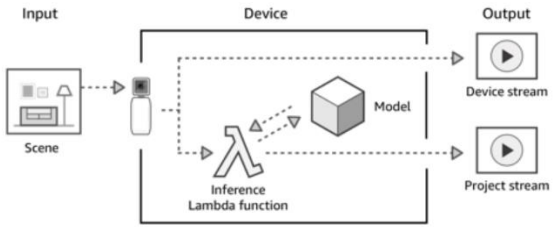
The AWS Deep Lens device

The AWS Deep Lens camera is powered by an Intel® Atom processor, which can process 100 billion floating-point operations per second (GFLOPS). This gives you all the computing power you need to perform inference on your device. The micro HDMI display port, audio out, and USB ports allow you to attach peripherals, so you can get creative with your computer vision applications.

You can use AWS DeepLens as soon as you register it.



An AWS DeepLens Device



How AWS Deep Lens works

How AWS Deep Lens works

AWS Deep Lens is integrated with multiple AWS services. You use these services to create, train, and launch your AWS DeepLens project. You can think of an AWS Deep Lens project as being divided into two different streams as the image shown above.

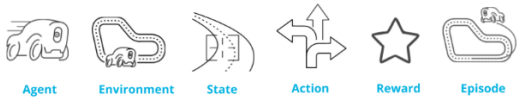
- First, you use the AWS console to create your project, store your data, and train your model.
- Then, you use your trained model on the AWS Deep Lens device. On the device, the video stream from the camera is processed, inference is performed, and the output from inference is passed into two output streams:
    - **Device stream** – The video stream passed through without processing.
    - **Project stream** – The results of the model's processing of the video frames.

Reinforcement Learning with AWS DeepRacer

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.

- 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:



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.

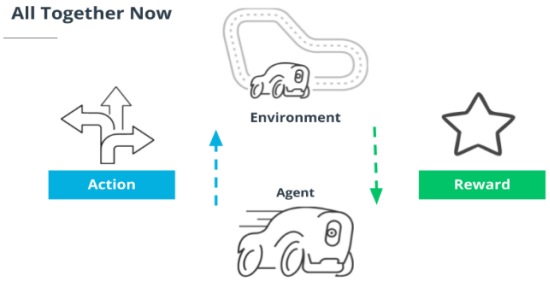
Reward

- 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

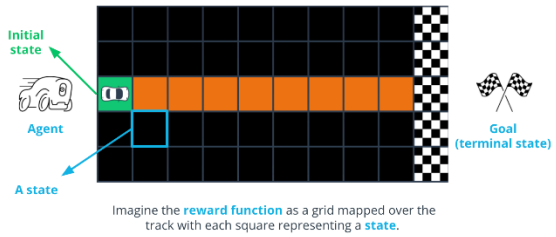
- An episode represents a period of trial and error when an agent makes decisions and gets feedback from its environment.
- For AWS DeepRacer, an episode begins at the initial state, when the car leaves the starting position, and ends at the terminal state, when it finishes a lap, bumps into an obstacle,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

on a graph. It also introduced the trade-off between exploration and exploitation, an important challenge unique to this type of machine learning.



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.

Computer vision (CV) has many real-world applications. In this video, we cover examples of image classification, object detection, semantic segmentation, and activity recognition. Here's a brief summary of what you learn about each topic in the video:

- ***Image classification*** is the most common application of computer vision in use today. Image classification can be used to answer questions like *What's in this image?* This type of task has applications in *text detection* or *optical character recognition (OCR)* and *content moderation*.
- ***Object detection*** is closely related to image classification, but it allows users to gather more granular detail about an image. For example, rather than just knowing whether

an object is present in an image, a user might want to know if there are *multiple instances of the same object* present in an image, or if *objects from different classes* appear in the same image.

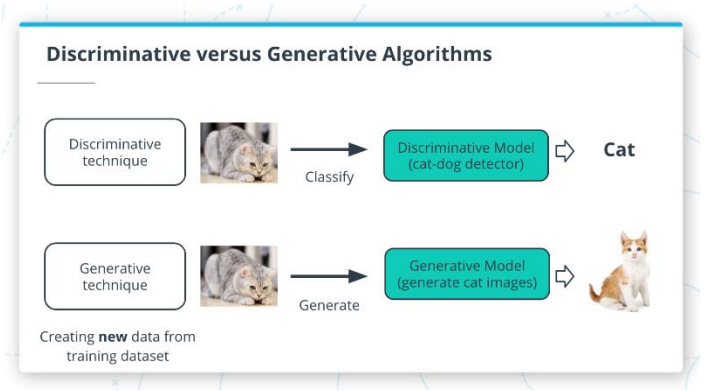
- ***Semantic segmentation*** is another common application of computer vision that takes a pixel-by-pixel approach. Instead of just identifying whether an object is present or not, it tries to identify down the pixel level which part of the image is part of the object.
- ***Activity recognition*** is an application of computer vision that is based around videos rather than just images. Video has the added dimension of time and, therefore, models are able to detect changes that occur over time.

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.

What are GANs?

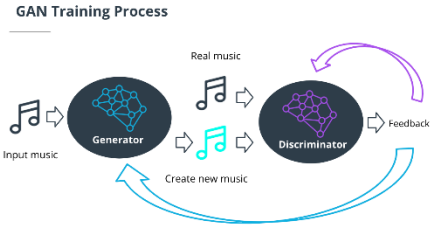
A GAN is a type of generative machine learning model which pits two neural networks against each other to generate new content: a generator and a discriminator.

- 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.

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.

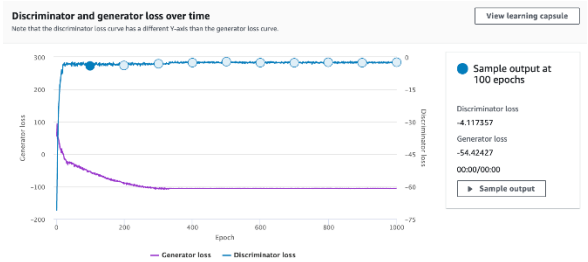
Generator

- 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).

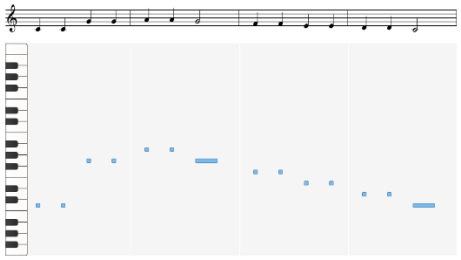


Discriminator loss and generator loss reach a plateau

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.

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.

Welcome To Software Engineering Practices

- *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*: 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.

Writing clean code: 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.
- B*e 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.

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.

Project Documentation

Project documentation is essential for getting others to understand why and how your code is relevant to them, whether they are potentials users of your project or developers who may contribute to your code. A great first step in project documentation is your README file. It will often be the first interaction most users will have with your project.

Whether it's an application or a package, your project should absolutely come with a README file. At a minimum, this should explain what it does, list its dependencies, and provide sufficiently detailed instructions on how to use it. Make it as simple as possible for others to understand the purpose of your project and quickly get something working.

Translating all your ideas and thoughts formally on paper can be a little difficult, but you'll get better over time, and doing so makes a significant difference in helping others realize the value of your project. Writing this documentation can also help you improve the design of your code, as you're forced to think through your design decisions more thoroughly. It also helps future contributors to follow your original intentions.

GIT

STEP -1

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
```

STEP -2

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
```

STEP -3

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
```

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.

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.

Test-driven development and data science

- *Test-driven development*: Writing tests before you write the code that's being tested. Your test fails at first, and you know you've finished implementing a task when the test passes.
- Tests can check for different scenarios and edge cases before you even start to write your function. When start implementing your function, you can run the test to get immediate feedback on whether it works or not as you tweak your function.
- When refactoring or adding to your code, tests help you rest assured that the rest of your code didn't break while you were making those changes. Tests also helps ensure that your function behavior is repeatable, regardless of external parameters such as hardware and time.

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
```

NEXT

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?
- 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)
```

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

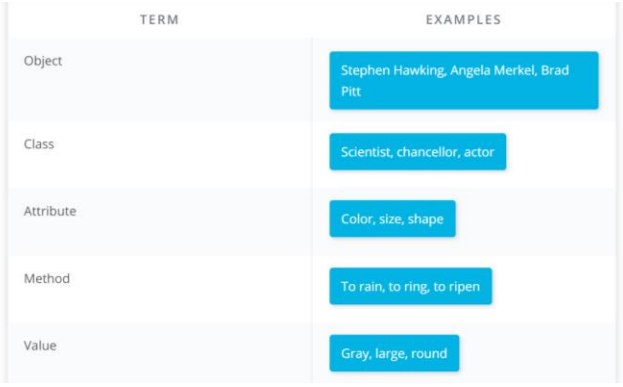
Object-oriented programming (OOP) vocabulary

- *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.



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.

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.

NEXT

; 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.ipynb` contains explanations and instructions.
- `answer.py` contains solution to 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)
```

NEXT

;

Gaussian distribution formulas

probability density function

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-} \frac{(x-\mu)^2}{2\sigma^2}$$

where: μ is the mean σ is the standard deviation σ^2 is the variance

Binomial distribution formulas

mean

$$\mu = n * p$$

In other words, a fair coin has a probability of a positive outcome (heads) p = 0.5. If you flip a coin 20 times, the mean would be 20 * 0.5 = 10; you'd expect to get 10 heads.

variance

$$\sigma^2 = np(1-p)$$

Continuing with the coin example, n would be the number of coin tosses and p would be the probability of getting heads.

standard deviation

$$\sigma = \sqrt{np(1-p)}$$

In other words, the standard deviation is the square root of the variance.

probability density function

$$f(k, n, p) = \frac{n!}{k!(n-k)!} p^k (1-p)^{(n-k)}$$

Exercise: Code magic methods

Extend the code from the previous exercise by using two new methods, `add` and `repr`. This exercise requires three files, which are located on this page in the **Supporting materials** section.

- `Magic_methods.ipynb` contains explanations and instructions.
- `Answer.py` contains the solution to the exercise.
- `Numbers.txt` can be read in by the read_data_file() method.

Demo: Modularized code

This is a code demonstration, so you do not need to write any code.

So far, the coding exercises have been in Jupyter Notebooks. Jupyter Notebooks are especially useful for data science applications because you can wrangle data, analyze data, and share a report all in one document. However, they're not ideal for writing modular programs, which require separating code into different files.

At the bottom of this page under **Supporting materials**, download three files.

- `Gaussiandistribution.py`
- `Generaldistribution.py`
- `example_code.py`

Look at how the distribution class and Gaussian class are modularized into different files.

The `Gaussiandistribution.py` imports the `Distribution` class from the `Generaldistribution.py file`. Note the following line of code:

```
from Generaldistribution import Distribution
```

This code essentially pastes the distribution code to the top of the `Gaussiandistribution` file when you run the code. You can see in the `example_code.py` file an example of how to use the Gaussian class.
The `example_code.py` file then imports the Gaussian distribution class.
For the rest of the lesson, you'll work with modularized code rather than a Jupyter Notebook. Go through the code in the `modularized_code` folder to understand how everything is organized.

Advanced OOP topics

Inheritance is the last object-oriented programming topic in the lesson. Thus far you've been exposed to:

- Classes and objects
- Attributes and methods
- Magic methods
- Inheritance

Classes, object, attributes, methods, and inheritance are common to all object-oriented programming languages.

Knowing these topics is enough to start writing object-oriented software. What you've learned so far is all you need to know to complete this OOP lesson. However, these are only the fundamentals of object-oriented programming.

What is `pip`?

`pip` is a Python package manager that helps with installing and uninstalling Python packages. You might have used `pip` to install packages using the command line: `pip install numpy`. When you execute a command like `pip install numpy`, `pip` downloads the package from a Python package repository called PyPi.
For this next exercise, you'll use `pip` to install a Python package from a local folder on your computer. The last part of the lesson will focus on uploading packages to PyPi so that you can share your package with the world.
You can complete this entire lesson within the classroom using the provided workspaces; however, if you want to develop a package locally on your computer, you should consider setting up a virtual environment. That way, if you install your package on your computer, the package won't install into your main Python installation. Before starting the next exercise, the next part of the lesson will discuss what virtual environments are and how to use them.

Object-oriented programming and Python packages

A Python package does not need to use object-oriented programming. You could simply have a Python module with a set of functions. However, most—if not all—of the popular Python packages take advantage of object-oriented programming for a few reasons:

1. Object-oriented programs are relatively easy to expand, especially because of inheritance.

2. Object-oriented programs obscure functionality from the user. Consider `scipy` packages. You don't need to know how the actual code works in order to use its classes and methods.

Exercise: Binomial class

In this exercise, you'll extend the distributions package with a new class called `Binomial`.
In the **Supporting materials** section of this page, there is a .zip file called
called `4a_binomial_package.zip`. Download and unzip this file.
Inside the folder called `4a_binomial_package`, there is another folder and these files:

- `distributions`, which contains the code for the distributions package including `Gaussiandistribution.py` and `Generaldistribution.py` code.
- `setup.py`, a file needed for building Python packages with `pip`.
- `test.py` unit tests to help you debug your code.
- `numbers.txt` and `numbers_binomial.txt`, which are data files used as part of the unit tests.
- `Binomialdistribution.py` and `Binomialdistribution_challenge.py`. Choose one of these files for completing the exercise. `Binomialdistribution.py` includes more of the code already set up for you. In `Binomialdistribution_challenge.py`, you'll have to write all of the code from scratch. Both files contain instructions with TODOS to fill out.

In these files, you only need to change the following:

- `__init__.py`, inside the distributions folder. You need to import the binomial package.
- Either `Binomialdistribution.py` or `Binomialdistribution_challenge.py` You also need to put your `Binomialdistribution.py` file into the distributions folder.

When you're ready to test out your code, follow these steps:

1. **`pip`** install your distributions package**. In the terminal, make sure you are in the `4a_binomial_package` directory. If not, navigate there by entering the following at the command line:

```
cd 4a_binomial_package
pip install
```

2. **Run the unit tests**. Enter the following.

```
python -m unittest test
```

Modify the `Binomialdistribution.py` code until all the unit tests pass.
If you change the code in the distributions folder after `pip` installing the package, Python will not know about the changes.
When you make changes to the package files, you'll need to run the following:

```
pip install --upgrade
```

PyPi vs. test PyPi

Note that pypi.org and test.pypy.org are two different websites. You'll need to register separately at each website. If you only register at pypi.org, you will not be able to upload to the test.pypy.org repository.
Remember that your package name must be unique. If you use a package name that is already taken, you will get an error when trying to upload the package.

Summary of the terminal commands used in the video

```
cd binomial_package_files
python setup.py sdist
pip install twine

# commands to upload to the pypi test repository
twine upload --repository-url https://test.pypi.org/legacy/ dist/*
pip install --index-url https://test.pypi.org/simple/ dsnd-probability

# command to upload to the pypi repository
twine upload dist/*
pip install dsnd-probability
```

More PyPi resources

This tutorial explains how to distribute Python packages, including more configuration options for your `setup.py` file. You'll notice that the Python command to run the `setup.py` is slightly different, as shown in the following example:

```
python3 setup.py sdist bdist_wheel
```

This command still outputs a folder called `dist`. The difference is that you will get both a .tar.gz file and a .whl file. The .tar.gz file is called a *source archive*, whereas the .whl file is a *built distribution*. The .whl file is a newer type of installation file for Python packages. When you `pip` install a package, `pip` firsts look for a .whl file (wheel file); if there isn't one, it looks for the .tar.gz file.
A .tar.gz file (an `sdist`) contains the files needed to compile and install a Python package. A .whl file (a *built distribution*) only needs to be copied to the proper place for installation. Behind the scenes, `pip`installing a .whl file has fewer steps than installing a .tar.gz file.