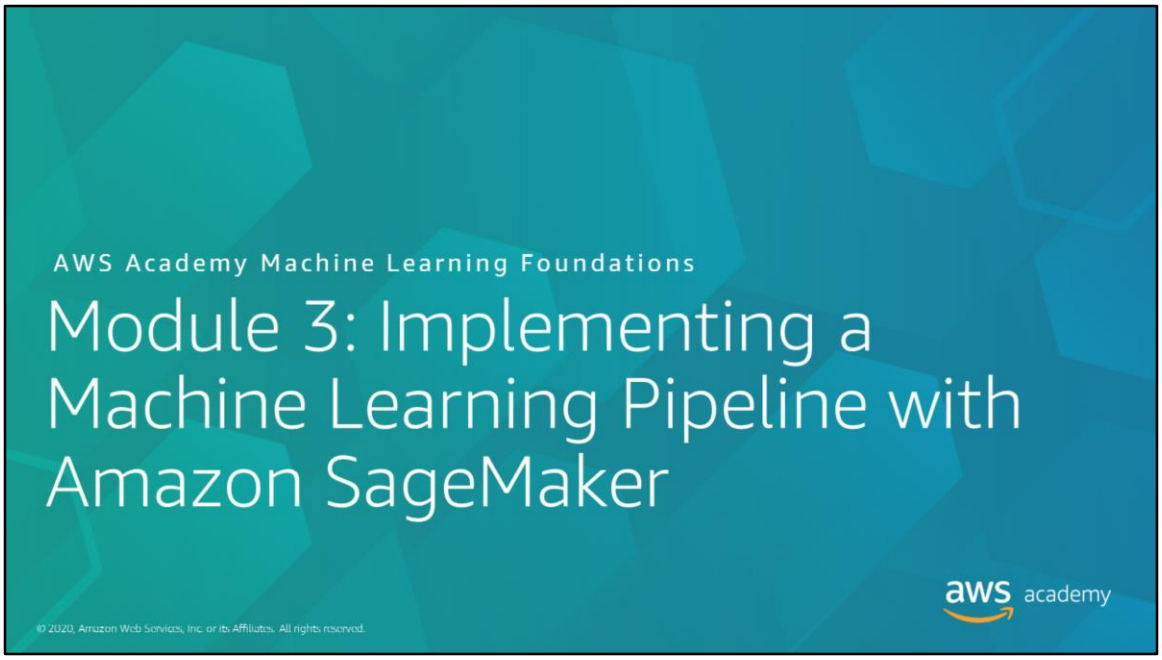
AWS Academy Machine Learning Foundations
Module 03 Student Guide
Version 1.0.3

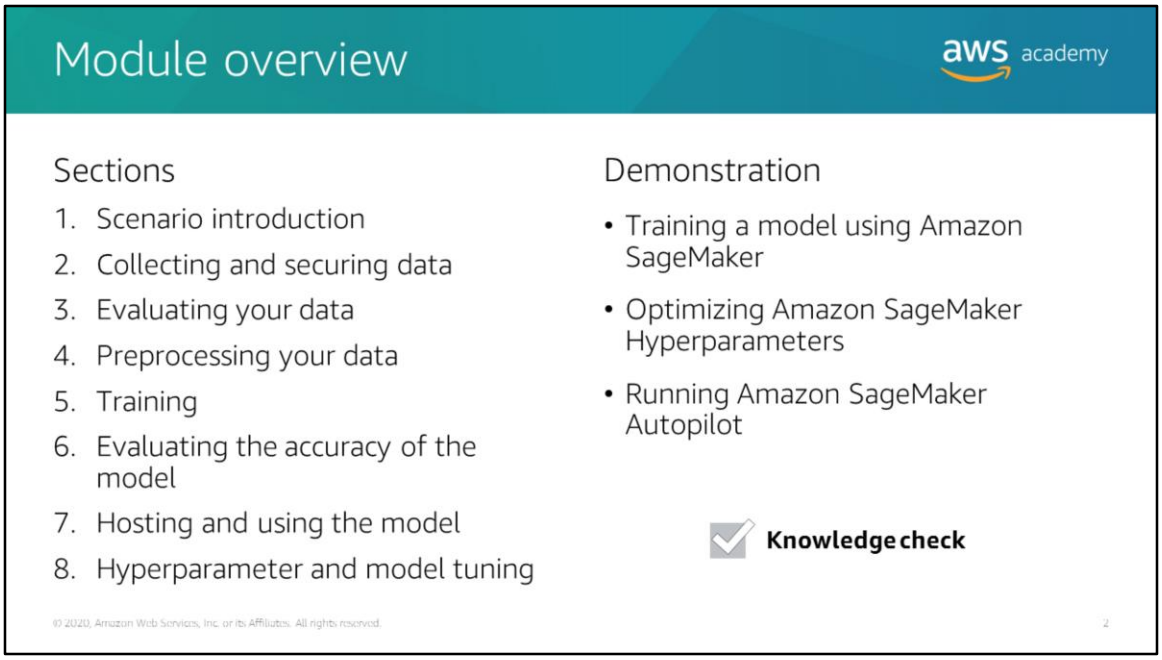
200-ACMLFO-10-EN-SG

AWS Academy Machine Learning Foundations

Contents



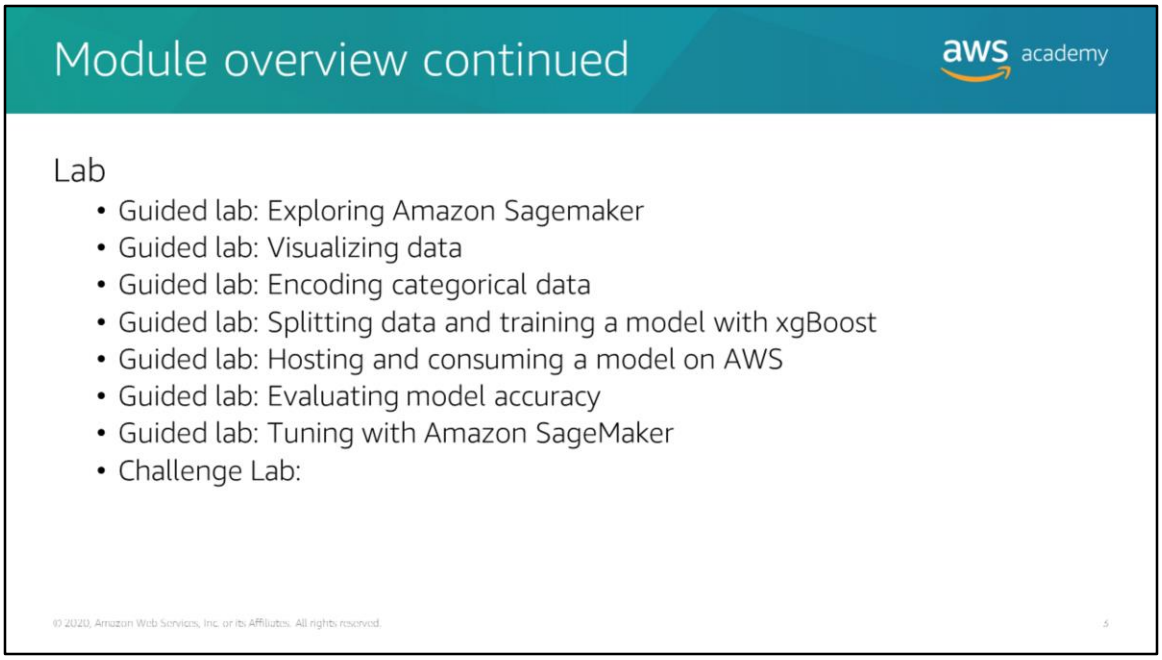
Welcome to Module 3: Implementing a Machine Learning Pipeline with Amazon SageMaker.



This module introduces and describes a typical process for handling a machine learning problem. A machine learning pipeline can be applied to many machine learning problems. This module focuses on supervised learning, but the process that you learn in this module can be adapted to other types of machine learning.

This module includes several guided hands-on labs, which enable you to practice what you learn. It also offers a challenge lab where you can explore on your own.

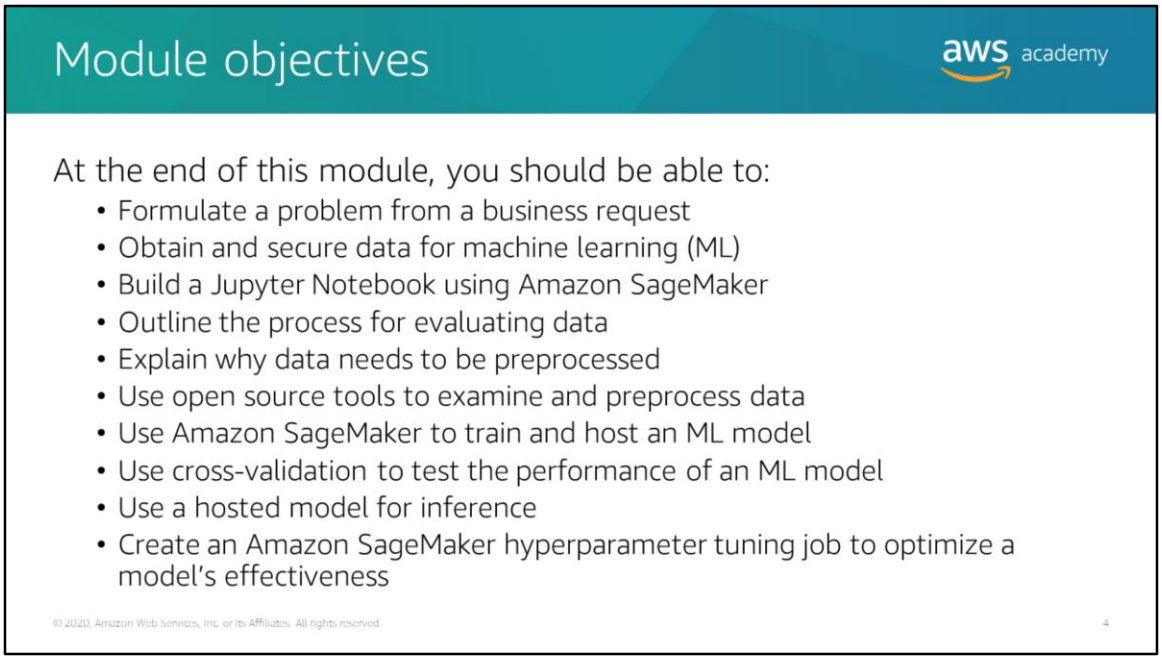
Finally, you will be asked to complete a knowledge check that will test your understanding of key concepts covered in this module.



This module introduces and describes a typical process for handling a machine learning problem. A machine learning pipeline can be applied to many machine learning problems. This module focuses on supervised learning, but the process that you learn in this module can be adapted to other types of machine learning.

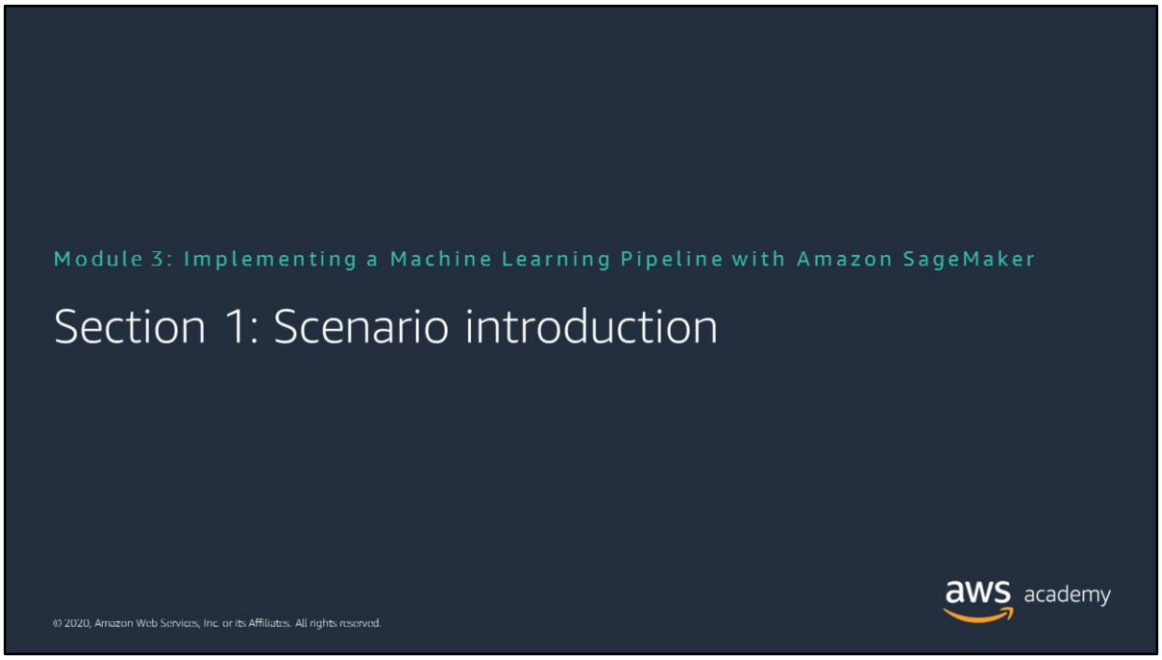
This module includes several guided hands-on labs, which enable you to practice what you learn. It also offers a challenge lab where you can explore on your own.

Finally, you will be asked to complete a knowledge check that will test your understanding of key concepts covered in this module.

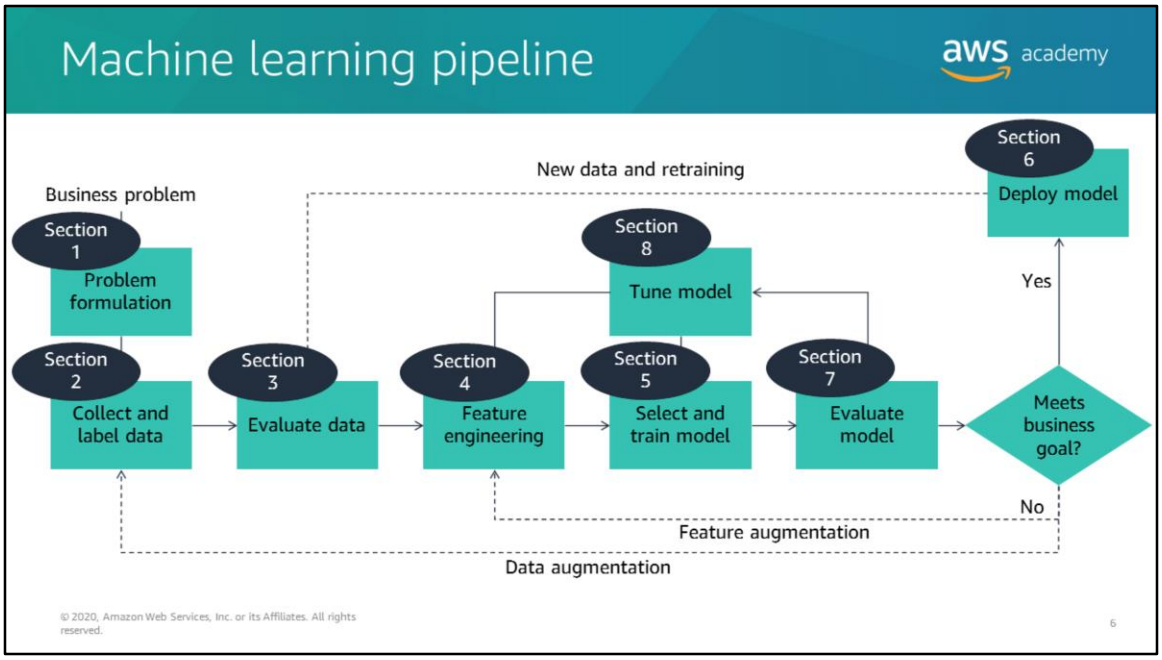


At the end of this module, you will be able to:

- Formulate a problem from a business request
- Obtain and secure data for machine learning (ML)
- Build a Jupyter Notebook by using Amazon SageMaker
- Outline the process for evaluating data
- Explain why data must be preprocessed
- Use open source tools to examine and preprocess data
- Use Amazon SageMaker to train and host an ML model
- Use cross-validation to test the performance of an ML model
- Use a hosted model for inference
- Create an Amazon SageMaker hyperparameter tuning job to optimize a model's effectiveness



Introducing Section 1: Scenario introduction. You will look at the different datasets that you will use in this module, along with guidance on how to formulate a business problem.



The machine learning pipeline is the focus of this module. The diagram shows which sections of this module cover each stage in the pipeline.

Section 1 covers problem formulation and the datasets that you will use throughout this module.

Section 2 explains how to obtain and secure data for your machine learning activities.

Section 3 shows you the tools and techniques for understanding your data.

Section 4 deals with preprocessing your data so that it is ready to train a model.

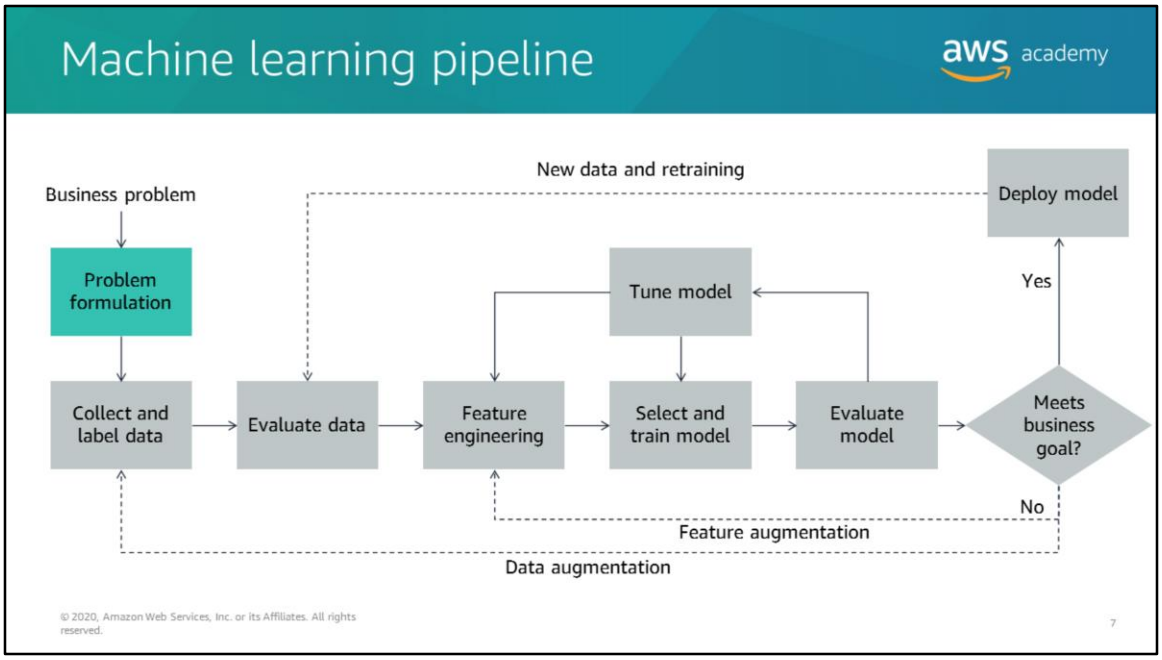
Section 5 is about selecting and training an appropriate machine learning model.

Section 6 shows you how to deploy a module so that you can predict possible outcomes.

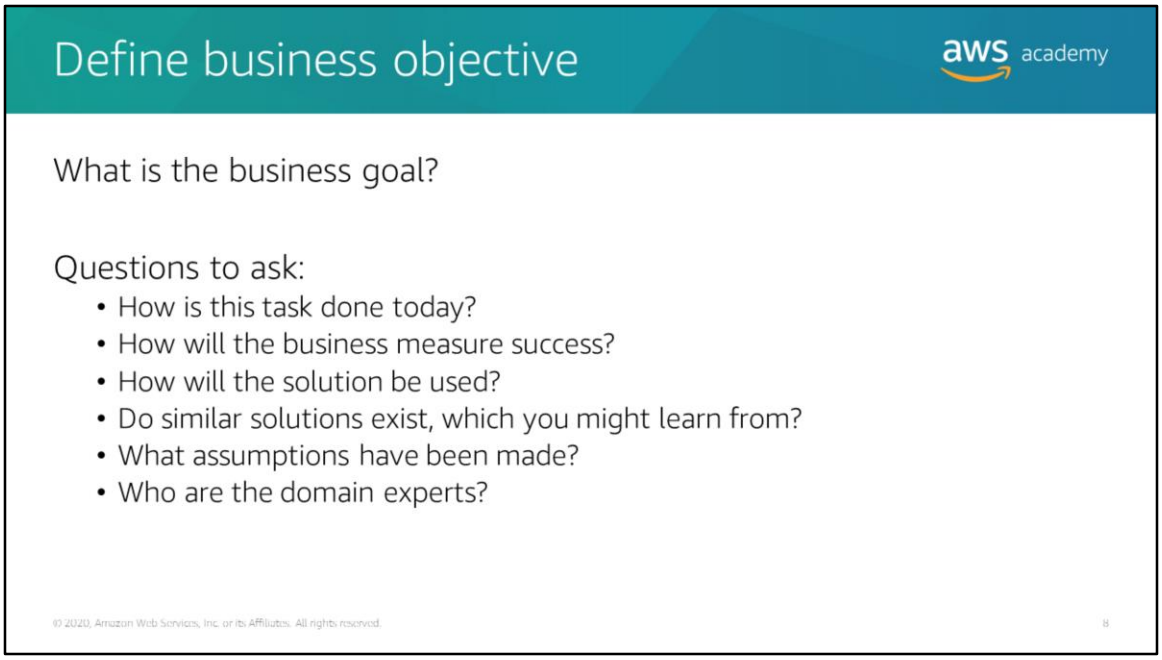
Section 7 examines the process of evaluating the performance of a machine learning model.

Section 8 walks you through tuning the model.

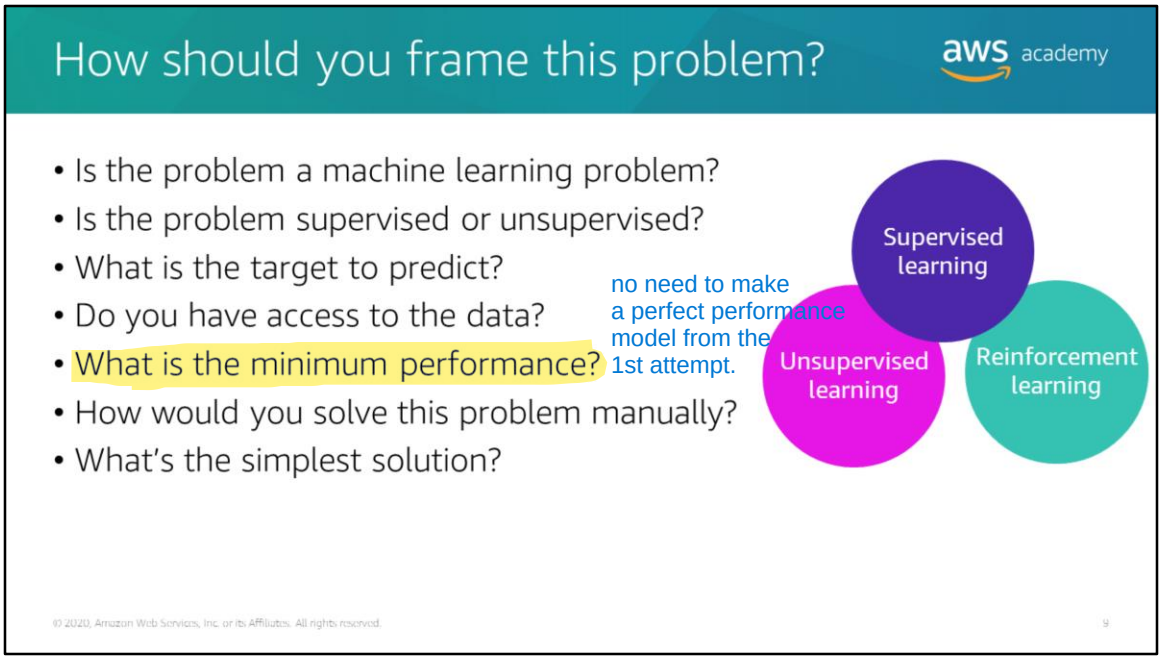
The machine learning pipeline is an iterative process. When you work on a real-world problem, you might iterate many times before you find a solution that meets the business needs.



In this first section, you examine how to think about turning a business requirement into a machine learning problem.



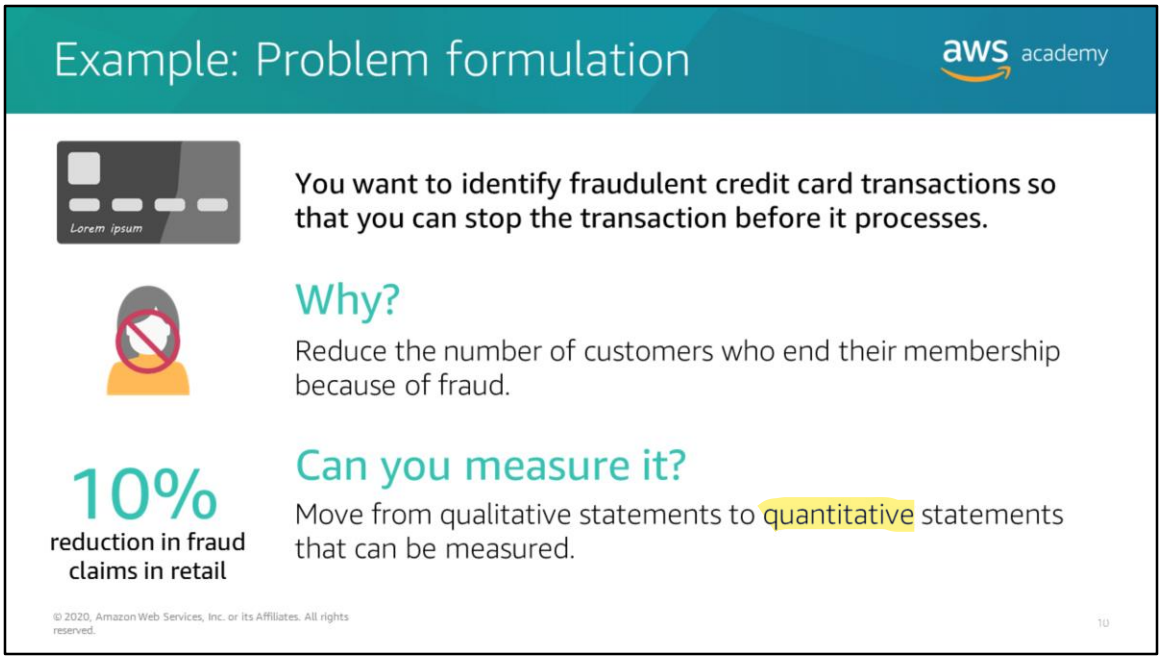
Your first step in this phase is to define the problem that you're trying to solve and the goal that you want to reach. Understanding the business goal is key, because you use that goal to measure the performance of your solution. You frequently must clarify the business problem before you can begin to target a solution. You must ask other questions so that you can thoroughly understand the problem.



With more information about the problem, you can begin to design an approach. First, can machine learning solve the problem, or would a traditional approach make more sense?

Is this problem a supervised or unsupervised machine learning problem? Do you have labeled data to train a supervised model?

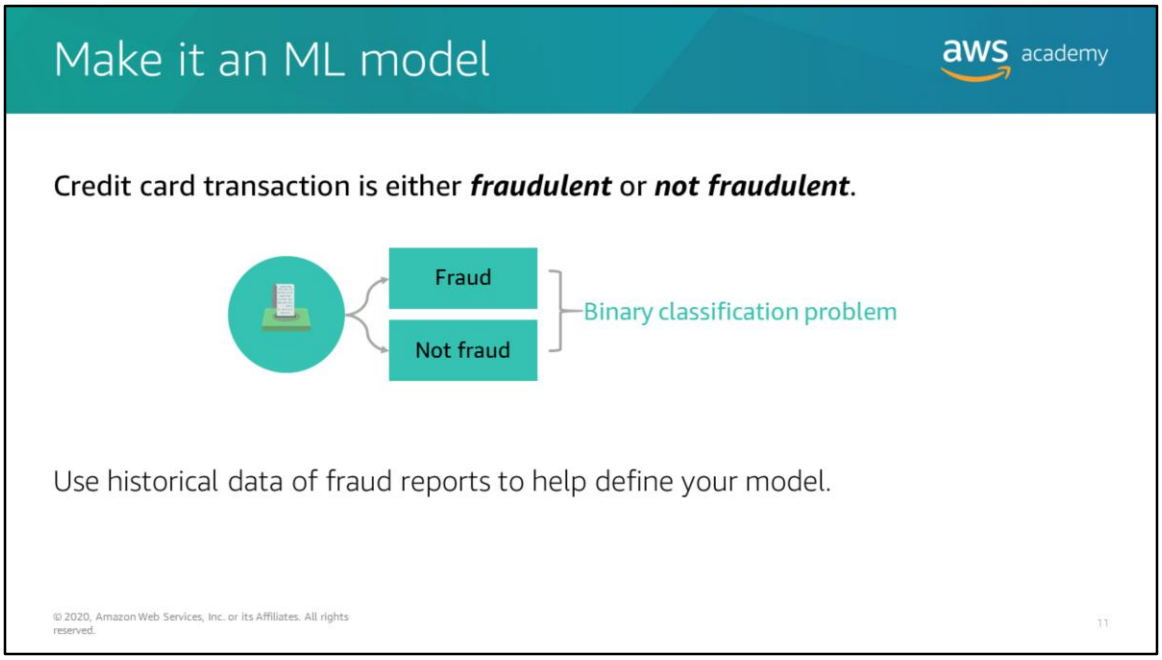
Again, you have many questions to ask yourself and the business. Ultimately, you should try to validate the use of machine learning and confirm that you have access to the right people and data. Then, devise the simplest solution to the problem.



Consider the following example. You want to identify fraudulent credit card transactions so that you can stop the transaction before it processes. That's the problem.

What is the business goal or outcome that drives this problem statement? The intended outcome is a reduction in the number of customers who end their membership to the credit card because of a fraudulent transaction.

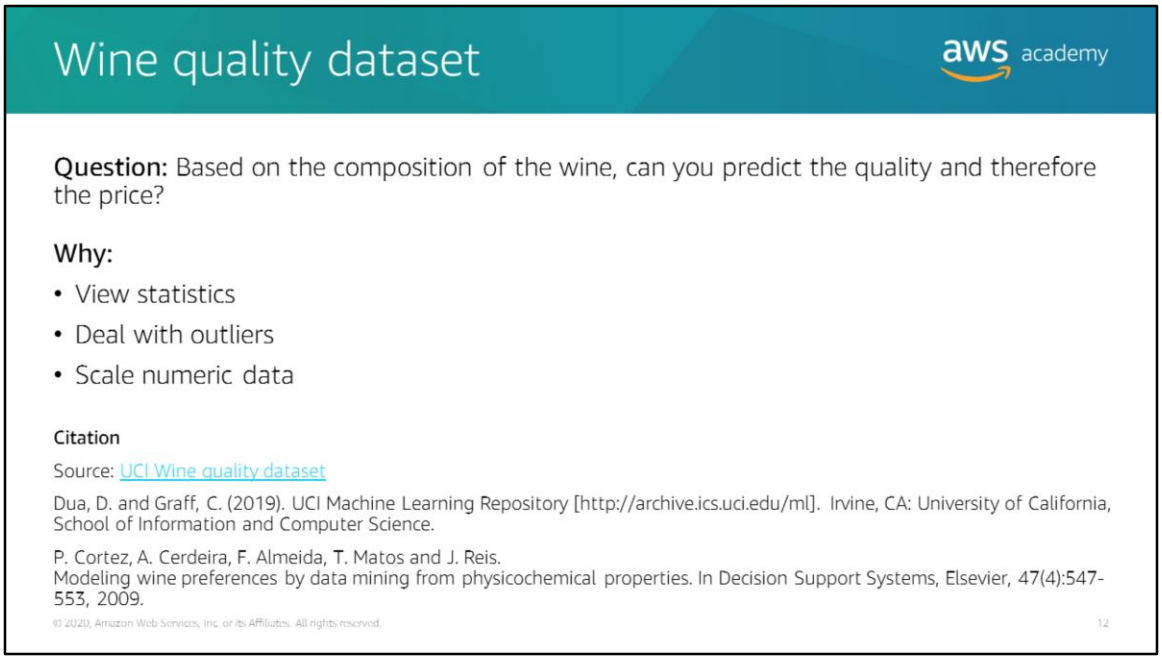
From a business perspective, how do you define success when you have this problem and intended outcome? At this stage, you must move from qualitative statements to quantitative statements that can be easily measured. Continuing with this example, you might define success for this problem with the following metric: a successful outcome is a 10 percent reduction in the number of customers who file claims for fraudulent transactions within a 6-month period.



You have now defined the business end of your problem. Next, you start to think about the problem in terms of your machine learning (ML) model. What output do you want to see from your model? Be specific—it should be a statement that reflects what an ML model can output. An example might be: The model will output whether a credit card transaction is *fraudulent* or *not fraudulent*.

Now that you know what your ML model should achieve, you can use this information to determine the type of ML that you will work with. If you have historical data where customers filed reports for fraud transactions, you can use this data for your machine learning purpose. This historical data fits into the supervised learning approach, where the labels are given.

Recall from earlier that you categorized supervised ML types into two buckets: classification and regression. In the credit card example, your intended output is to classify a transaction as either *fraud* or *not fraud*. Therefore, in this case, you can see that you're dealing with a *binary classification problem*.



Throughout this module, you will examine several datasets. These datasets, and more, can be obtained from the UC Irvine Machine Learning Repository at http://archive.ics.uci.edu/ml/index.php.

The first dataset contains numerical information about the composition of wine, along with the quality of the wine. You might use this dataset to ask the question: *Based on the composition of the wine, can you predict the quality and therefore the price?* You can also use this dataset to view statistics, deal with outliers, and scale numeric data.

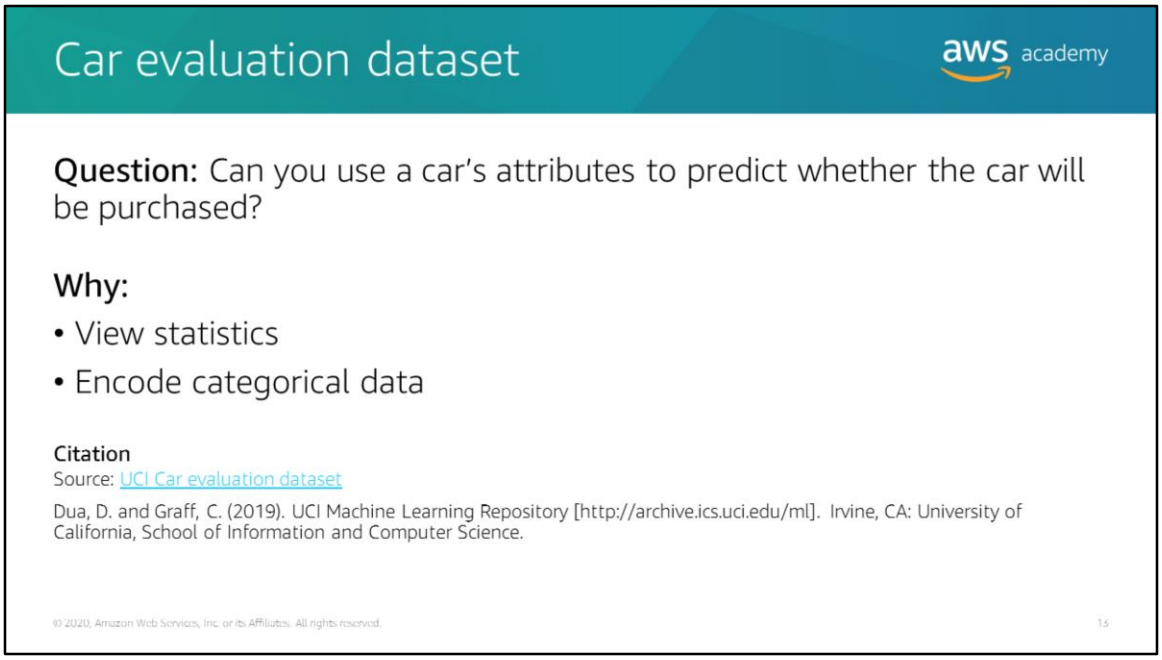
More information on this dataset can be found at: https://archive.ics.uci.edu/ml/datasets/wine+quality.

Dataset information
The two datasets are related to red and white variants of the Portuguese Vinho Verde wine. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available. No data about grape types, wine brand, or wine selling price is available.

Citation
Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml].

Irvine, CA: University of California, School of Information and Computer Science.

P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis.
Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

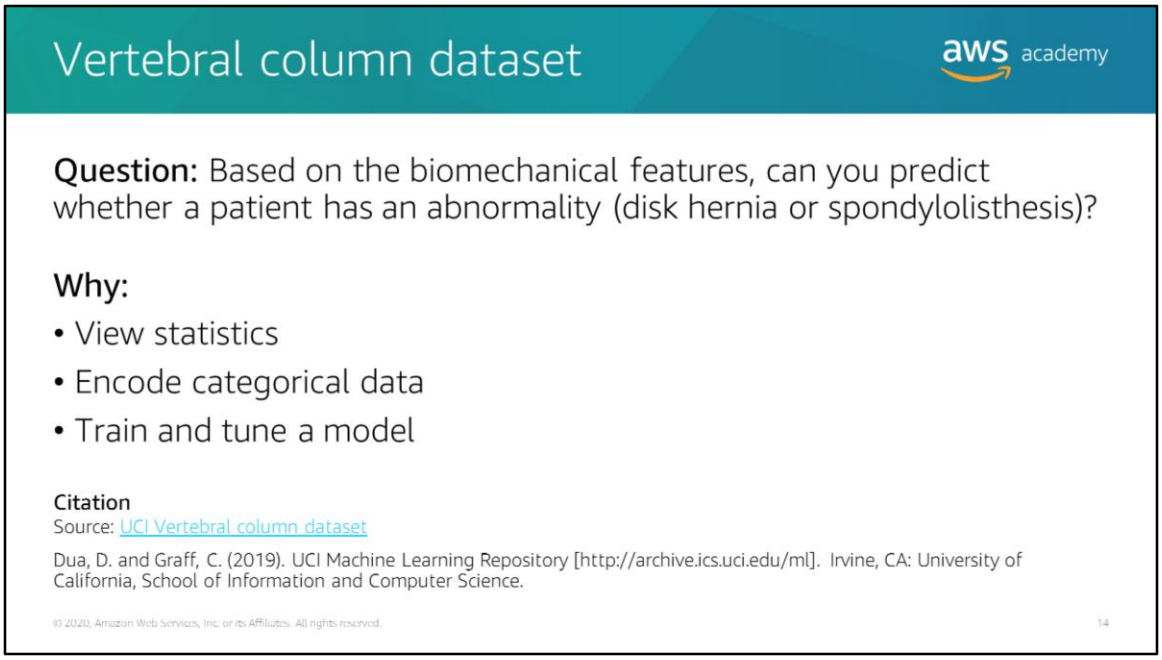


The second dataset is a car evaluation database. This dataset is mostly text-based. You can use it to explore encoding categorical data, which converts the text values into numbers that can be processed through machine learning.

More information on the dataset can be found at https://archive.ics.uci.edu/ml/datasets/Car+Evaluation.

Dataset information
Car Evaluation Database was derived from a simple hierarchical decision model. This model was originally developed for the demonstration of DEX, M. Bohanec, V. Rajkovic: Expert system for decision making. Sistemica 1(1), pp. 145-157, 1990.

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.



The third dataset is a biomedical dataset, which you will use in the lab. The question to answer is: *Based on the biomechanical features, can you predict whether a patient has an abnormality?*

You will take this dataset through the entire end-to-end process. You will end with a trained model that is tuned, and that you can use to predict an outcome.

More information on this dataset can be found at:
http://archive.ics.uci.edu/ml/datasets/vertebral%2Bcolumn

Dataset information
Dr. Henrique da Mota built this biomedical dataset during a medical residence period. He created it in the Group of Applied Research in Orthopaedics (GARO) of the Centre Médico-Chirurgical de Réadaptation des Massues, Lyon, France.

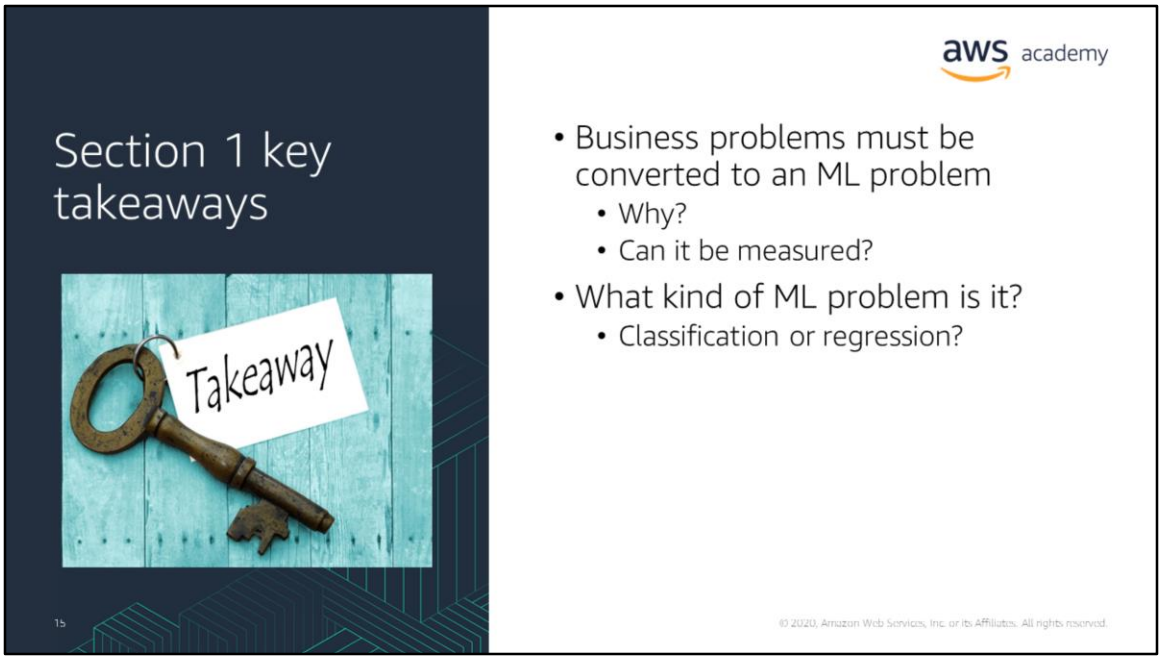
The data are organized in two different but related classification tasks. The first task consists of classifying patients as belonging to one of three categories: *Normal* (100 patients), *Disk Hernia* (60 patients), and *Spondylolisthesis* (150 patients).

For the second task, the categories *Disk Hernia* and *Spondylolisthesis* are merged into a single

category that is labeled as *abnormal*. Thus, the second task consists of classifying patients as belonging to one of two categories: *Normal* (100 patients) or *Abnormal* (210 patients).

Citation

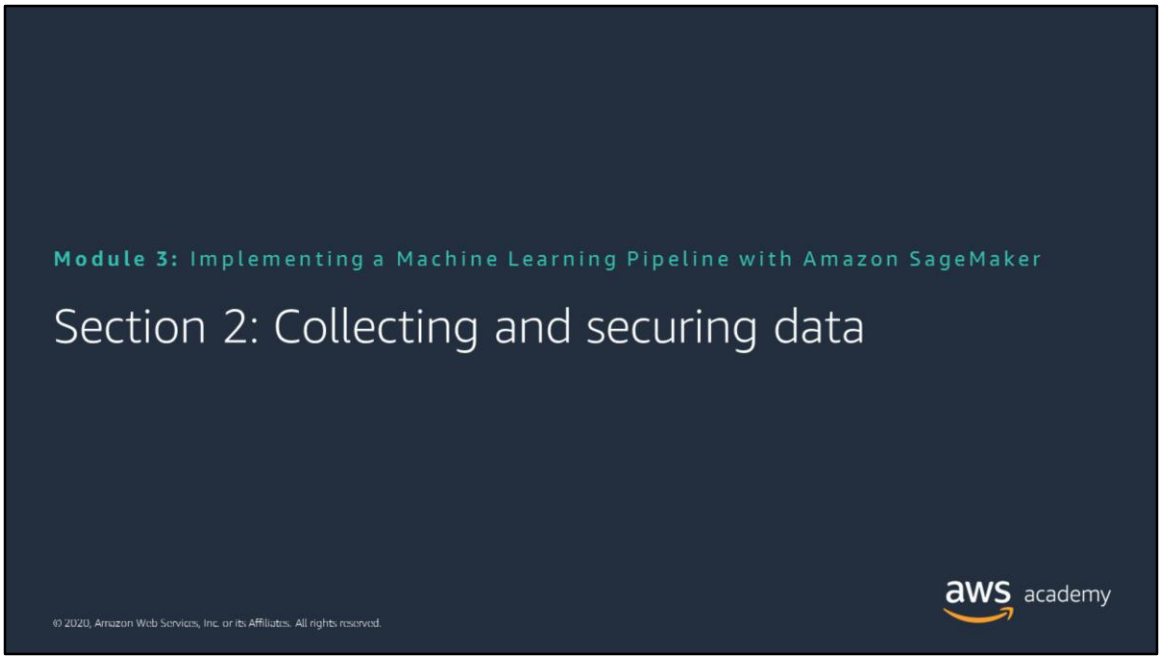
Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.



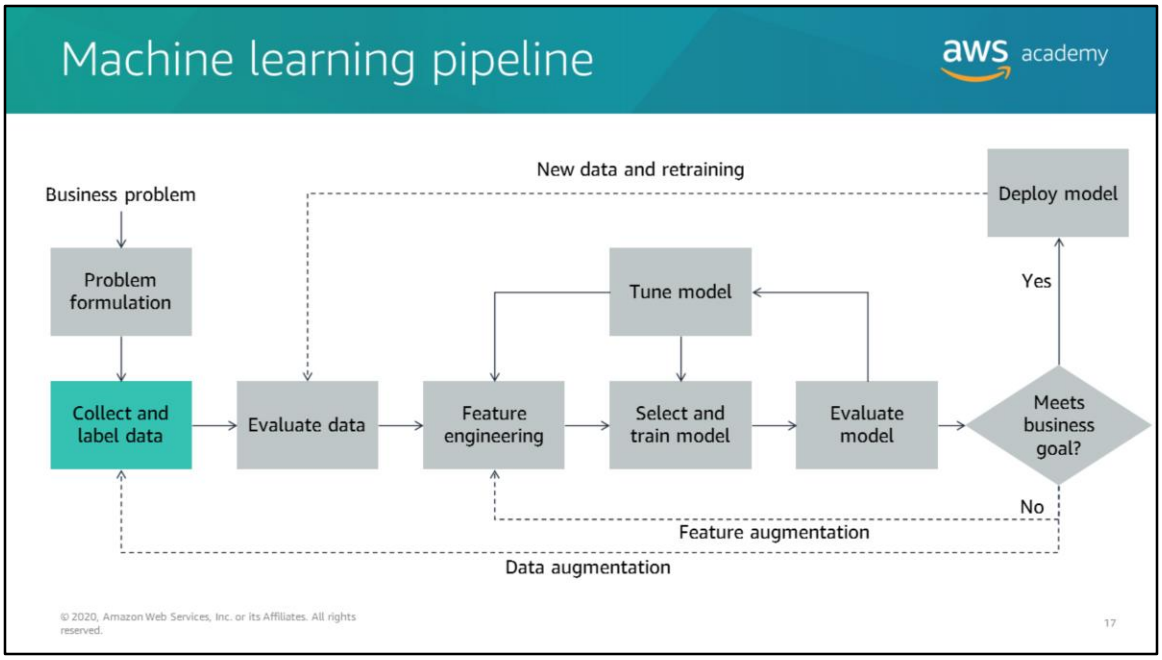
Some key takeaways from this section of the module are as follows.

- Business problems must be converted into an ML problem. Questions to ask include –
  - Have we asked why enough times to get a solid business problem statement and know why it is important?
  - Can you measure the outcome or impact if your solution is implemented?

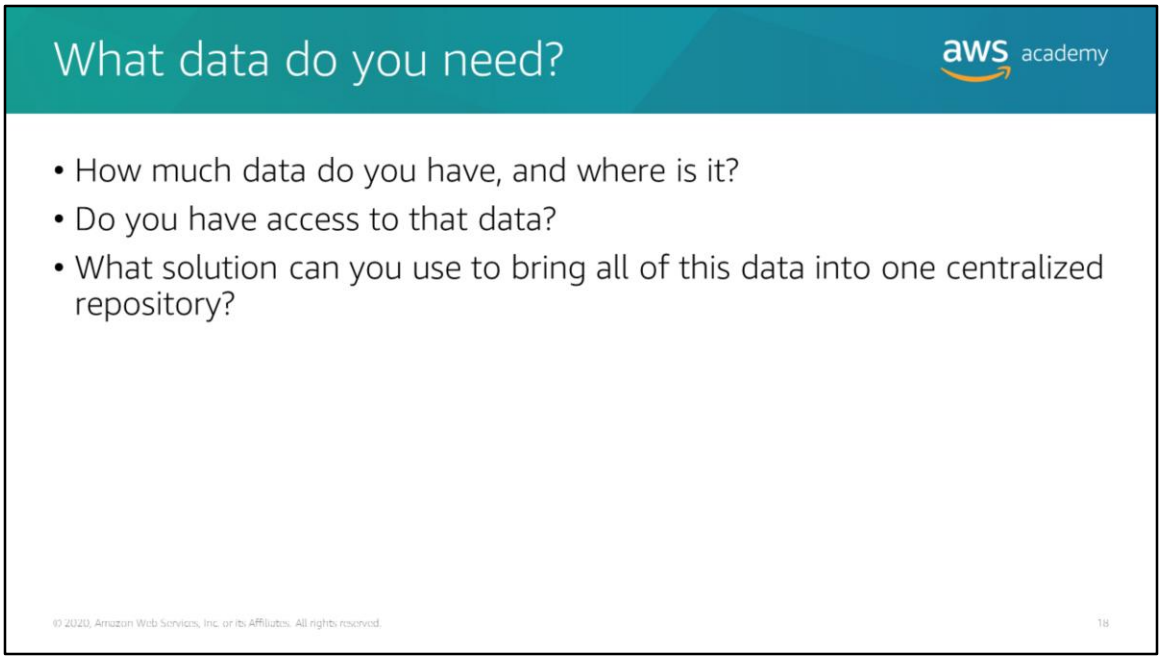
- Most business problems fall into one of two categories –
  - Classification (binary or multi): Does the target belong to a class?
  - Regression: Can you predict a numerical value?



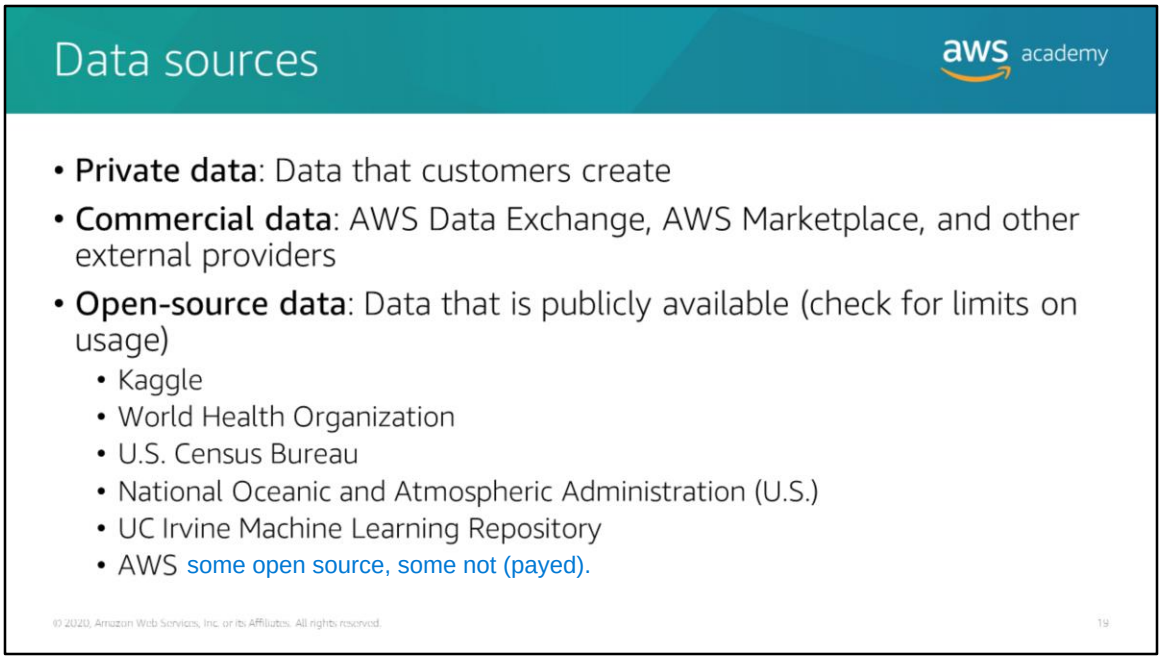
Introducing Section 2: Collecting and securing data.



In this section, you will explore some of the techniques and challenges involved with collecting and securing the data that you need machine learning.



Recall the original example about predicting credit card fraud. You formulated the problem. However, what data do you need to train your model to reach the intended output—and subsequently, your specified business outcome? Do you have access to that data? If so, how much data do you have, and where is it? What solution can you use to bring all this data into one centralized repository? Answering these questions is essential at this stage.

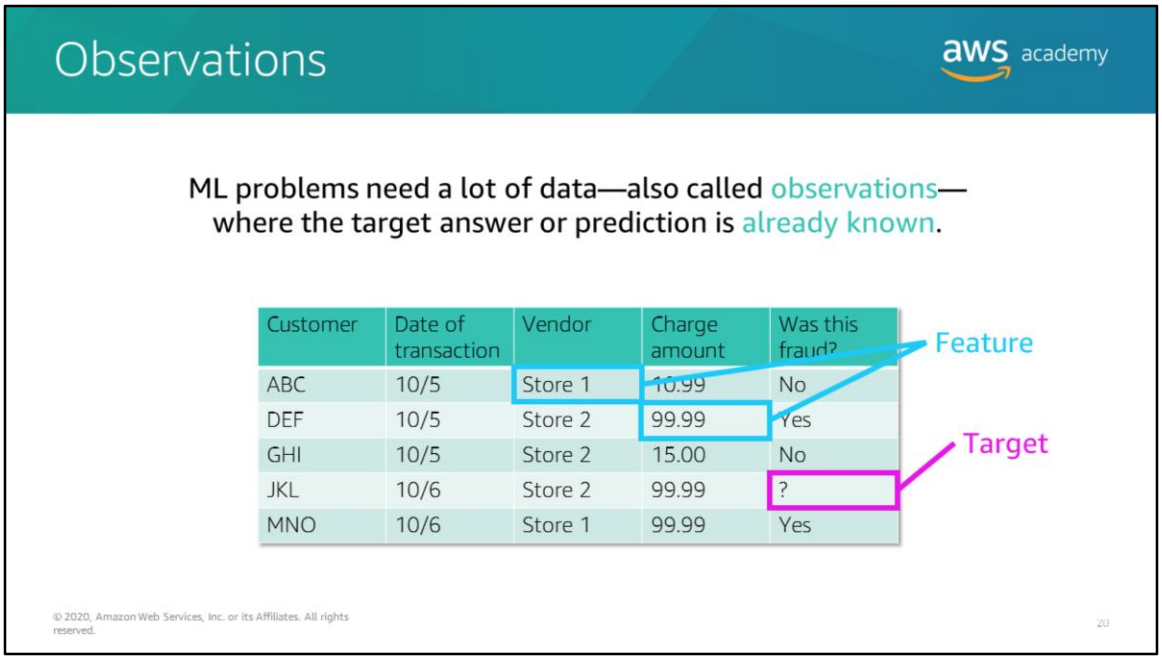


You can obtain data from several places.

- *Private data* is data that you (or your customers) have in various existing systems. Everything from log files to customer invoice databases can be useful, depending on the problem that you want to solve. In some cases, data is found in many different systems.

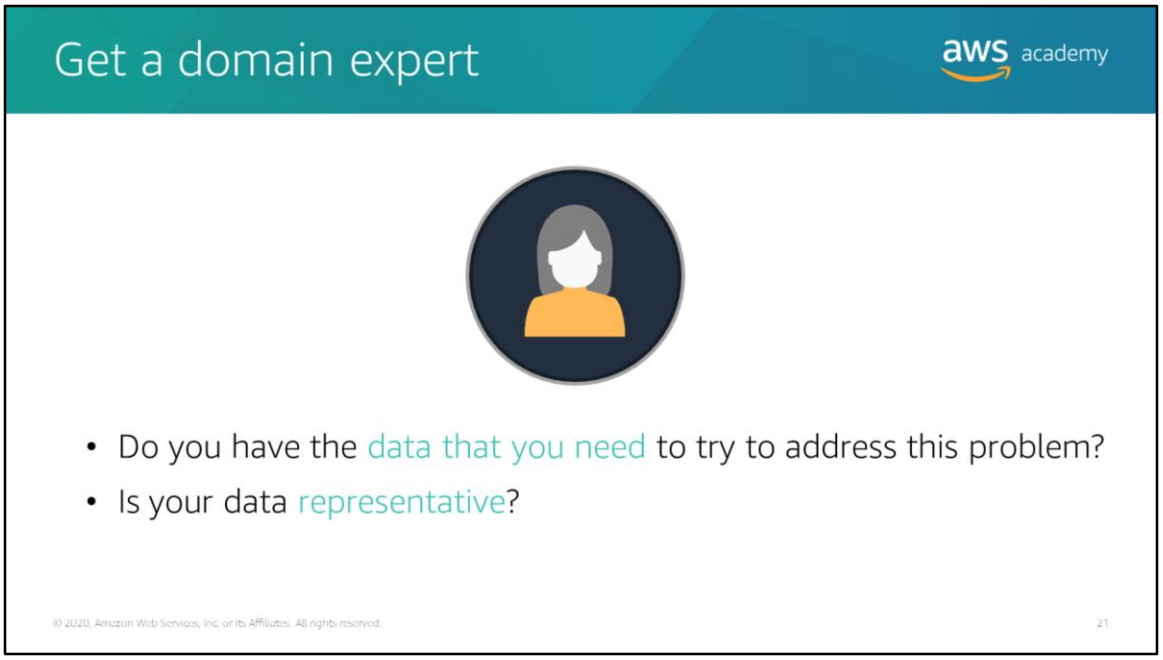
- *Commercial data* is data that a commercial entity collected and made available. Companies such as Reuters, Change Healthcare, Dun & Bradstreet, and Foursquare maintain databases that you can subscribe to. These databases include curated news stories, anonymized healthcare transactions, global business records, and location data. Supplementing your own data with commercial data can provide useful insights that you would not have otherwise.

- *Open-source data* comprises many different open-source datasets that range from scientific information to movie reviews. These datasets are usually available for use in research or for teaching purposes. You can find open-source datasets hosted by AWS, Kaggle, and the UC Irvine Machine Learning Repository. Government and health organizations are other sources of data that might be useful.



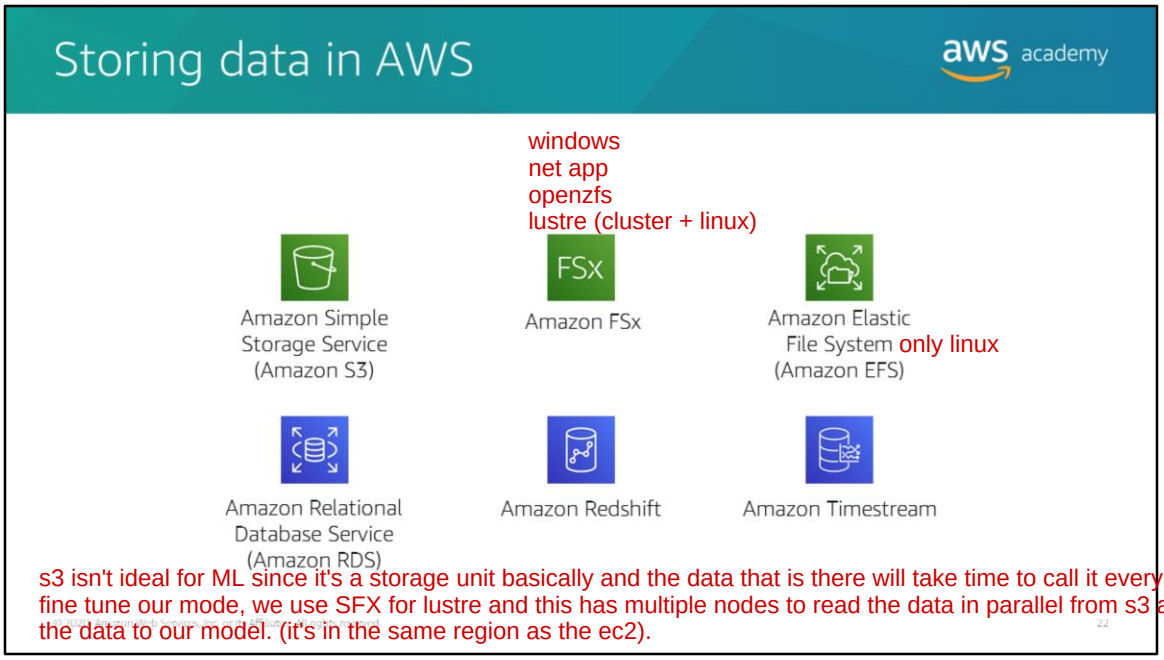
Supervised machine learning problems need much data—also called observations—for which you know the target answer or prediction. This kind of data is called *labeled data*. Each observation in your data is made up of two elements: the *target* and the *features*. The target is the answer that you want to predict. In the credit card transaction example, the target of any given observation is either *fraud* or *not fraud*. A feature is an attribute of the example that can be used to identify patterns to predict the target answer. A feature in the credit card example might be the *date of transaction*, the *vendor*, or the *amount, in dollars, of the transaction*.

You might wonder about the source of the target, *fraud* or *not fraud*. Typically, this information is discovered after the transaction is complete and the actual card owner notices a fraudulent transaction on their statement. This information is then recorded with the transaction so it can be used to train a future model.



You now know about the elements of an ML dataset. Recall a previous question: *What data do you need to train your model to reach the intended output, and subsequently, your intended business outcome?* This example depicts a stage in the ML pipeline when it's vital to get domain expertise to help you answer this question. With domain knowledge, you can begin to determine the features and target data that your model needs to make accurate predictions.

Your data should be representative of the data that you are going to have when you use the model to make a prediction. For example, if you want to predict credit card fraud, you must collect both positive data (fraudulent transactions) and negative data (non-fraudulent transactions). The ML algorithm must have both types of data so that it can find patterns that distinguish between the two types. For example, suppose that the average amount of fraudulent transactions is actually 3 percent. However, your training dataset includes only a small fraction of fraudulent observations, say 0.4 percent. In this scenario, it might be difficult for your model to correctly learn patterns for fraudulent transactions that it might encounter in production.



AWS has many different services where you can find or store your data. You might use some of the following key services.

Amazon Simple Storage Service (Amazon S3) is *object-level storage*. Thus, if you want to change part of a stored object or file, you must make the change and then re-upload the entire modified file. Amazon S3 enables you to store as much data as you want, in the aggregate that you want. However, individual objects cannot be larger than 5 TB.

By default, data in Amazon S3 is stored redundantly across multiple facilities and multiple devices in each facility. You can access Amazon S3 through the web-based AWS Management Console or programmatically through application programming interfaces (APIs) and software development kits (SDKs). You can also use third-party solutions (which use APIs and SDKs). Amazon S3 includes *event notifications* that enable you to set up automatic notifications when certain events occur. An example of such an event might be an object that is uploaded to or deleted from a specific bucket. Those notifications can be sent to you, or they can be used to trigger other processes, such as AWS Lambda scripts.

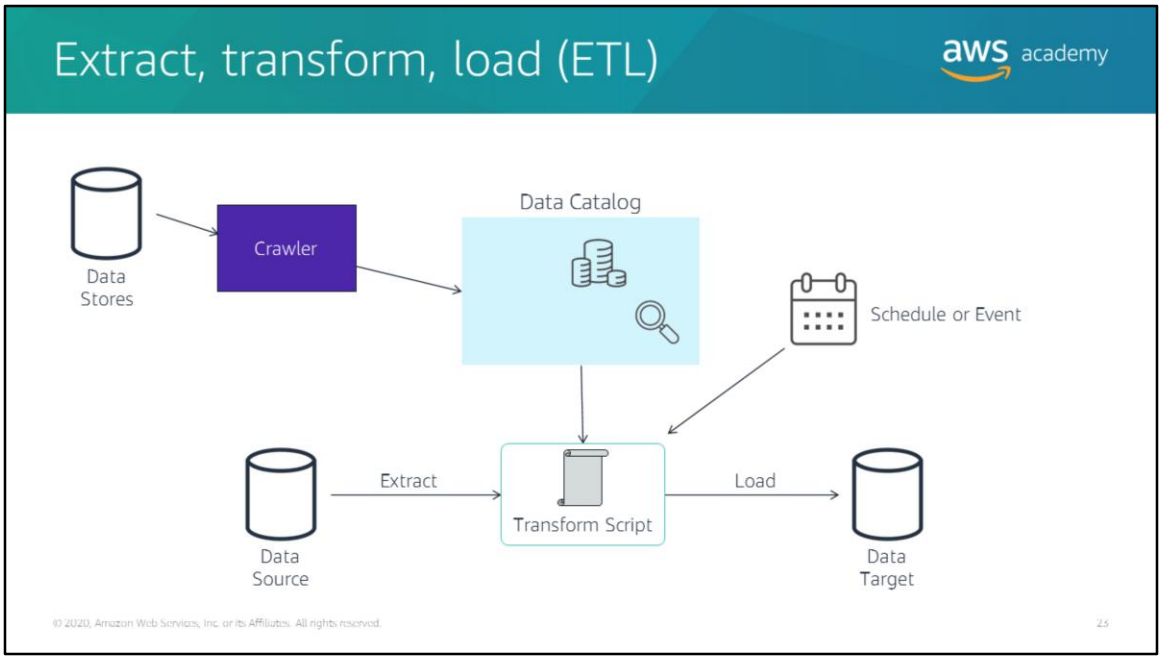
When your training data is in Amazon S3, you might plan to run training jobs several times with different algorithms and parameters. For these runs, consider trying Amazon FSx for Lustre, a file system service. FSx for Lustre speeds up your training jobs by serving your

Amazon S3 data to Amazon SageMaker at high speeds. The first time that you run a training job, FSx for Lustre automatically copies data from Amazon S3 and makes it available to Amazon SageMaker. You can use the same Amazon FSx file system for subsequent iterations of training jobs, which prevents repeated downloads of common Amazon S3 objects.

Alternatively, if your training data is in Amazon Elastic File System (Amazon EFS), you should use it as your training data source. Amazon EFS has the benefit of directly launching your training jobs from the service without the need for data movement. The result is faster training start times. When data scientists have home directories in Amazon EFS, they can quickly iterate on their models. They can bring in new data, share data with colleagues, and experiment with including different fields or labels in their dataset. For example, data scientists can use a Jupyter notebook to do initial cleansing on a training set and launch a training job from Amazon SageMaker. They then can use their notebook to drop a column and re-launch the training job, and they can compare the resulting models to see which one works better.

You might find data in many other places in AWS. These sources include Amazon Relational Database Service (Amazon RDS), Amazon Redshift (which is a managed data warehouse service), and Amazon Timestream. Amazon Timestream is a managed time-series database that is designed specifically to handle large amounts of Internet of Things (IoT) data. You can even spin up your own EC2 instances and host your own database.

You must be able to extract useful data from these sources when you assemble your data for machine learning, which is the next topic.



Data is typically spread across many different systems and data providers. The challenge is to bring all these data sources together into something that a machine learning model can consume.

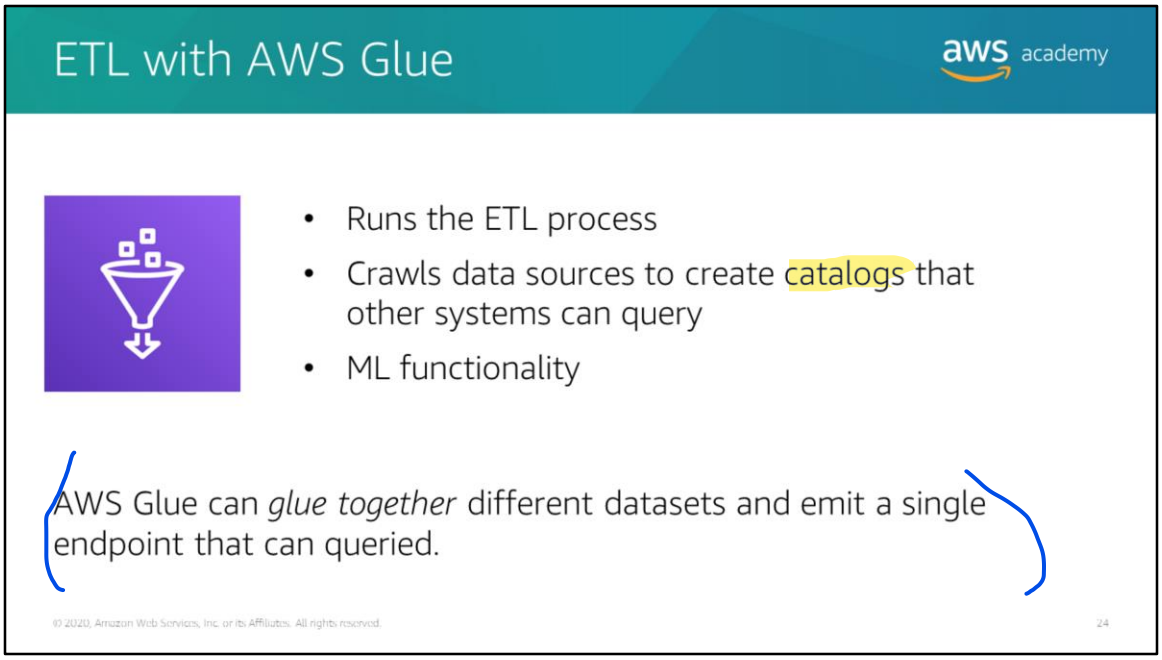
The steps in an extract, transform, and load (ETL) process are defined as follows.

- **Extract –** Pull the data from the sources to a single location.
- **Transform –** During extraction, the data might need to be modified, matching records might need to be combined, or other transformations might be necessary.
- **Load –** Finally, the data is loaded into a repository, such as Amazon S3 or Amazon Athena.

A typical ETL framework has several components. From the diagram, these components include:

- **Crawler –** A program that connects to a data store (source or target). It progresses through a prioritized list of classifiers to determine the schema for your data, and creates metadata tables in the AWS Glue Data Catalog.
- **Job –** The business logic that is required to perform ETL work.
- **Schedule or event –** A scheduling service that periodically runs the ETL process.

Note: The services that are shown exist in the *transform* partition of the ETL process.



AWS Glue is a fully managed ETL service. With AWS Glue, it is simple and cost-effective to categorize your data, clean it, enrich it, and move it reliably between various data stores.

AWS Glue consists of the following components:

- A central metadata repository, which is known as the AWS Glue Data Catalog
- An ETL engine that automatically generates Python or Scala code
- A flexible scheduler that handles dependency resolution, job monitoring, and retries

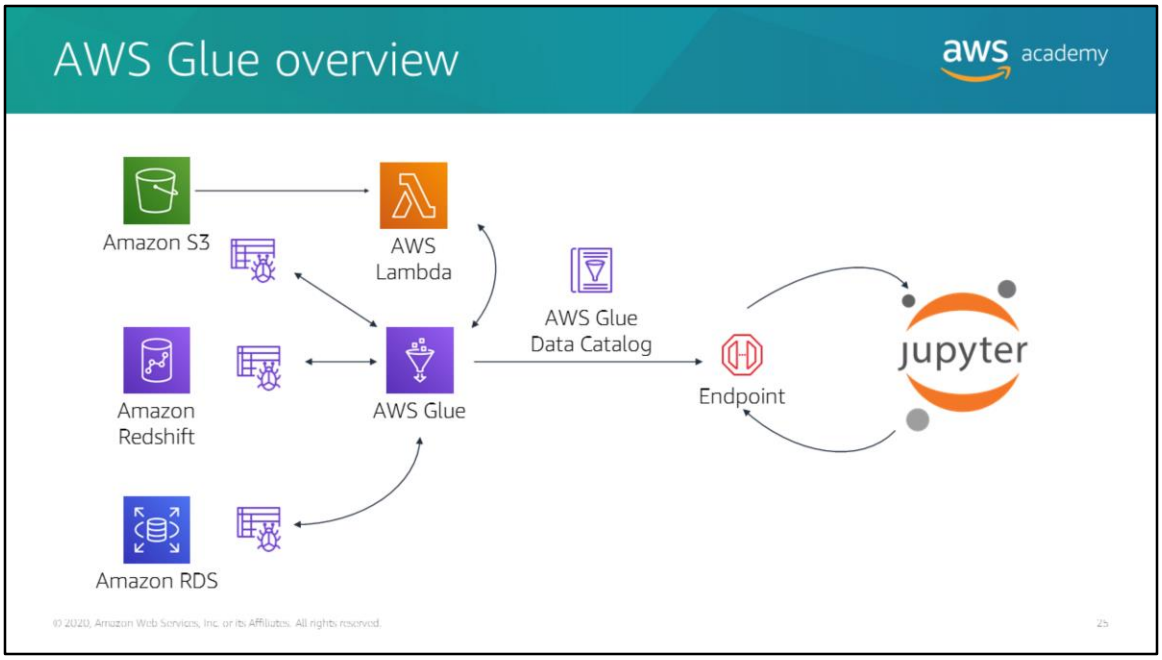
AWS Glue is serverless, so you don't have infrastructure to set up or manage.

AWS Glue is designed to work with semistructured data. It introduces a component that is called a *dynamic frame*, which you can use in your ETL scripts. A dynamic frame is similar to an Apache Spark DataFrame, except that each record is self-describing, so no schema is required initially. With dynamic frames, you get schema flexibility and a set of advanced transformations that are specifically designed for dynamic frames. You can convert between dynamic frames and Spark dataframes. In this way, you can take advantage of both AWS Glue and Spark transformations to do the kinds of analysis that you want.

You can use the AWS Glue console to discover data, transform it, and make it available for

search and querying. The console calls the underlying services to orchestrate the work that is required to transform your data. You can also use the AWS Glue API operations to interface with AWS Glue services. You can edit, debug, and test your Python or Scala Apache Spark ETL code in a familiar development environment.

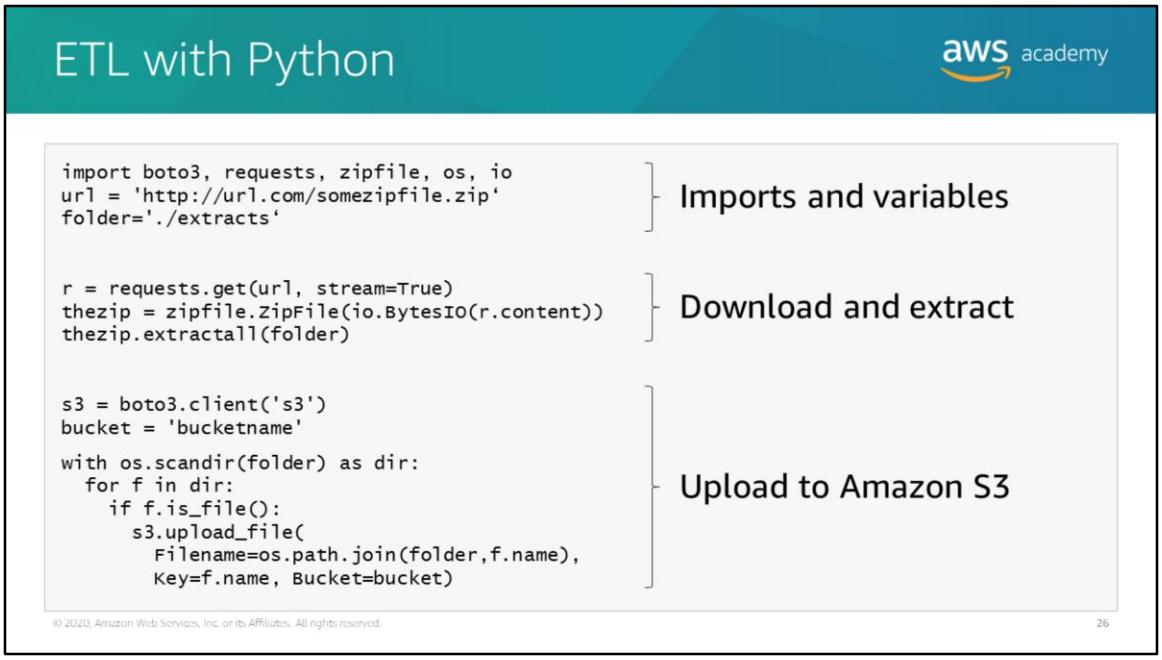
The ML capability references the AWS Glue capability to receive labeled data that can be used for training. One example might be to provide training data that teaches the model what duplicate records would look like in the data source. Then, AWS Glue can identify the duplicates and present them to a data engineer for further analysis.



AWS Glue enables complex ETL jobs to be orchestrated. In the example that is shown, AWS Glue crawls the data sources and presents the information as a data catalog to clients, such as Jupyter.

Event-driven ETL pipelines

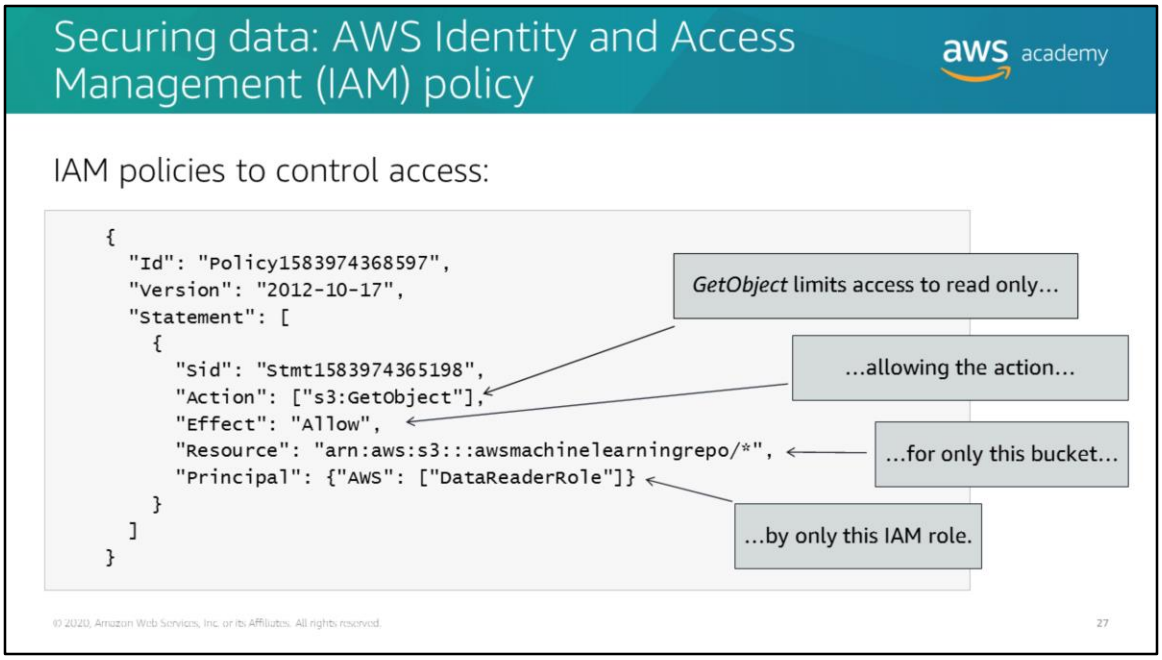
AWS Glue can run ETL jobs that are based on an event, such as getting a new dataset. For example, you can use a Lambda function to trigger your ETL jobs to run when new data becomes available in Amazon S3. You can also register this new dataset in the AWS Glue Data Catalog as part of your ETL jobs.



Although managed tools are available in AWS to manipulate data, data scientists also write scripts in their Jupyter notebook to handle data. A simple extract and load script is shown, which includes:

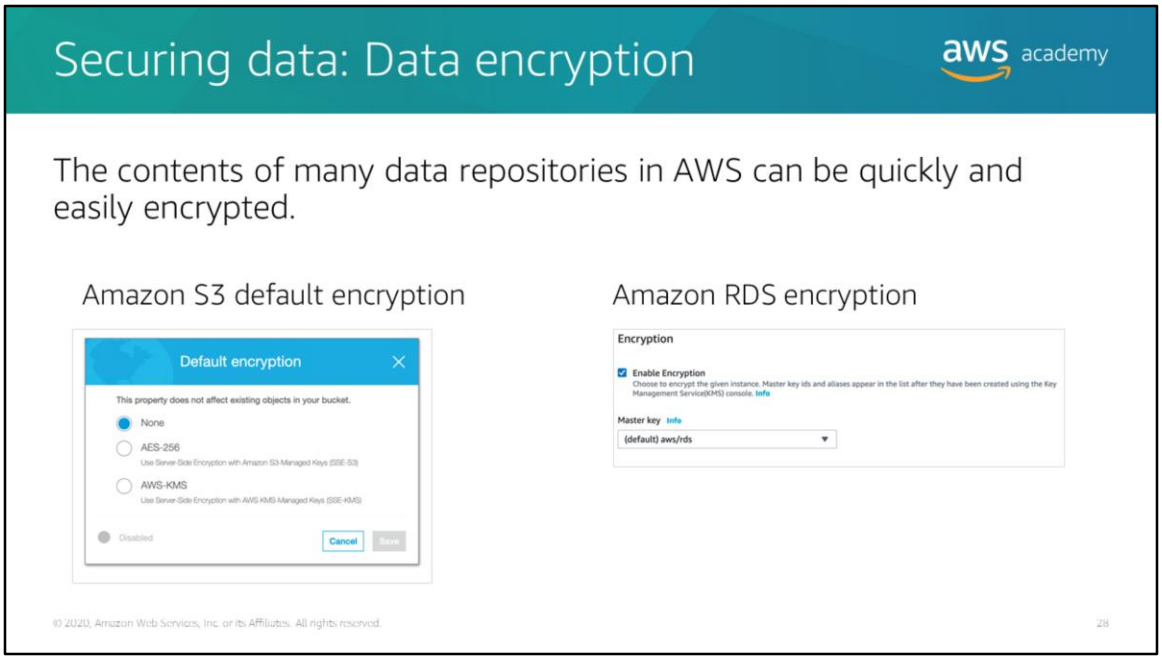
- Imports and variables – This section imports the libraries that are used. Notice that *boto3* is the library for AWS. Variables are also set here for the *zipfiles* web location and a local folder for extraction.
- Download and extract – This section makes a web request and saves the bytes from the URL as a stream. This stream is then passed to the ZipFile function, which is then used to extract the data.
- Upload to Amazon S3 – With the extracted files in a folder, this section enumerates the folder's files. It also uploads each file to Amazon S3.

If you find that this script is used often, it should be migrated to a standalone function that Python applications can import.



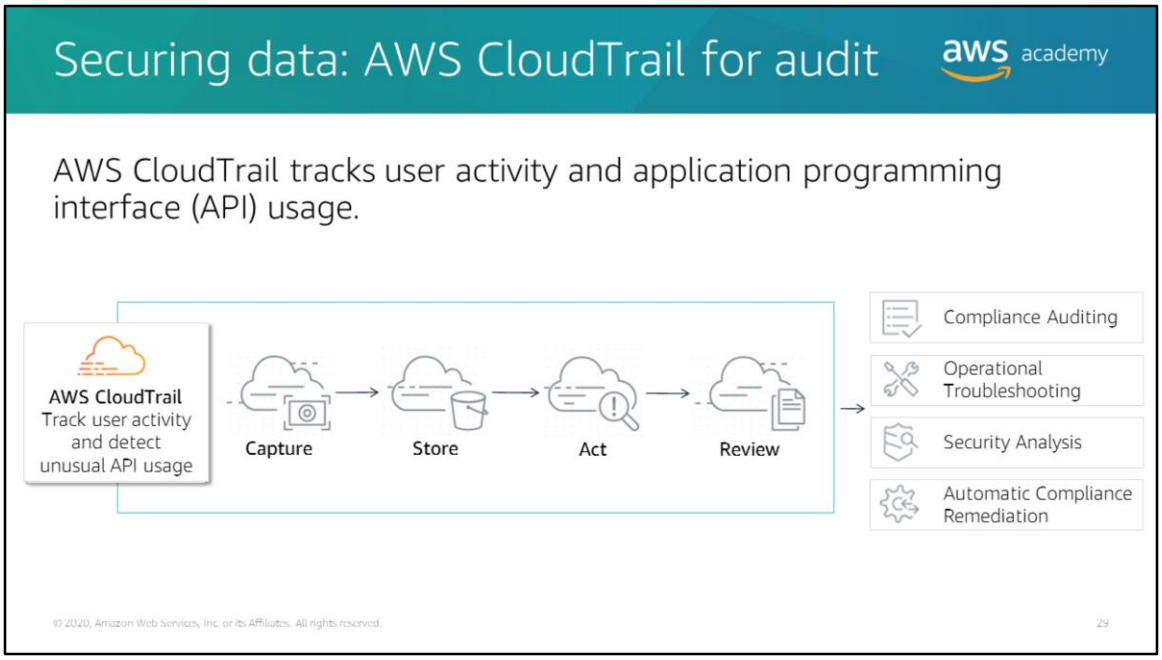
It's important to consider the security of your data. The datasets that you use in this course are all public, but real data about customer transactions or health records must be kept secure. The AWS Identity and Access Management (IAM) service controls access to resources. Make sure that you correctly secure your data within AWS to avoid data breaches.

The diagram shows a simple IAM policy that allows only read access to a specific S3 bucket for the specified role.



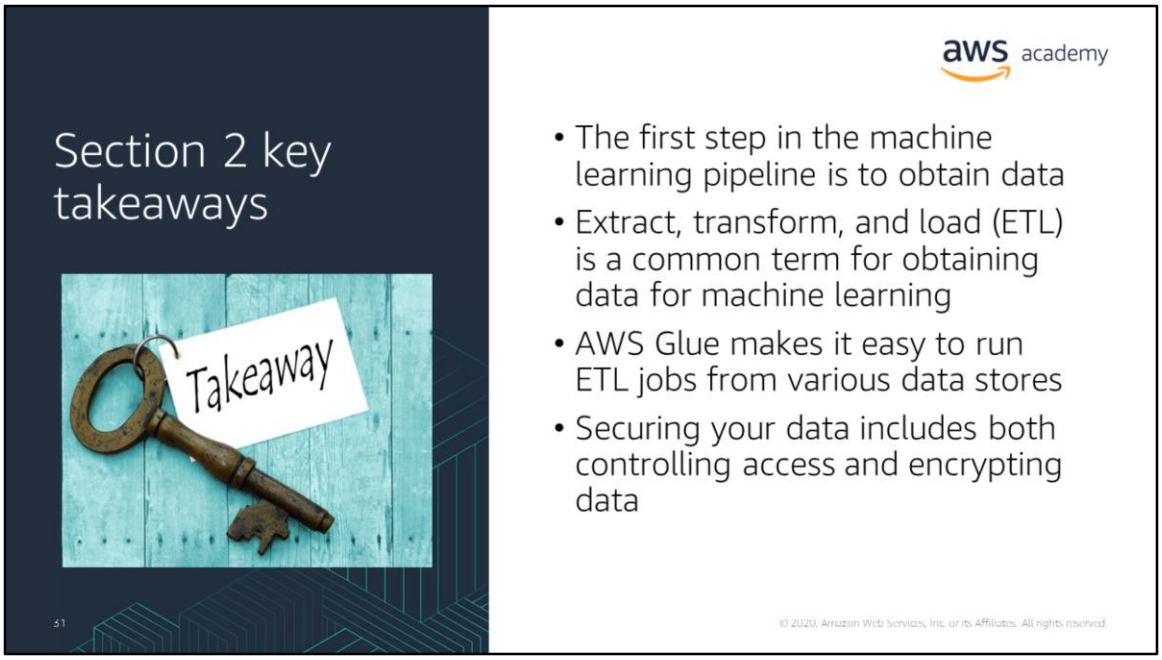
In addition to controlling access to data, you must make sure that your data is secure. It's a good practice, and it might also be legally required for certain data types, such as financial or healthcare records.

AWS provides encryption features for storage services, typically both at rest and in transit. You can often meet these encryption requirements by enabling encryption on the object or service that you need to protect. For in-transit data, you must use secure transports, such as Secure Sockets Layer/Transport Layer Security (SSL/TLS).



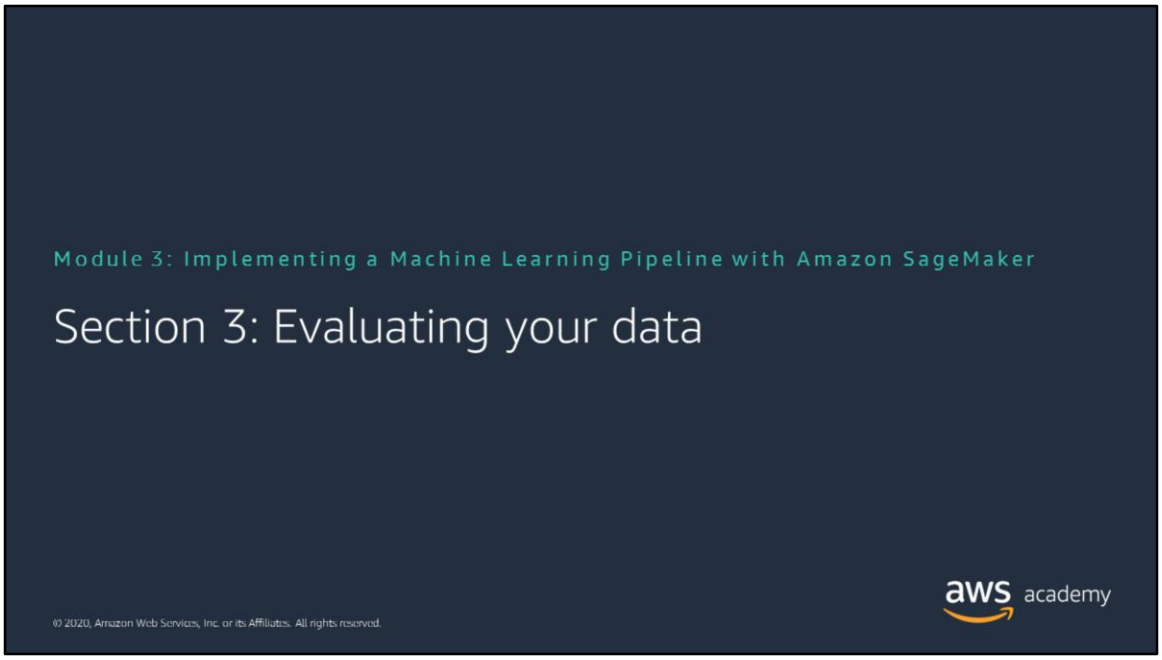
Another aspect to consider is compliance audits. When you work with data from regulated industries, you often must audit access to data. AWS CloudTrail is a service that enables governance, compliance, operational auditing, and risk auditing of your AWS account. With CloudTrail, you can log, continuously monitor, and retain account activity that is related to actions across your AWS infrastructure. CloudTrail provides an event history of your AWS account activity. This activity includes actions that are taken through the AWS Management Console, AWS SDKs, command line tools, and other AWS services. This event history simplifies security analysis, resource change tracking, and troubleshooting. In addition, you can use CloudTrail to detect unusual activity in your AWS accounts. These capabilities help simplify operational analysis and troubleshooting.

You now complete Module 3 – Guided Lab 1: Exploring Amazon SageMaker.

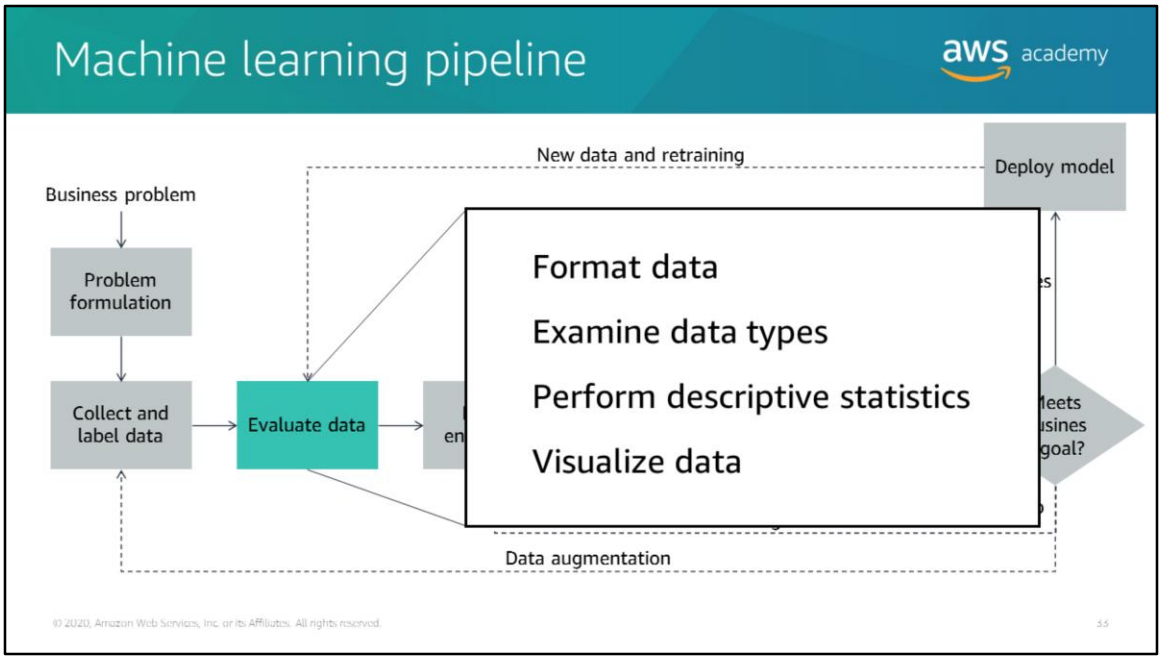


Some key takeaways from this section of the module include:

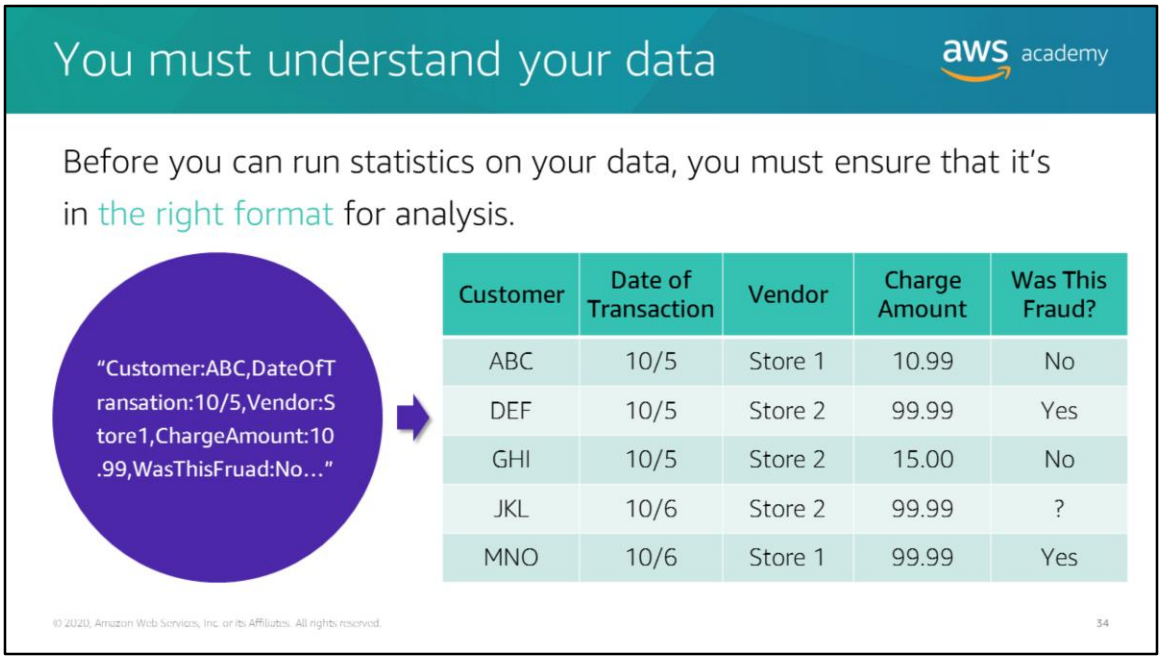
- The first step in solving machine learning problems is to obtain data that is required to train your machine learning model.
- ETL can be used to obtain data from multiple sources.
- Services such as AWS Glue can make it easy to obtain data from multiple data stores.
- Security requirements should be understood, and they should be based on both business need and any regulatory requirements. You want to ensure that your data is secure, only authorized users can access it, and it is encrypted where possible.



Introducing Section 3: Evaluating your data.



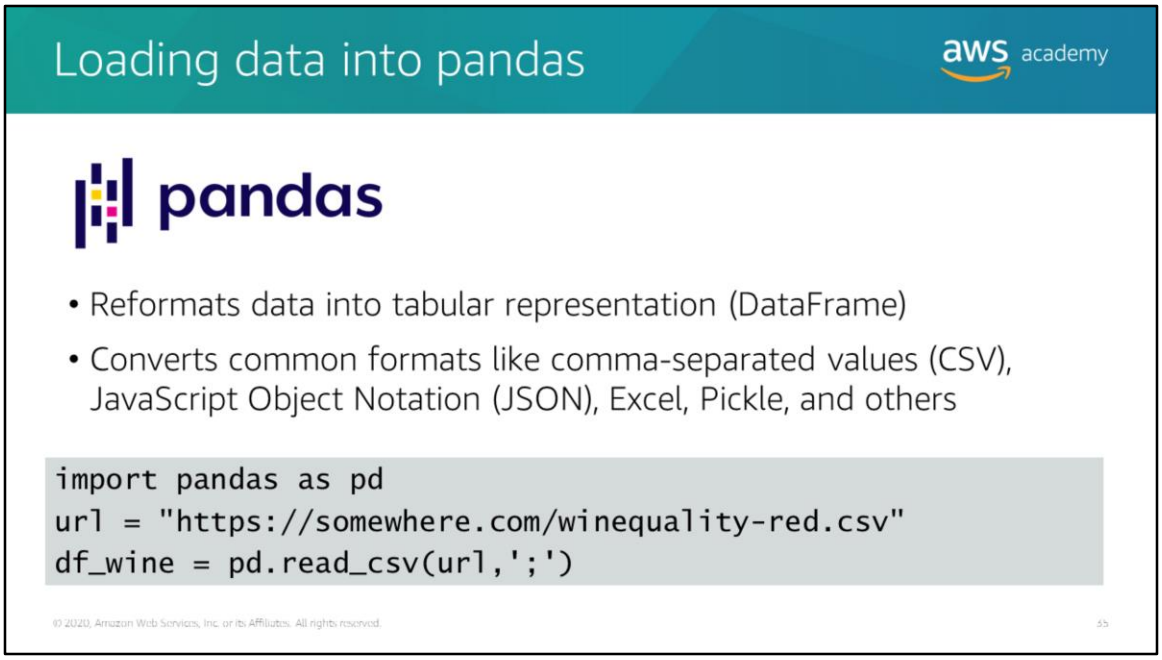
In this section, you learn how to evaluate data for machine learning. You look at different data formats and types, and how you can visualize and analyze the data before feature engineering.



Before you can run statistics on your data to better understand it, you must ensure that it's in the right format for analysis. For Amazon SageMaker, algorithms support training with data in comma-separated values (CSV) format. Many of the tools that you use to explore, visualize, and analyze the data can also read the data in CSV format.

In general, you must have at least some domain knowledge for the problem that you are trying to solve with machine learning. To develop a model that predicts whether a set of symptoms indicates a disease, you must know the relationship between the symptoms and the disease.

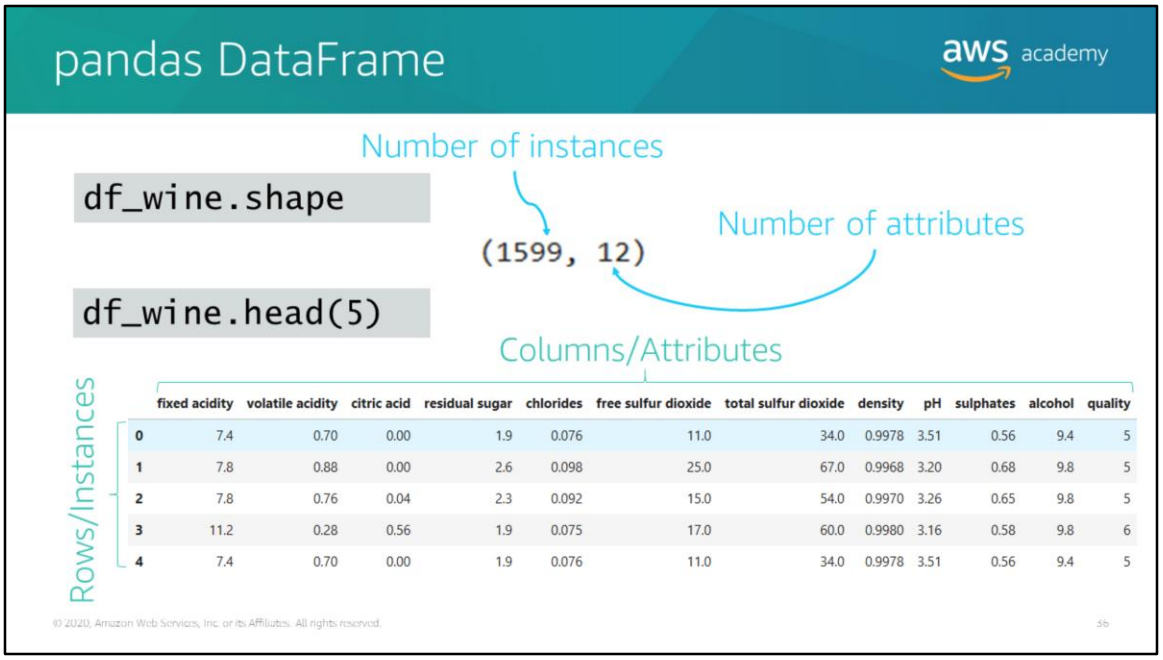
Data generally must be put into numeric form so ML algorithms can use the data to make predictions. In the next section, you will learn about ways that you can convert text data. For now, you will learn how to explore the data and how to gain some insights into the overall set of data.



One of the more popular open-source Python libraries is *pandas*. It can reformat data from various formats like CSV, JavaScript Object Notation (JSON), Excel, Pickle, and others into a tabular representation of your data in rows and columns. In addition, pandas has data analysis and manipulation features, which you will use throughout this module.

By using pandas, you can load data in many different formats such as CSV, JSON, Excel, and Pickle.

Loading data can be as simple as the example, which retrieves the CSV file from the URL that is specified.

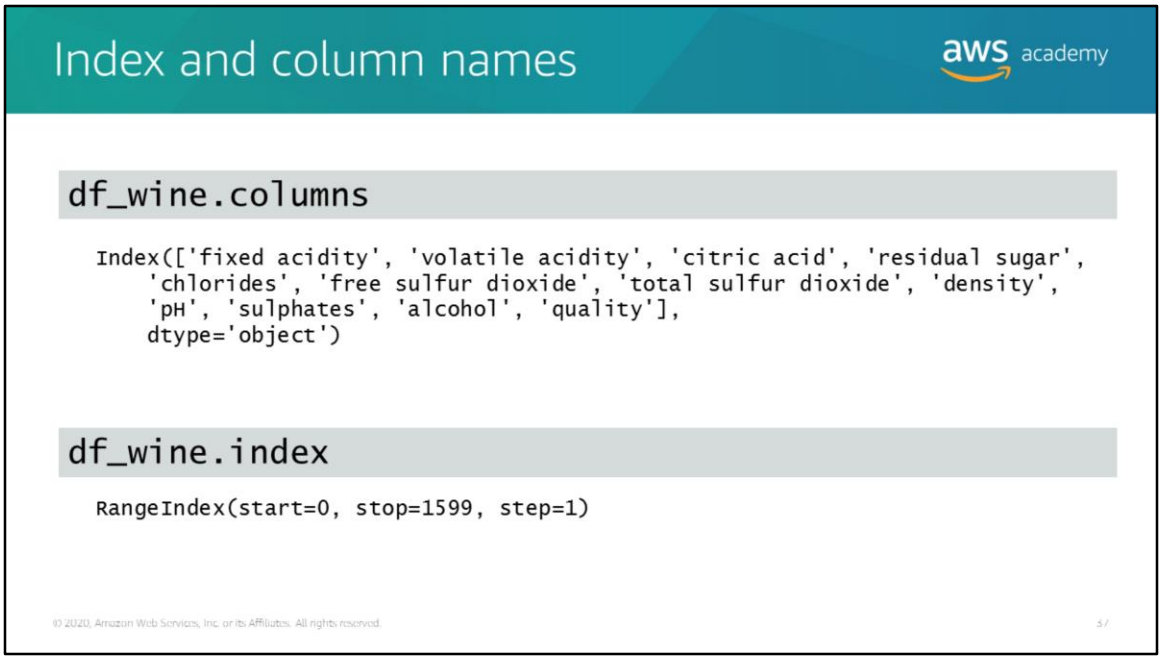


When you load data into pandas, it is stored as a *pandas DataFrame*. The **Getting started > Package overview** section of the pandas documentation describes a DataFrame as a "General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column." A more helpful way to think of a DataFrame would be like a spreadsheet or SQL table with instances (or rows) and attributes (or columns).

The *shape* attribute of a DataFrame describes the number of instances (rows) and attributes (columns).

Each column in a DataFrame is a series. A series is a one-dimensional labeled array. A series can store data of any type.

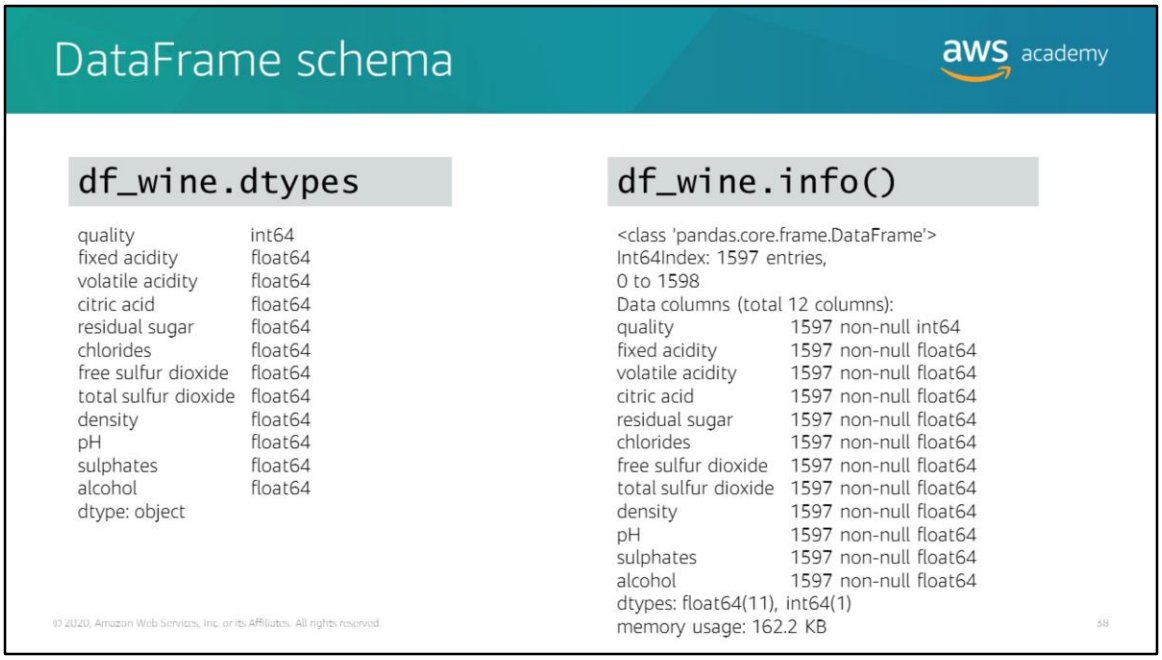
For more information about pandas data structures, see pandas data structures introduction.



Along with data, you can load a DataFrame with an *index* (row labels) and *columns* (column labels). If you load the data from a CSV with a header row, the columns are created from the first line of the file. However, you can change that behavior.

If you don't have column names in the source file, you can pass them as a parameter (as demonstrated in the following code). If you have no axis labels, then the columns are ordered in the insertion order (for Python versions later than version 3.6 and pandas versions later than version 0.23). For lower versions, the columns are a lexically ordered list of dict keys.

```
    url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/car/car.data"
    df_car = pd.read_csv(url,',',

names=['buying','maint','doors','persons','lug_boot','safety','class'
])
```

When you perform data analysis, it is important to make sure that you use the correct data types. In many cases, pandas correctly infers the correct data types when it loads data, and you can continue.

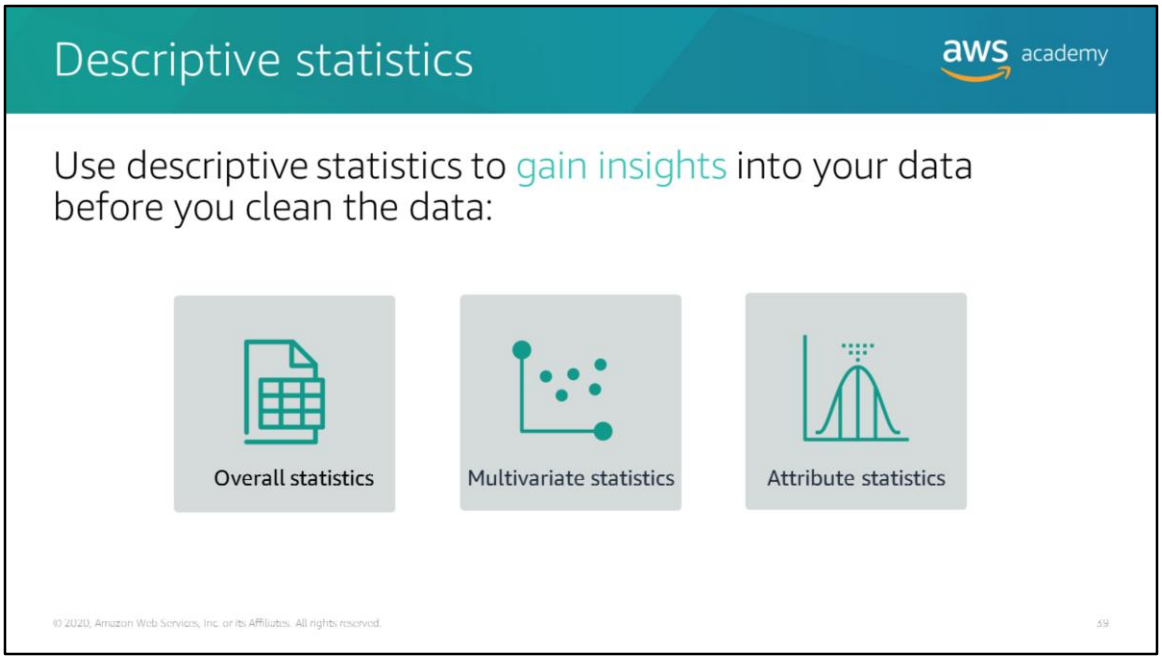
If you have domain knowledge, or access to a domain expert, they can often spot data-type issues.

You can use either **dtypes** or **info()** to obtain information about the column types, like these examples.

If you don't have the correct data types, then you must work out the reason. Often, it can be a numeric column that has missing data, or a single text value. For example, in the car dataset, the number of doors can be 2, 3, 4, 5, or more. After you analyze the data, you can convert it by using astype:

```
df_data['col'] = df_data['col'].astype('int')
```

In the example, you convert the column that is named **col** into the type **int**, and replace the column within the DataFrame **df_data**.



Now that you have your data in a format that is readable, you can perform descriptive statistics on the data to better understand it. Descriptive statistics give you valuable insight into your data so that you can more effectively preprocess the data and prepare it for your ML model. Next, you will look at how you can use descriptive statistics and why it's so important.

Descriptive statistics can be organized into different categories. *Overall statistics* include the number of rows (instances) and the number of columns (features or attributes) in your dataset. This information, which relates to the *dimensions* of your data, is important. For example, it can indicate that you have too many features, which can lead to high dimensionality and poor model performance.

Attribute statistics are another type of descriptive statistic, specifically for *numeric attributes*. They give a better sense of the shape of your attributes, including properties like the mean, standard deviation, variance, minimum value, and maximum value.

Multivariate statistics look at *relationships between more than one variable*, such as correlations and relationships between your attributes.

For cases when you have multiple variables or features, you might want to look at the

correlations between them. It's important to identify correlations between attributes because high correlation between two attributes can sometimes lead to poor model performance. When features are closely correlated and they are all used in the same model to predict the response variable, you might have problems. For example, the model loss might not converge to a minimum state. Be aware of highly correlated features in your dataset.

Statistical characteristics

aws academy

```
df_wine.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	pH	sulphates	alcohol	quality
count	1599.00	1599.00	1599.00	1599.00	1599.00	1599.00	1599.00	1599.00	1599.00	1599.00	1599.00
mean	8.32	0.53	0.27	2.54	0.09	15.87	46.47	3.31	0.66	10.42	5.64
std	1.74	0.18	0.19	1.41	0.05	10.46	32.90	0.15	0.17	1.07	0.81
min	4.60	0.12	0.00	0.90	0.01	1.00	6.00	2.74	0.33	8.40	3.00
25%	7.10	0.39	0.09	1.90	0.07	7.00	22.00	3.21	0.55	9.50	5.00
50%	7.90	0.52	0.26	2.20	0.08	14.00	38.00	3.31	0.62	10.20	6.00
75%	9.20	0.64	0.42	2.60	0.09	21.00	62.00	3.40	0.73	11.10	6.00
max	15.90	1.58	1.00	15.50	0.61	72.00	289.00	4.01	2.00	14.90	8.00

 40

Mean and *median* are two different measures that describe the extent that your data is clustered around some value or position.

Mean can be a useful method for understanding your data when the data is symmetrical.

However, your data might be skewed, or it might contain outliers. In that case, the median tends to provide the better metric for understanding your data as it relates to central tendency. For instance, your data might contain outliers with large values. Large outliers can skew the mean one way so that it doesn't serve as an accurate representation of where your values are truly centered. Outliers don't affect the median in the same way. You will learn more about outliers soon.

Statistics are available and can be viewed on numerical data by using methods such as *describe()*.

You can also view statistics on a single column by using *mean()*, *median()*, *min()*, *max()*, *skew()*, *and describe()*. For example, to find the mean for sulphates you can use *mean()*:

```
df_wine['sulphates'].mean()
```

Statistics can also be calculated for multiple columns at a time:

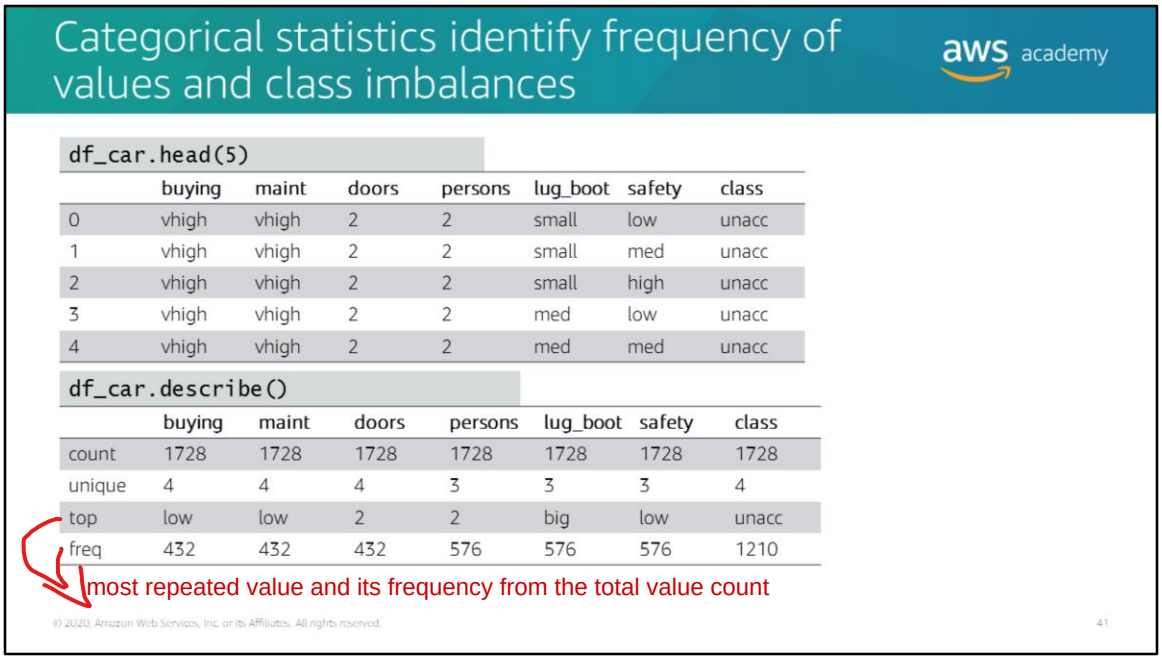
```
df_wine[['sulphates','alcohol']].mean()
```

To calculate the number of records for a category in a column, you can use *value_counts()*:

```
df_wine["quality"].value_counts()
```

pandas can also calculate statistics for each category in a column. This calculation can be done with the *groupby()* method, as follows:

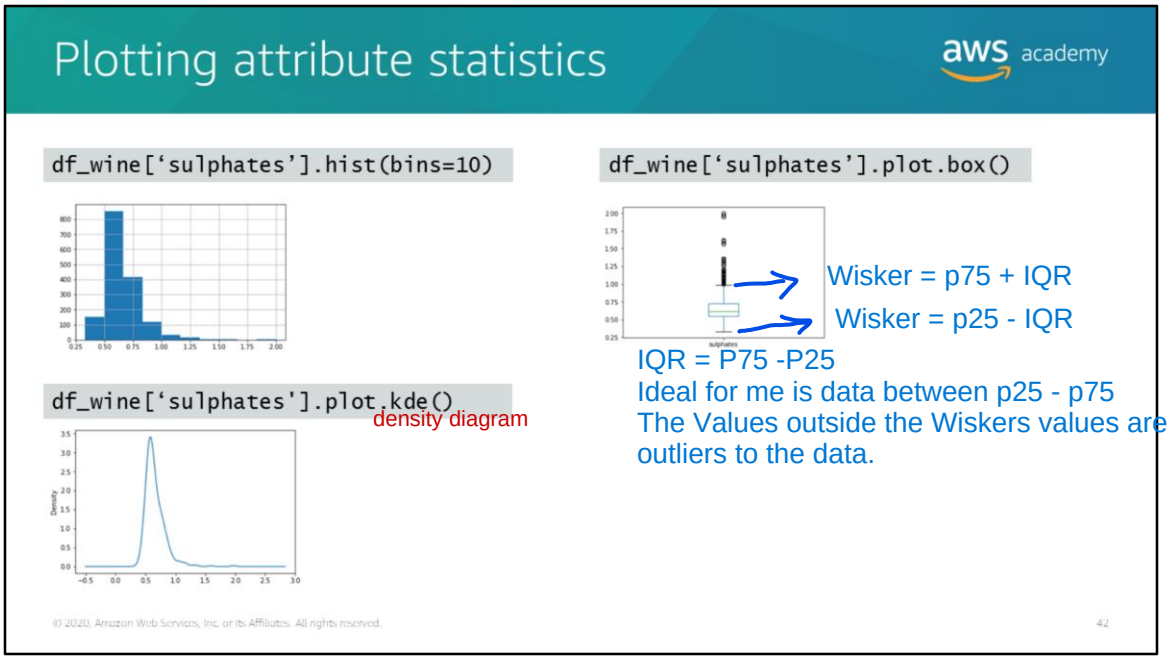
```
df_wine.groupby("quality")["sulphates"].mean()
```



For *categorical attributes*, you can look at the frequency of attribute values in your dataset. That information will give you some idea about what is inside that categorical variable.

The tables display data from the car dataset, which is made up of several categorical values: *buying*, *maint*, *lug_boot*, *safety*, and *class*. Safety can be *low*, *med* or *high*. From the *describe()* function, you can see that it has three unique values, and *low* is the most frequent. In the class column, it appears that the top value of the 4 is *unacc*. It accounts for 1210 of the 1728 values (70 percent), which might suggest an imbalance.

For a target variable that is also of categorical type, the class distribution can indicate whether your dataset has a class imbalance. Imbalanced data can mark a disproportionate ratio of your classes. For instance, suppose that your dataset is made up of credit card transactions, but only a tenth of a percent is labeled as *fraud*. In this case, your algorithm might not learn adequately to predict examples of credit card fraud.



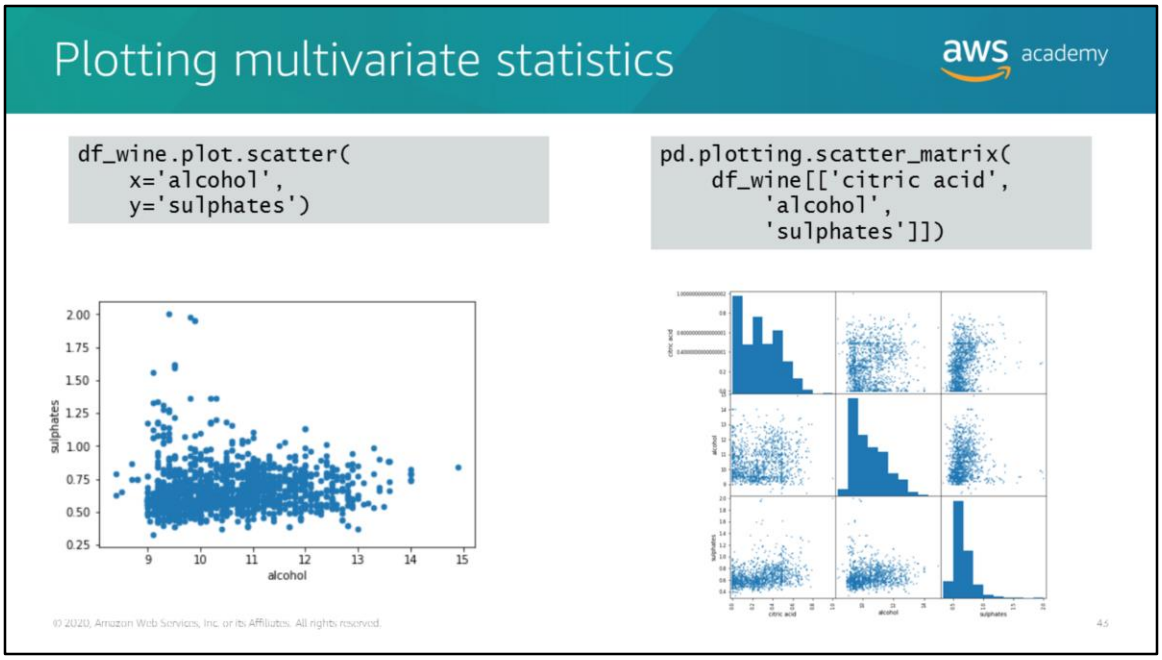
A *histogram* is often a good visualization technique for seeing the overall behavior of a particular feature. With a histogram, you can answer questions, such as: *Is the feature data normally distributed? How many peaks are in the data? Does that particular feature have any skewness?*

When you use histograms for your data visualization, values are binned. Binning is a technique that you will learn about in a later module on feature engineering. The taller peaks of the histogram indicate the most common values.

In addition to histograms, you can use *density plots* and *box plots* for numerical features so that you can get an idea of what's inside that particular feature. Like a histogram, these visualizations help you answer questions, like*: What's the range of the data? The peak of the data? Does it have any outliers? Does it have any special features?* Answering these questions can help you understand your data better. They can also help you decide whether you must do some more specialized data preprocessing.

A box plot is a method for graphically depicting groups of numerical data through their quartiles. The box extends from the Q1 to Q3 quartile values of the data, with a line at the median (Q2). Values outside that range are represented by the lines that extend from the box. These lines are sometimes called *whiskers*.

The whiskers extend from the edges of box to show the range of the data. The position of the whiskers is set by default to 1.5*IQR (IQR = Q3 - Q1) from the edges of the box. Outlier points are those points that are past the end of the whiskers.



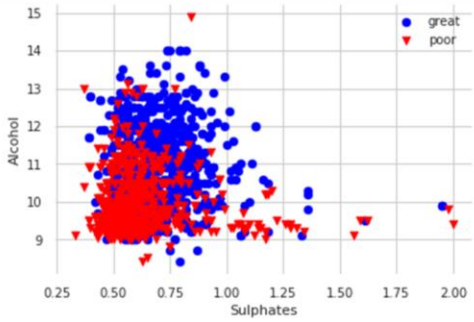
When you have more than two numerical variables in a feature dataset, sometimes you want to look at their relationship. A *scatter plot* is a good way to spot any special relationships among those variables.

For example, you have *Sulphates* and *Alcohol* in the left diagram. Those two variables are numeric, and you want to show their relationship so that the scatter plot can help you visualize that relationship. However, the plots are scattered. The correlation between the two variables might not be high because the data is scattered, but some positive relationships might exist between them.

Scatter plot matrices help you look at the relationship between multiple different features. In pandas, you can easily create scatter plot matrices that are based on the columns that you want to examine. In this case, you have three columns—it will give you the pair-wise scatter plot for any two of the columns.

Scatter plot with identification

```
high = df_wine[['sulphates','alcohol']][df_wine['quality']>5]
low = df_wine[['sulphates','alcohol']][df_wine['quality']<=5]

plt.scatter(high['sulphates'],high['alcohol'],s=50,c='blue',marker='o',label='great')
plt.scatter(x=low['sulphates'],y=low['alcohol'],s=50,c='red',marker='v',label='poor')
```



 44

With a scatter plot, you might want to identify special regions that a particular subset of data belongs to.

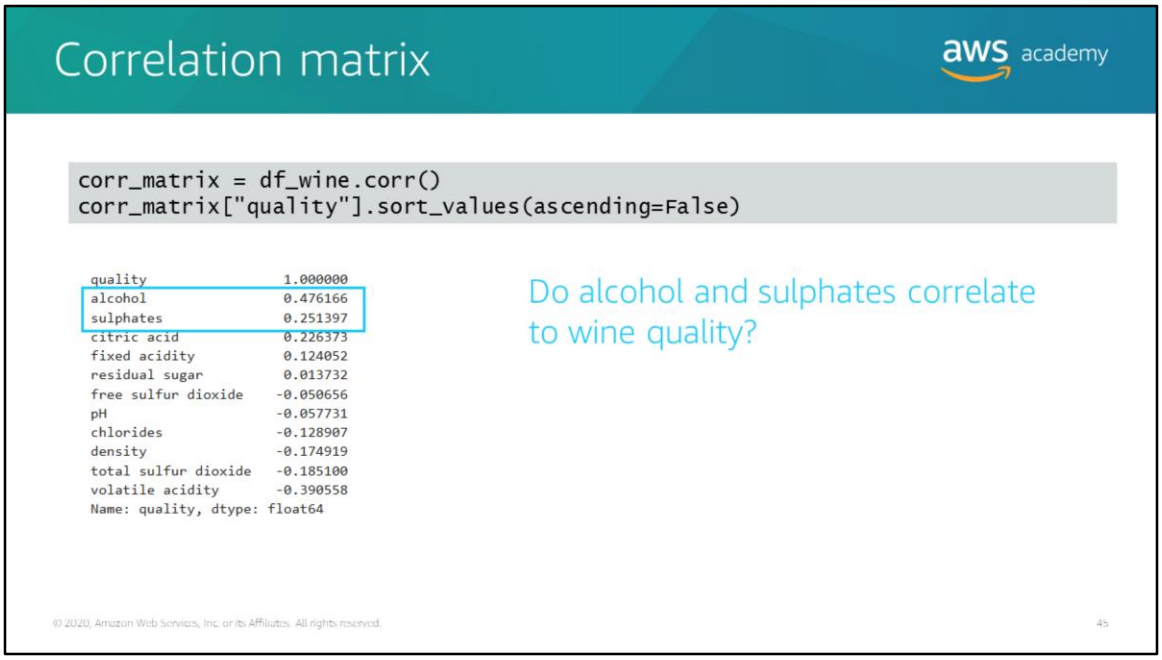
Does a relationship exist between alcohol, sulphates, and quality? You can plot out those values against *good* (quality > 5) and *poor* (quality <=5) wine, such as in the following example:

```
high = df_wine[['sulphates','alcohol']][df_wine['quality']>5]

low = df_wine[['sulphates','alcohol']][df_wine['quality']<5]

plt.scatter(high['sulphates'],high['alcohol'],s=50,c='blue',marker='o
',label='great')

plt.scatter(x=low['sulphates'],y=low['alcohol'],s=50,c='red',marker='
v',label='poor')

plt.xlabel('Sulphates')

plt.ylabel('Alcohol')

plt.legend()
```



```
plt.grid()

plt.show()
```

Plotting gives you an idea of how useful particular variables can be, if you use them for a classification problem.

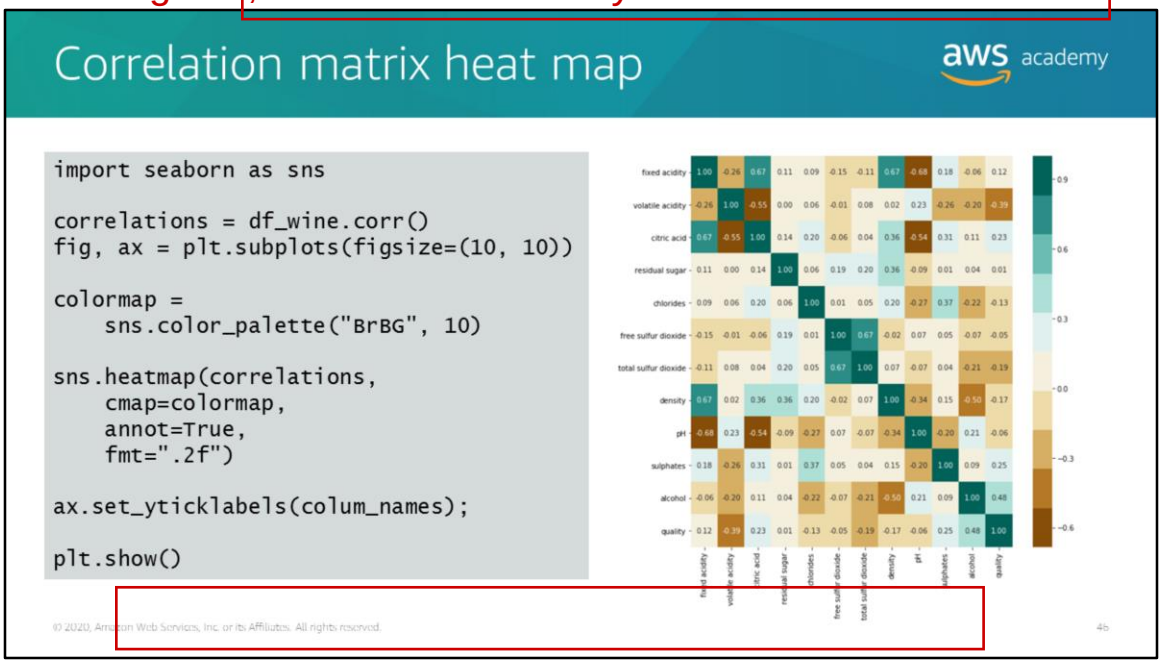


How can you quantify the linear relationship among the variables that you placed in a scatter plot?

A *correlation matrix* is a good tool in this situation, because it conveys both the strong and weak linear relationships among numerical variables.

Correlation can be as high as one, or as low as minus one. When the correlation is one, it means that those two numerical features are perfectly correlated with each other. It's like saying that *Y is proportional to X*. When the correlation of those two variables is minus one, it's like saying that *Y is proportional to minus X*. Any linear relationship in between can be quantified by using the correlation. If the correlation is zero, it means that no linear relationship exists, but it does *not* mean that no relationship exists. It's only an indication that the two variables have no linear relationship.

When I have green regions in areas that aren't in the diagonal, this is when to worry



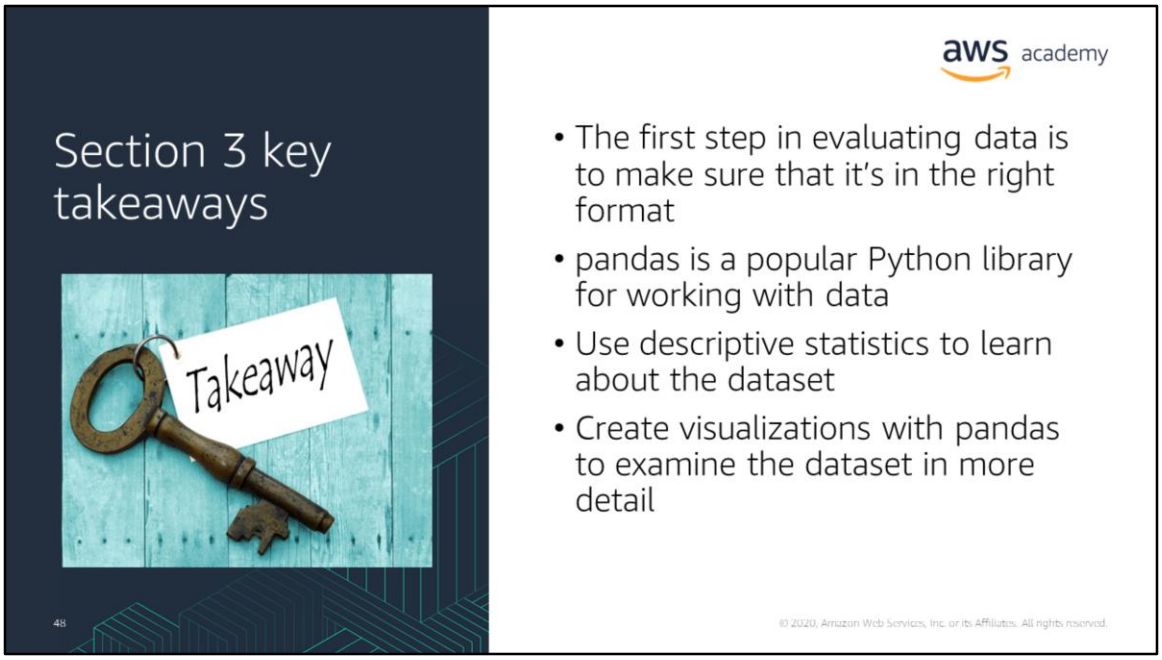
However, looking at a number is not always straightforward. It is often easier to view the numbers if they are represented with colors. Now, consider a *heat map*. You have the highest number (which is 1) in dark green, and –1 in dark brown. The color gives you both the positive and negative directions, and how strong the correlations are. Don't worry if you cannot see the text in the diagram, the point of the heat map is spotting correlations with the color!

You can use the Seaborn's heat map function to show the correlation matrix.

From the chart, you can see some correlation between citric acid and fixed acidity. You might expect that correlation in wine, because citric acid contributes to the acidity of the wine.

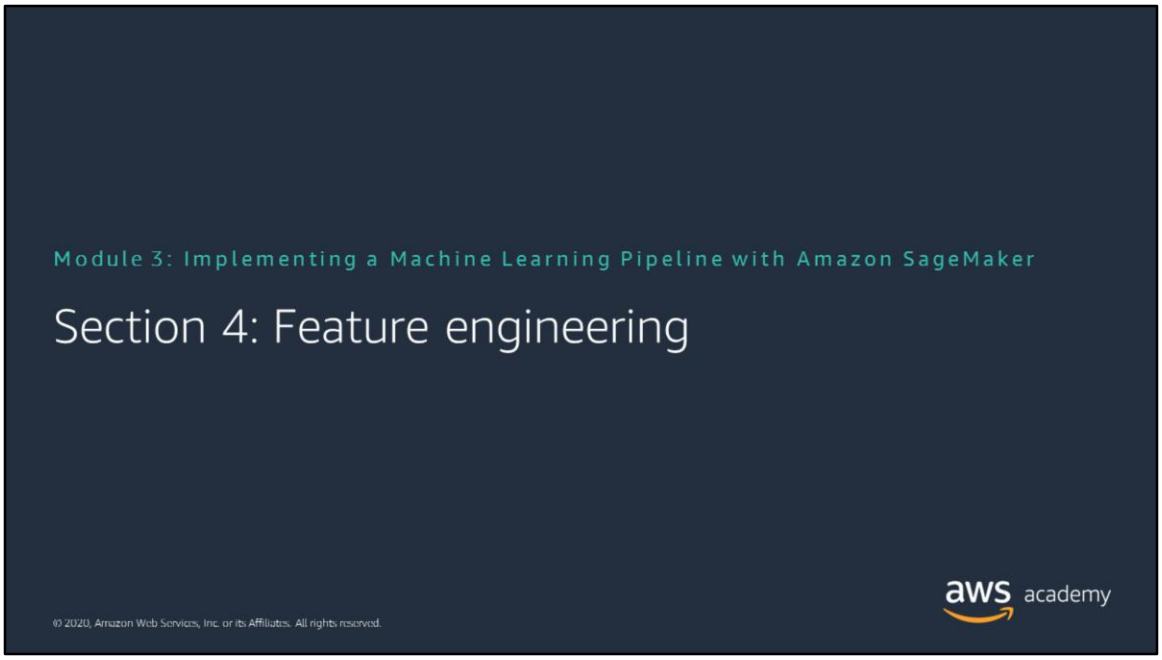
However, little correlation exists between fixed acidity and pH. pH is a measurement of the *strength* of those acids present, but fixed acidity is a measure of the *quantity*. In your dataset, no correlation is apparent.

You will now complete Module 3 – Guided Lab 2: Visualizing Data.

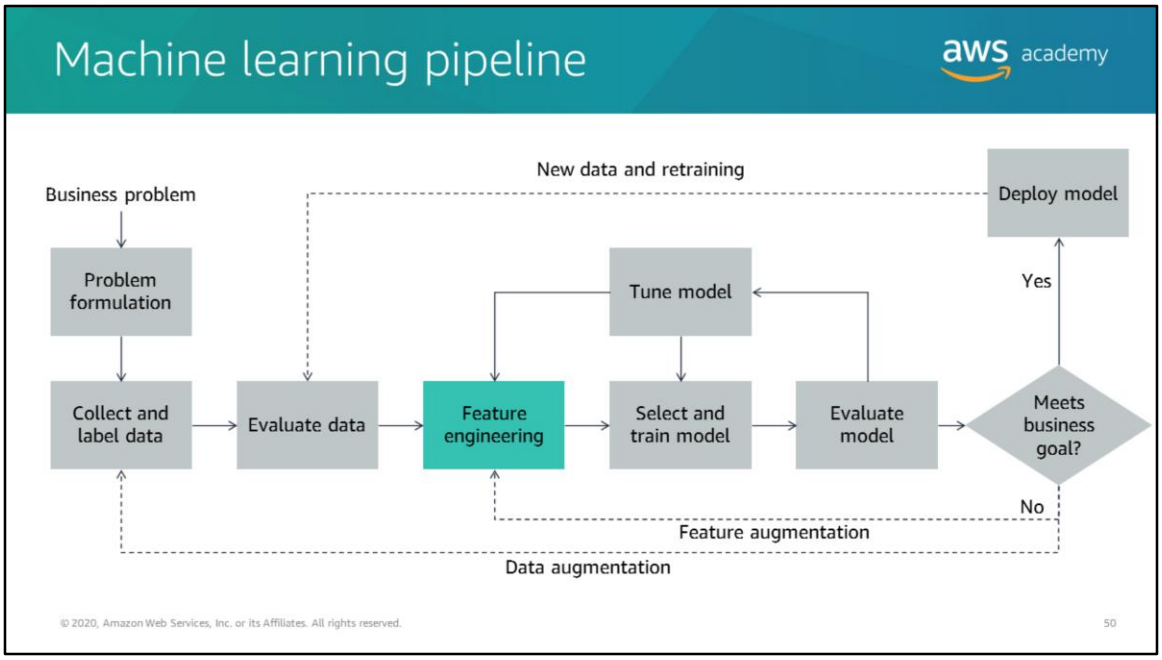


Some key takeaways from this section of the module include:

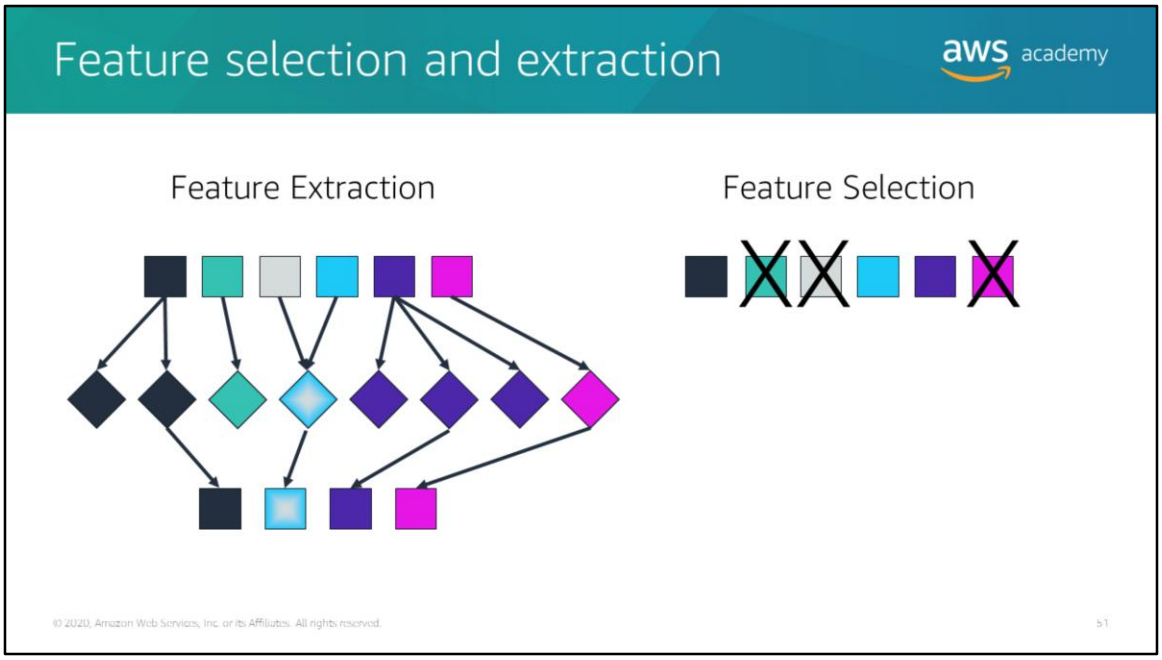
- The first step is to get your data into a format that can be used easily.
- pandas is a popular and useful Python library for working with data.
- Descriptive statistics help you gain insights into the data.
- Use visualizations to examine the dataset in more detail.



Introducing Section 4: Feature engineering.



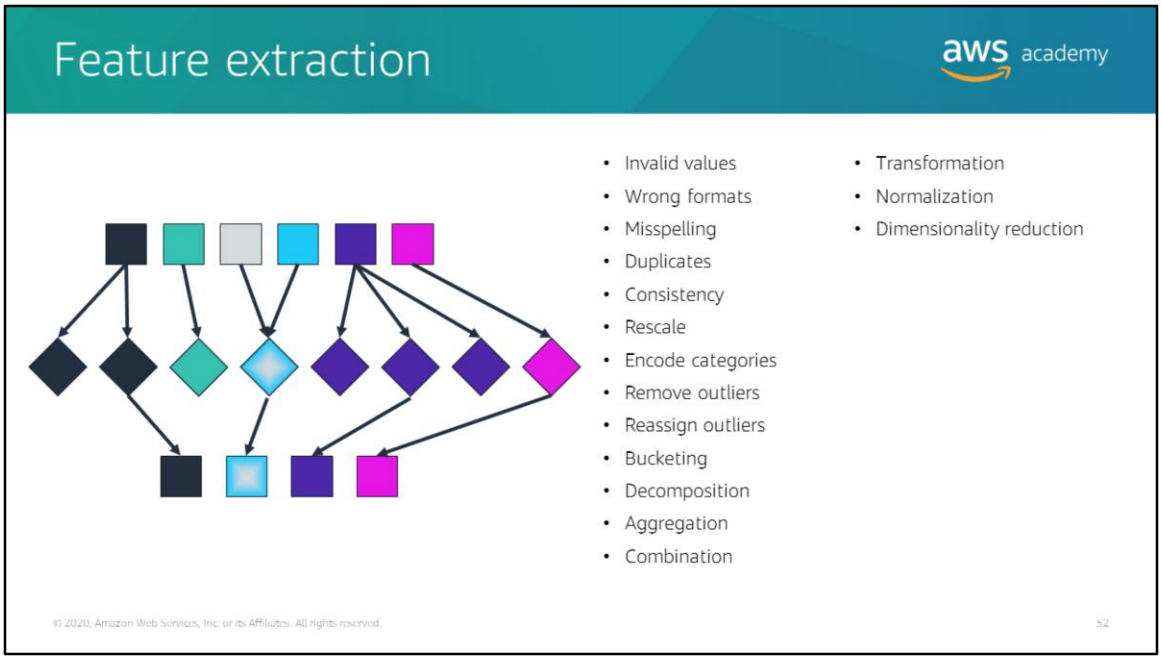
In this section, you look at the key task of feature engineering, which selects the columns of data that make the most impact in the model.



Two things can make your models more successful: *feature selection* and *feature extraction or creation*.

Feature selection is about selecting the features that are most relevant and discarding the rest. Feature selection is applied to prevent either redundancy or irrelevance in the existing features, or to get a limited number of features to prevent overfitting.

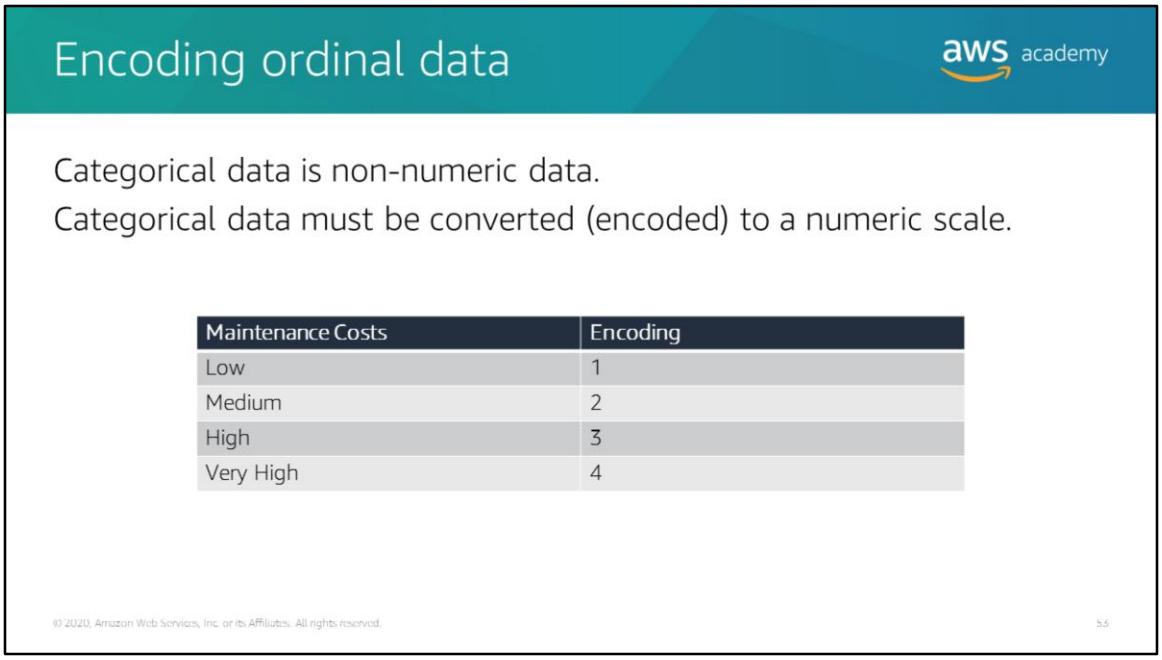
Feature extraction is about building up valuable information from raw data by reformatting, combining, and transforming primary features into new ones. This transformation continues until it yields a new set of data that can be consumed by the model to achieve the goals.



Feature extraction covers many activities that range from handling missing data to converting text data into numerical data. Although the list is not exhaustive, it should give you some idea of the data handling that's needed to get data into a useful state.

Many of the tasks are no different from any other job that works with data. You must make sure that data is in the correct format, with consistent representations of data, spelling, and the like. You might combine data, extract data into multiple columns, or remove them altogether.

Tasks that are specific to machine learning include being be able to convert text columns to numerical values, deciding how to handle outliers, and potentially rescaling your data. In this section, you will learn about some of the more common tasks.

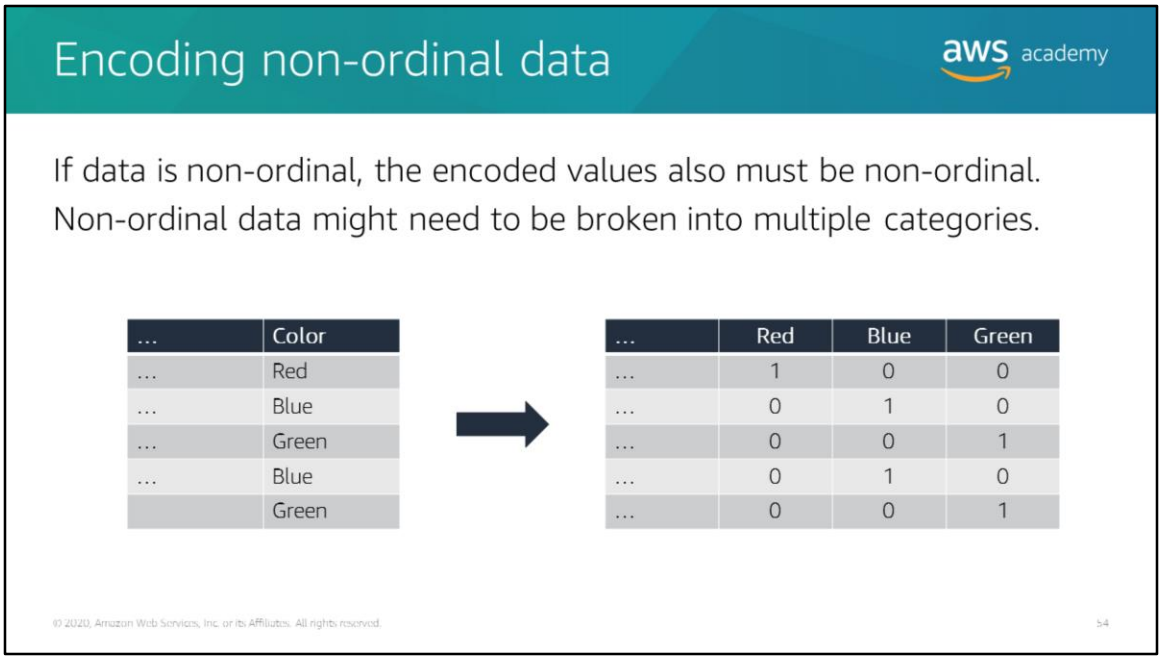


Most ML algorithms work best with numerical data. Therefore, you must make sure that all columns in your dataset contain numeric data by converting or encoding the data.

You might need to make several passes through the dataset before you can encode it. For example, you might have variability in the text values (for example, some rows might contain both *Medium* and *Med*).

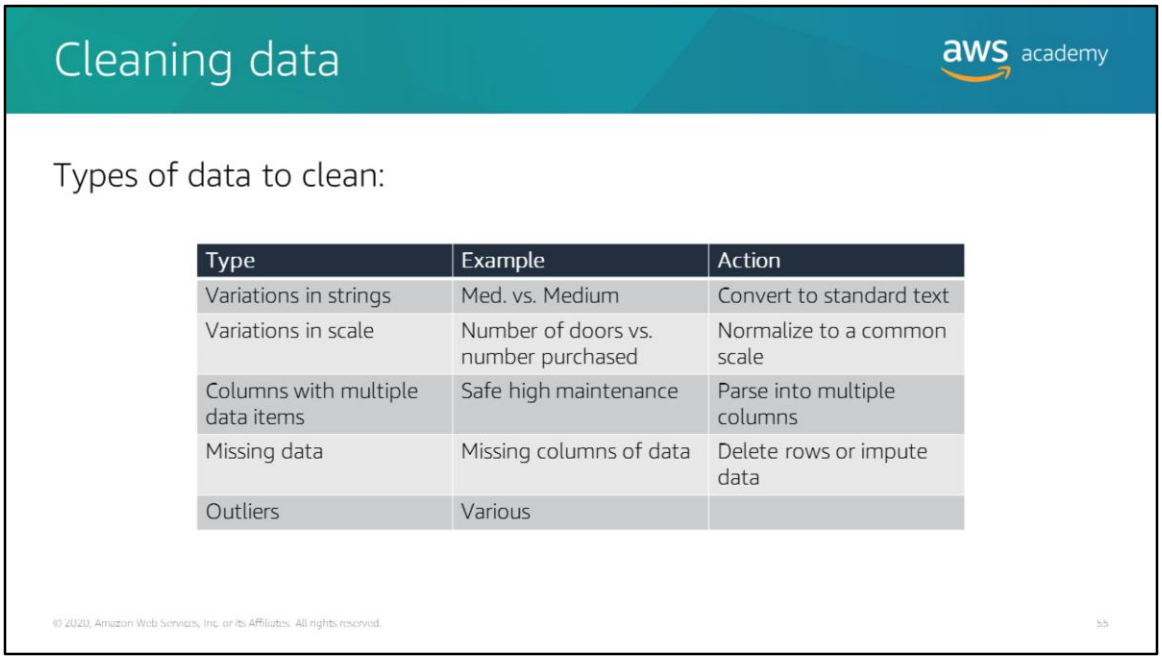
If the categorical data has order to it, you can encode the text into numerical values that capture this ordinal relationship. For data that shows maintenance costs, you might encode *Low* to *1*, *Medium* to *2*, *High* to *3*, and *Very High* to *4*.

Tools such as Scikit-learns and pandas can be used to encode your categorical data after you make sure that it is all uniform.



If the categorical data doesn't have any order to it, then you must break the data into multiple columns. You don't want to introduce an ordinal relationship to the data that isn't present. For example, suppose you assign a value of *1* to the first color (like *red*) and *2* to the next value (like *blue*). Then, the model might interpret blue as more important than red, because it has a higher numeric value.

A better way is to encode non-ordinal data into multiple columns or features. Think of the new features like a check box. In the example, three features are generated, and *1* represents that the instance has that feature, which is its color.

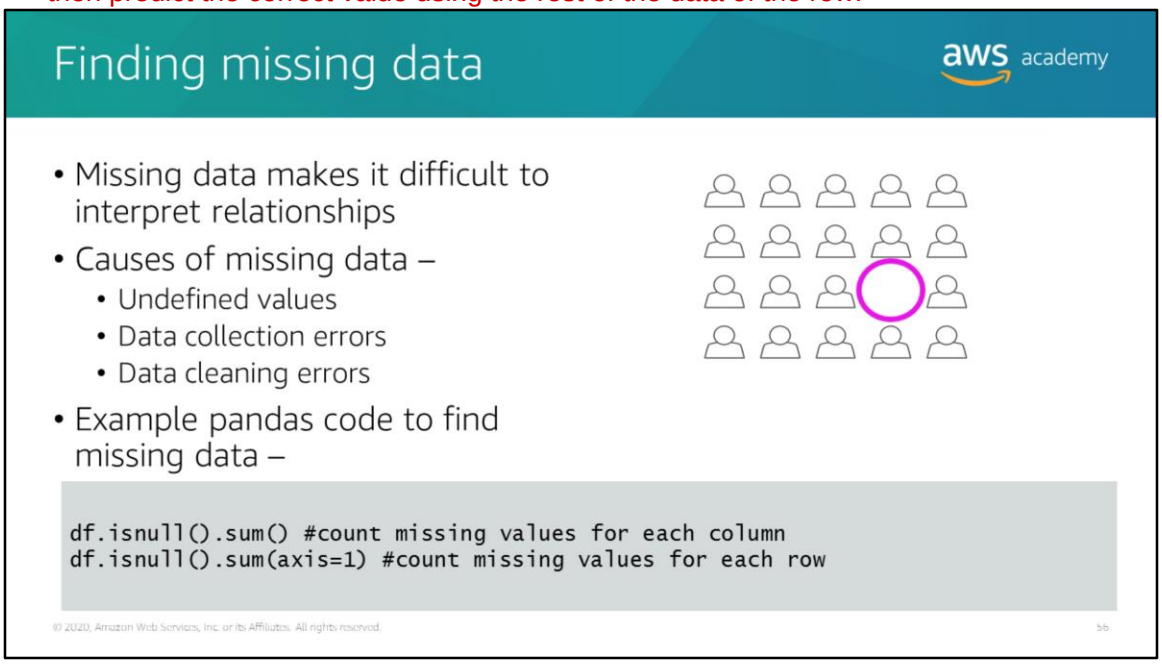


In addition to converting string data to numerical data, you must clean your dataset for several other potential problem areas. Before you encode the string data, you must make sure that the strings are all consistent. You also must make sure that variables use a consistent scale. For example, if one variable describes the number of doors in a car, the scale is probably between *2* and *8*. If another variable describes the number of cars of a particular type sold in the US state of California, the scale is probably in the thousands.

Some data items might also capture more than one variable in a single value. For example, the dataset might include a variable that combines safety and maintenance into a single variable (*safe high-maintenance*). If you want to train your ML system for both variables, you must split that variable into two variables.

You might also encounter datasets that are missing data for some variables, and some datasets that include outliers. In this section, you learn techniques for dealing with these types of datasets.

MICE(multiple imputation by chained equations): Is a way to fill missing values with random data then predict the correct value using the rest of the data of the row.

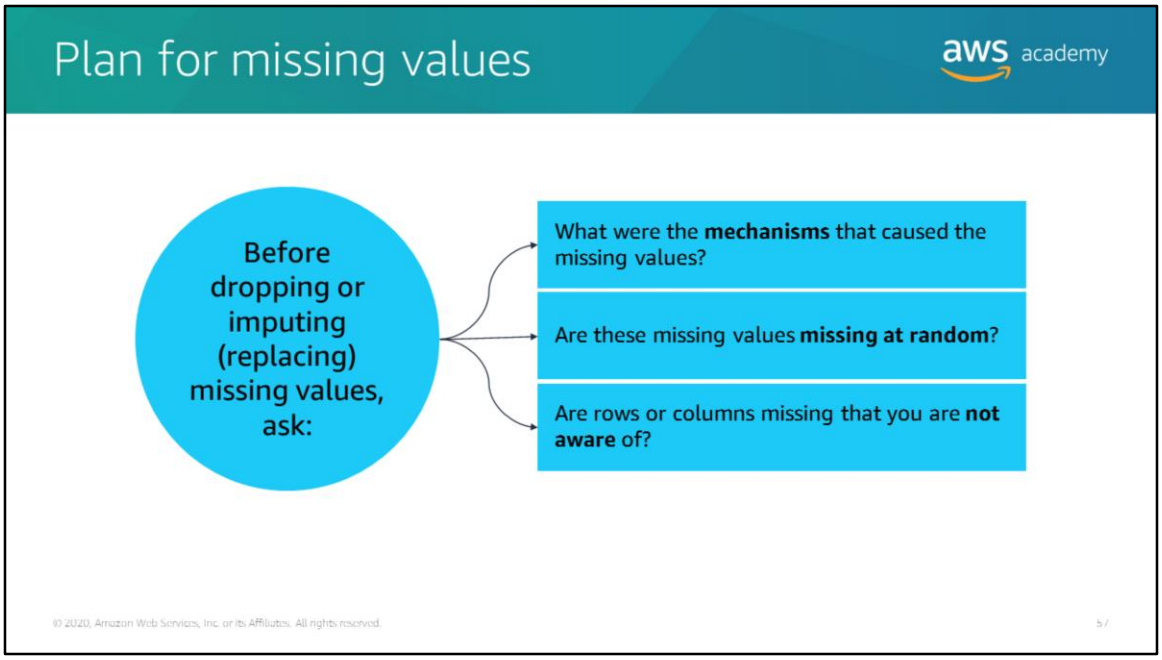


You might also find that you have missing data. For example, some columns in your dataset might be missing data because of a data collection error. Perhaps data wasn't collected on a particular feature until the data collection process was well under way. Missing data can make it difficult to accurately interpret the relationship between the related feature and the target variable. Thus, regardless of what caused the data to be missed, it is important that you deal with the issue.

Unfortunately, most ML algorithms can't deal with missing values automatically. You must use human intelligence to update missing values with something meaningful and relevant to the problem.

Most Python libraries for data manipulation include functions for finding missing data. The following example shows how to use the pandas isnull function to find missing data:

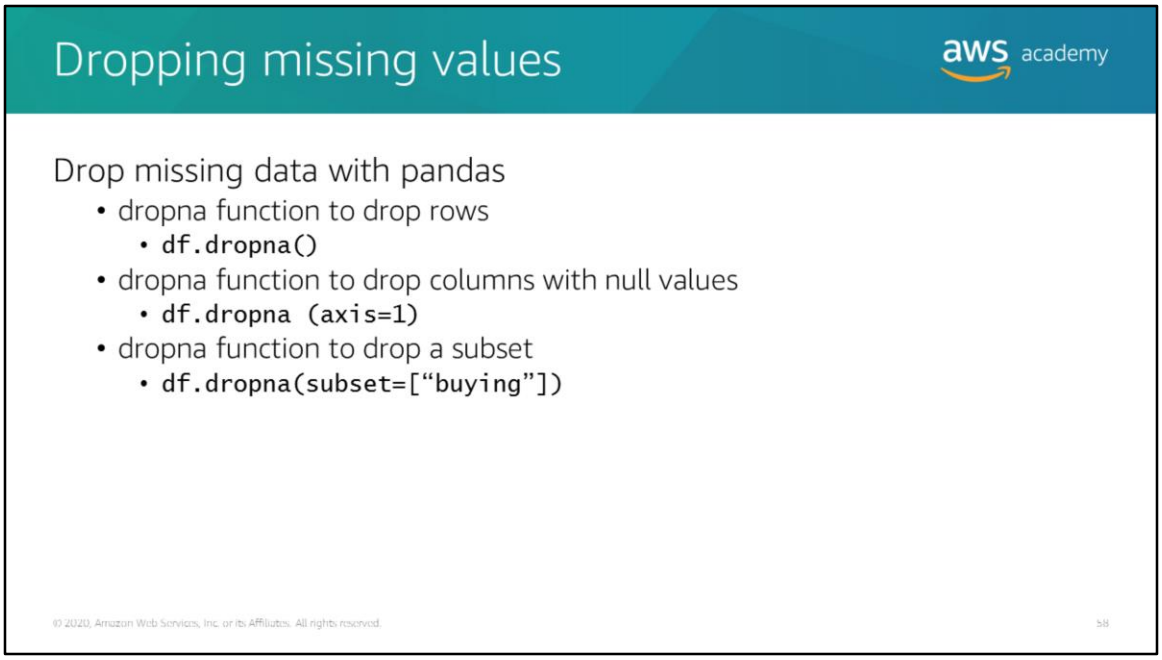
```
df.isnull().sum() #count missing values for each column
df.isnull().sum(axis=1) #count missing values for each row
```



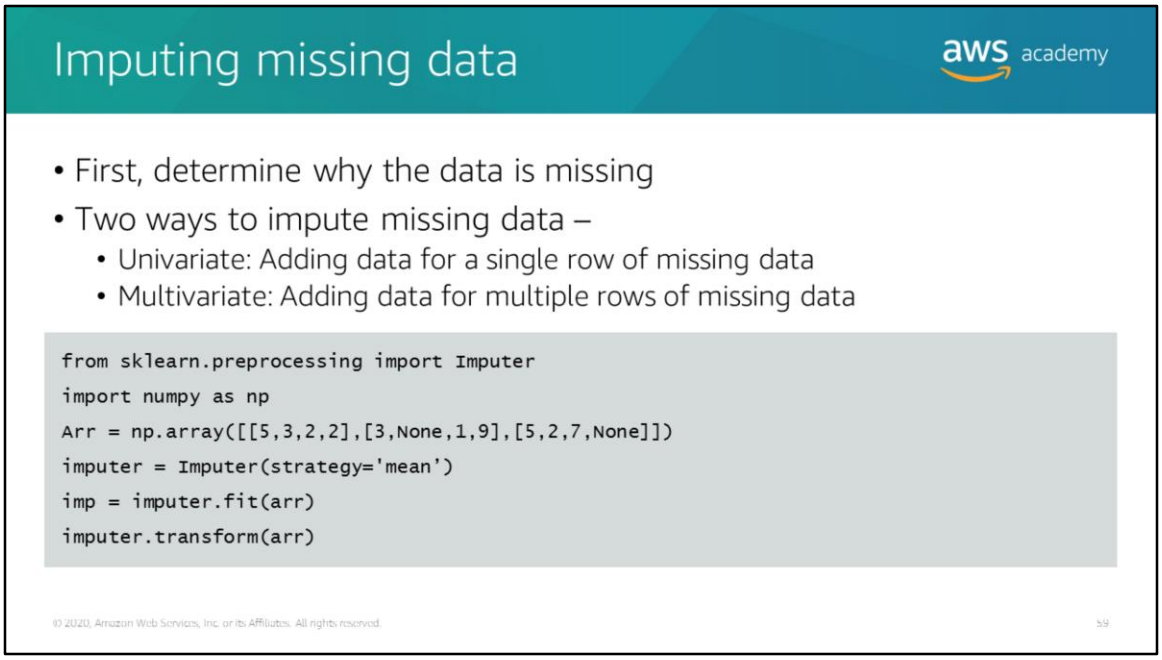
How do you decide whether you should drop or impute missing values? This question is answered in part by better understanding what caused those values to be missing. You also must know how much data the missing values represent within your larger dataset.

For instance, suppose that the missing values are randomly spread throughout your dataset and don't represent a larger portion of its row or column. In this case, *imputation* is most likely the better option.

Conversely, you might have a column or row that has a large percentage of missing values. In this case, *dropping* the entire row or column would be preferred over imputation.



If you decide to drop rows with missing data, you can use built-in functions. For example, the pandas dropna function can drop all rows with missing data, or you can drop specific datavalues by using a subset.



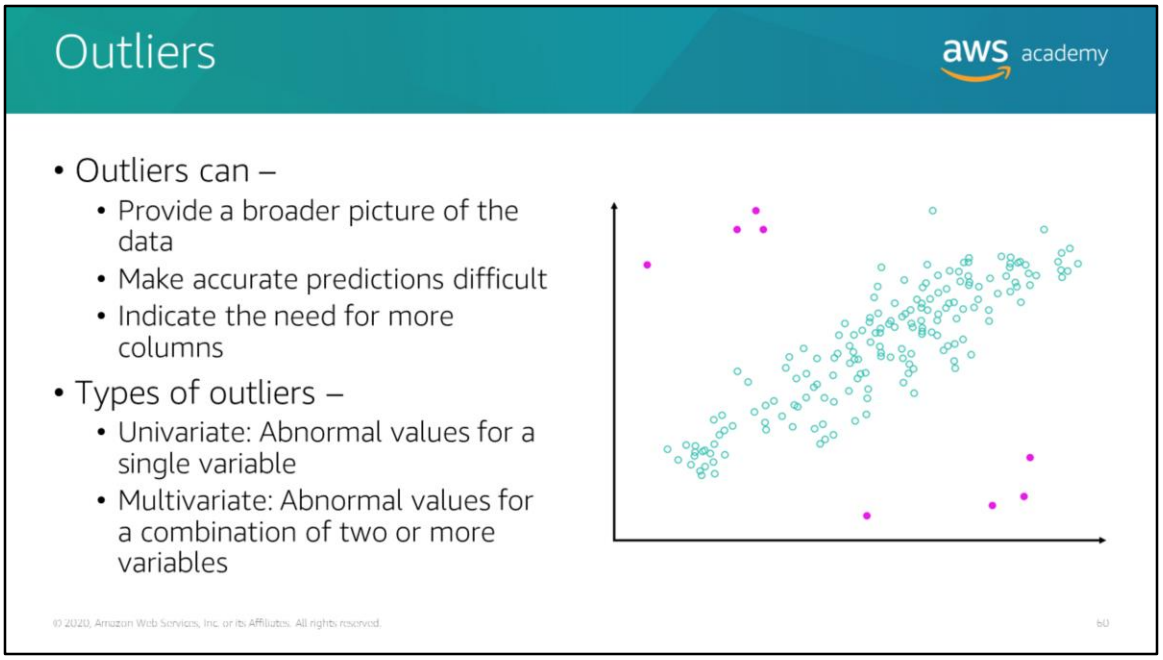
As an alternative to dropping missing values, you can impute values for those missing values.

You can impute a missing value in different ways. For categorical variables, you usually replace the missing value with the mean, median, or most frequent values. For numerical or continuous variables, you typically use mean or median.

You can impute either a single row of missing data (univariate) or multiple rows (multivariate) of missing data. Consider the univariate example that is shown. You can use the Scikit-Learn imputer function to impute some missing values. It's a small dataset, but you do have two missing values.

Here, you use the strategy of mean to impute the missing value. With this strategy, you calculate the mean—the mean of 3 and 2 is 2.5—and impute that value for the missing value.

Some data libraries include an impute package that provides more complex ways to impute data. They might include K nearest neighbor, soft impute, multiple imputation by chain equations, and a few other more advanced imputation methodologies.

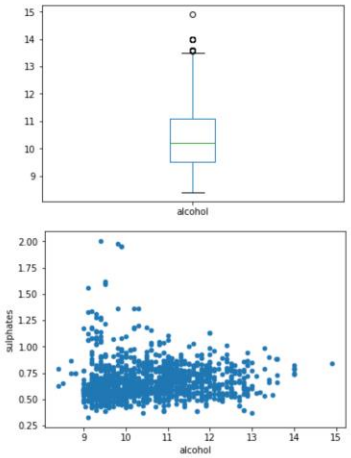


You might also need to clean your data because of any outliers that might exist. Outliers are points in your dataset that lie at an abnormal distance from other values. They are not always something that you want to clean up, because they can add richness to your dataset. However, they can also make it harder to make accurate predictions. The outliers affect accuracy because they skew values away from the other more normal values that are related to that feature. In addition, an outlier might indicate that the data point belongs to another column.

You can think of outliers as falling into two broad categories. The two categories are a single variation for a single variable (univariate) and a variation of two or more variables (multivariate).

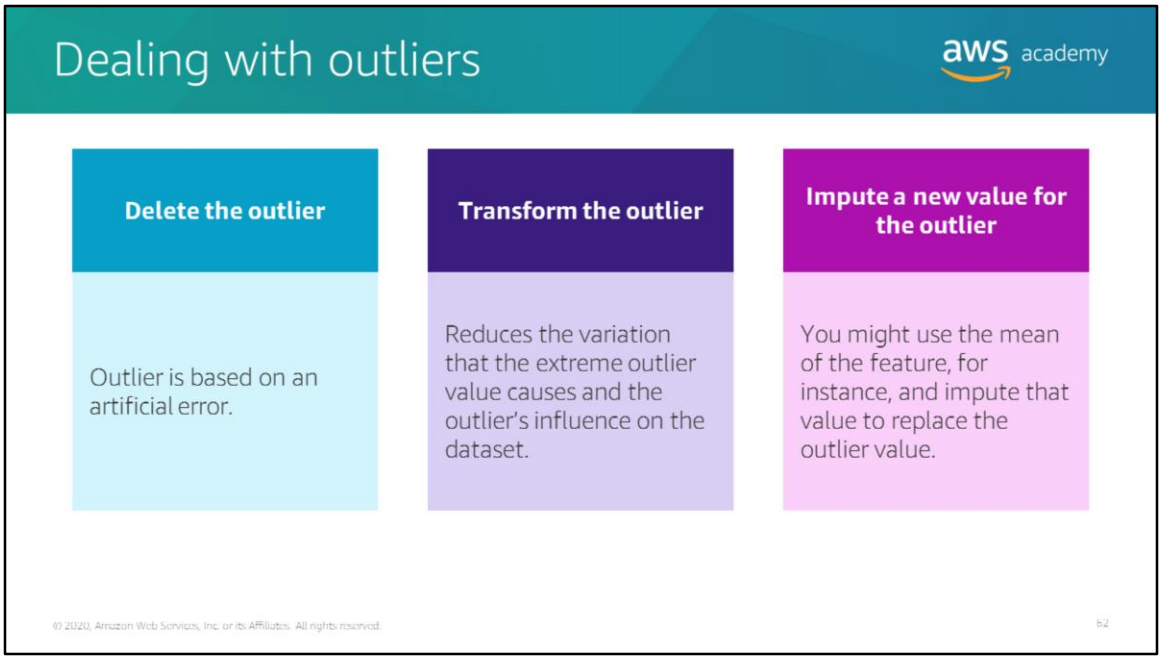
Finding outliers

- Box plots show variation and distance from the mean
  - Example to the right shows a box plot for the amount of alcohol in a collection of wines
- Scatter plots can also show outliers
  - Example to the right shows a scatter plot that shows the relationship between alcohol and sulphates in a collection of wines



One of the more common ways to find univariate outliers is with a box plot. A box plot shows how far a data point is from the mean for that variable. The box in the plot shows the data values within two quartiles of the mean.

A scatter plot can be an effective way to see multivariate outliers. For example, this diagram shows the amount of sulphates and alcohol in a collection of wines. With the scatter plot, you can quickly visualize whether multivariate outliers exist for two variables.

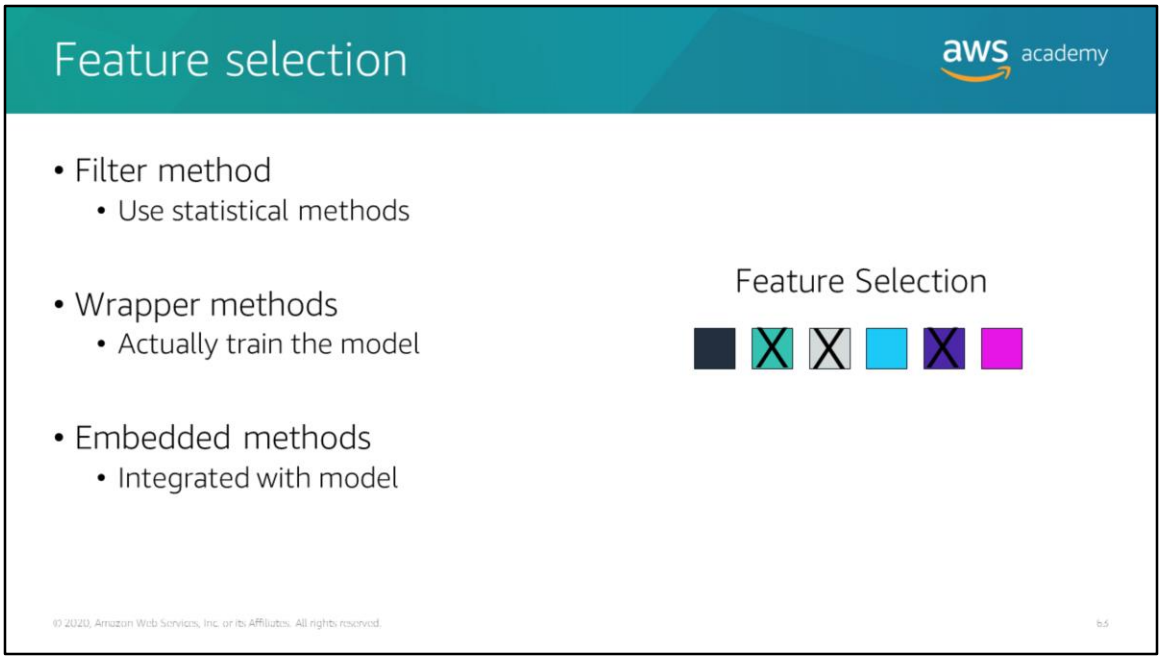


The origin of your outlier is likely to determine how you handle it during this preprocessing phase of the pipeline, or later during feature engineering. You can handle outliers with several different approaches. They include, but are not limited to:

Deleting the outlier: This approach might be a good choice if your outlier is based on an artificial error. *Artificial error* means that the outlier isn't natural and was introduced because of some failure—perhaps incorrectly entered data.

Transforming the outlier: You can transform the outlier by taking the natural log of a value, which in turn reduces the variation that the extreme outlier value causes. Therefore, it reduces the outlier's influence on the overall dataset.

Imputing a new value for the outlier: You can use the mean of the feature, for instance, and impute that value to replace the outlier value. Again, this would be a good approach if an artificial error caused the outlier.



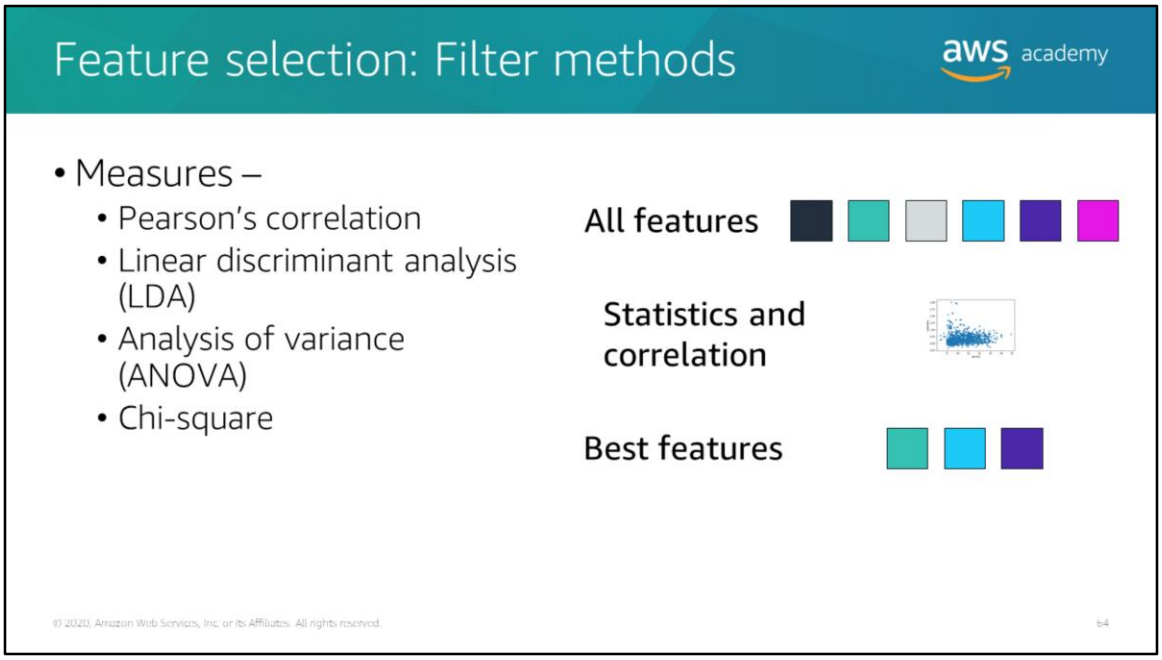
After you extract features, you must select the most appropriate features to train your model. Three main feature selection methods are available: filter, wrapper, and embedded.

Filter methods use statistical methods to measure the relevance of features by their correlation with the target variable.

Wrapper methods measure the usefulness of a subset of features by training a model on it and measuring the success of the model.

Filter methods are faster and cheaper than wrapper methods because they do not involve training the models repeatedly. Wrappers typically find the best subset of features, with a risk of overfitting compared to using subsets of features from filter methods.

Embedded methods are algorithm-specific and might use a combination of both.

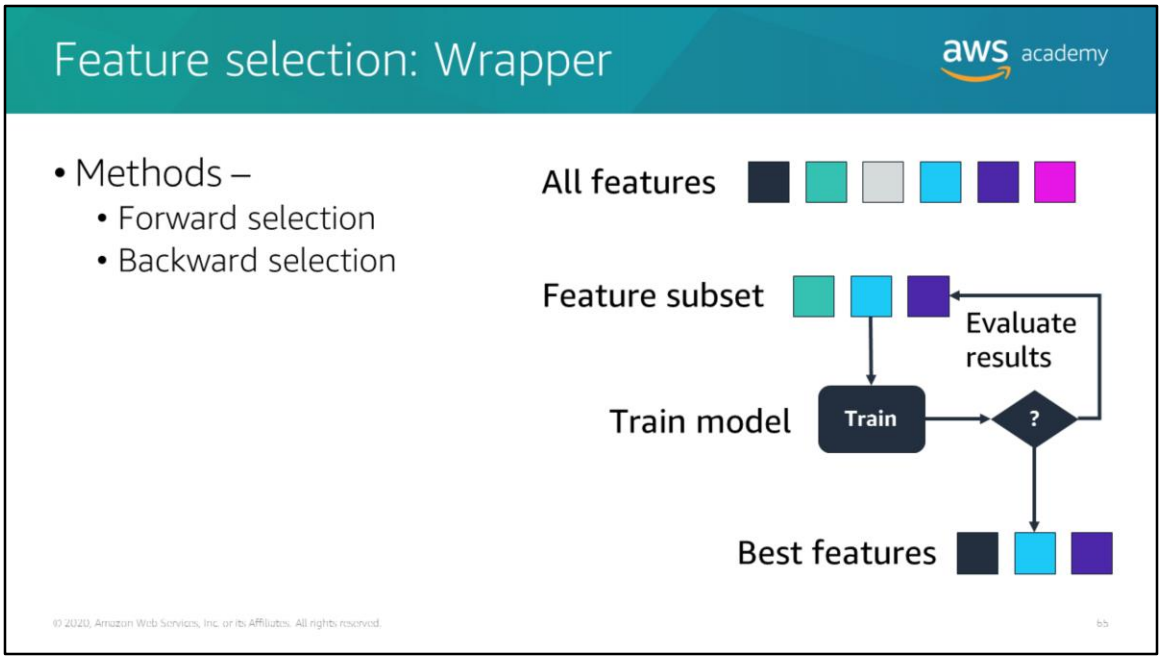


Filter methods use a proxy measure instead of the actual model's performance. Filter methods are fast to compute, and they still capturing the usefulness of the feature set. Common measures include:

- *Pearson's correlation coefficient* – Measures the statistical relationship or association between two continuous variables.
- *Linear discriminant analysis (LDA)* – Is used to find a linear combination of features that separates two or more classes.
- *Analysis of variance (ANOVA)* – Is used to analyze the differences among group means in a sample.
- *Chi-square* – Is a single number that tells how much difference exists between your observed counts and the expected counts, if no relationship exists in the population.

Filters are usually less computationally intensive than wrappers, but they produce a feature set that is not tuned to a specific type of predictive model. This lack of tuning means that a feature set from a filter is more general than the set from a wrapper. The result is usually lower prediction performance than a wrapper. However, the feature set doesn't contain the assumptions of a prediction model. Thus, it's more useful for exposing the relationships between the features. Many filters provide a feature ranking instead of an explicit best feature subset, and the cutoff point in the ranking is chosen through cross-validation. Filter

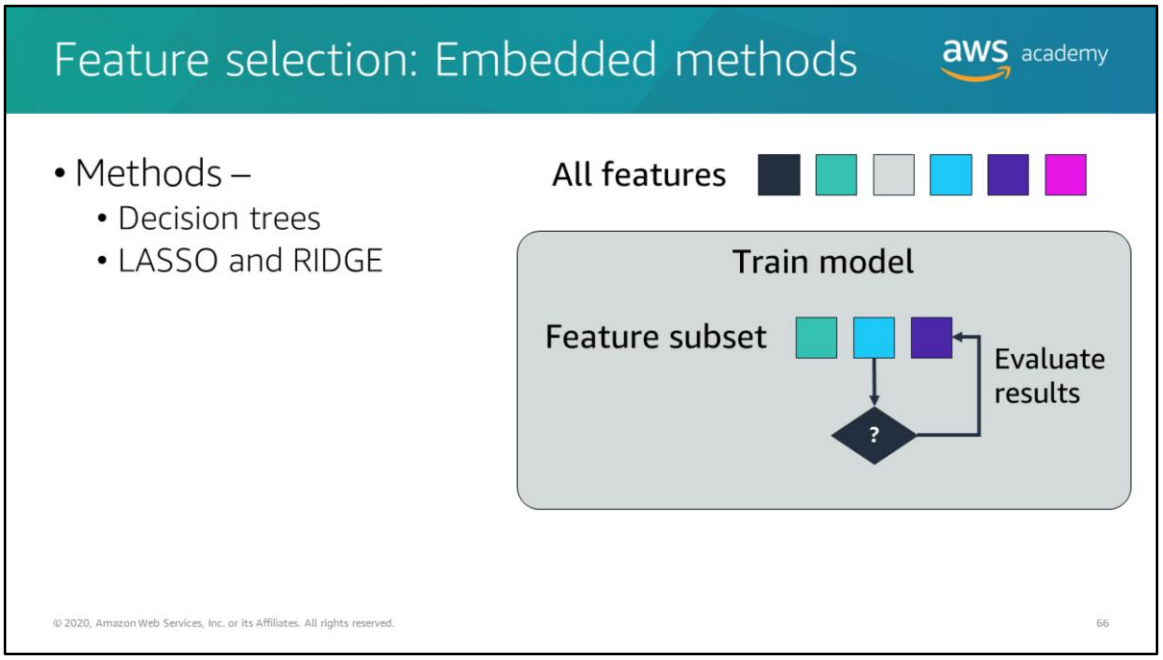
methods have also been used as a preprocessing step for wrapper methods, which enables a wrapper to be used on larger problems.



Wrapper methods use a predictive model to score feature subsets. Each new subset is used to train a model, which is tested on a hold-out set. To get the score for that subset, count the number of mistakes that are made on that hold-out set (the error rate of the model). As wrapper methods train a new model for each subset, they are computationally intensive. However, they usually provide the best-performing feature set for that type of model or typical problem.

Forward selection starts with no features, and adds them until the best model is found.

Backward selection starts with all features, drops them one at a time, and selects the best model.



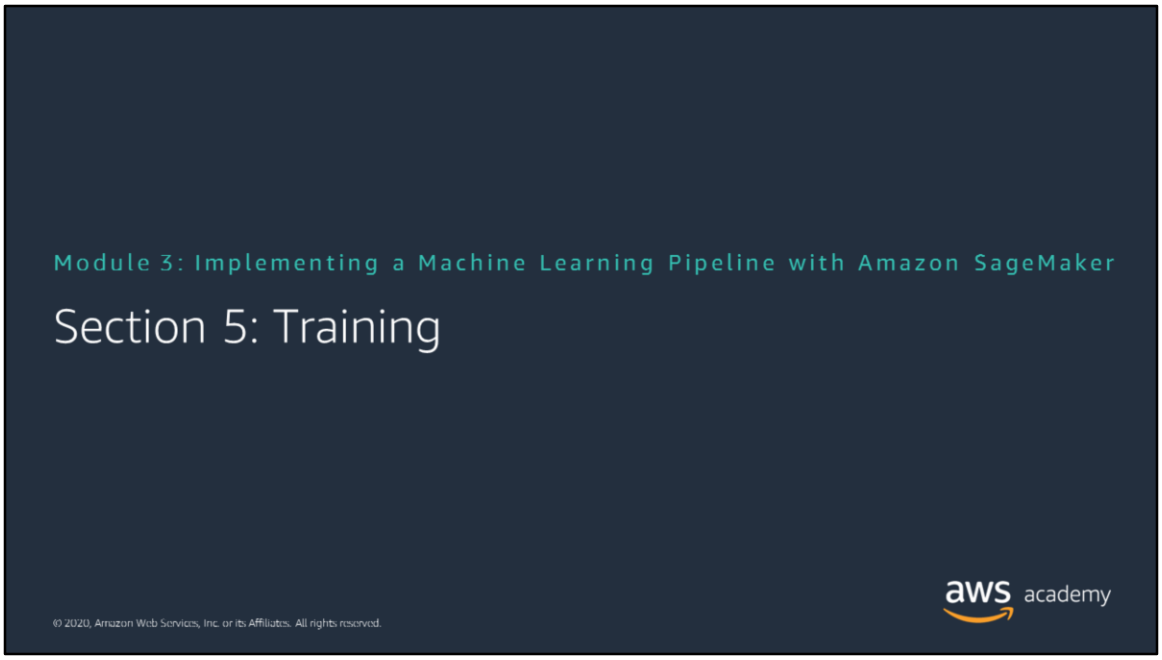
Embedded methods combine the qualities of filter and wrapper methods. They are implemented from algorithms that have their own built-in feature selection methods.

Some of the most popular examples of these methods are LASSO and RIDGE regression, which have built-in penalization functions to reduce overfitting.

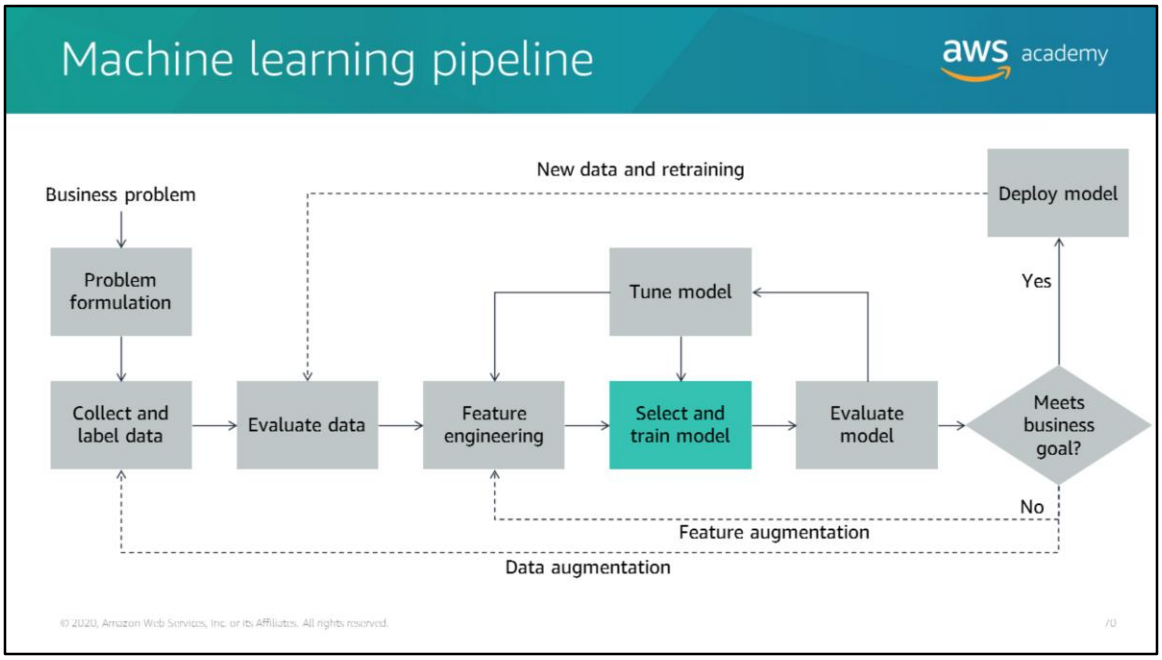
You will now complete Module 3 – Guided Lab 3: Encoding Categorical Variables.

Some key takeaways from this section of the module include:

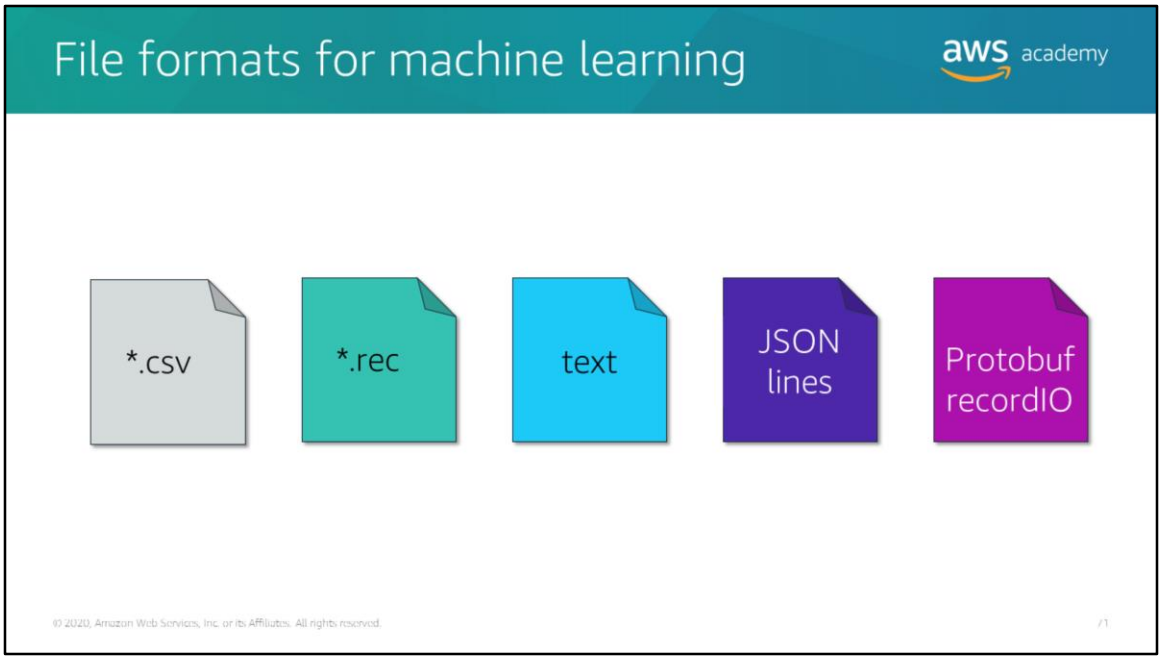
- Feature engineering involves selecting or extracting the best features for machine learning.
- Preprocessing gives you better data to work with. Better data typically provides better results.
- Preprocessing has two categories:
    - Converting data (to numerical values)
    - Cleaning up dirty data (removing missing data, cleaning outliers)
- How you deal with dirty data affects your model, pay attention to missing data and outliers.
- Develop a strategy for dirty data. Replace or delete rows with missing data.
    - Delete, transform, or impute new values for outliers.



Introducing Section 5: Training.



In this section, you will learn how to select a model and train it with the data that you preprocessed.



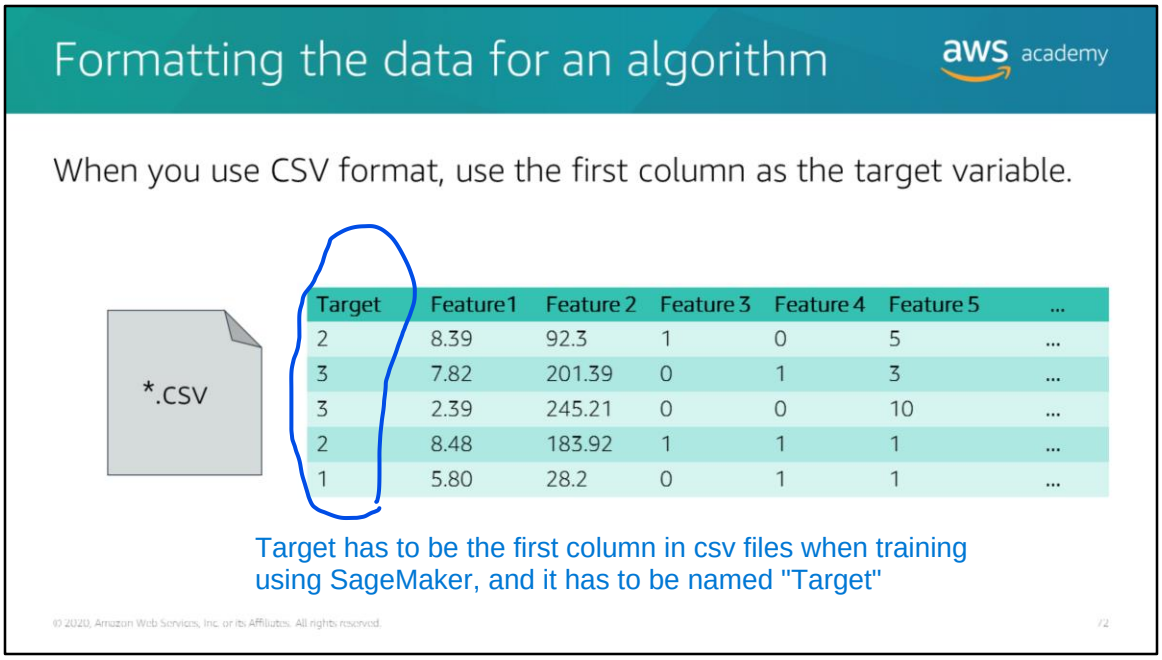
You have done much to clean and prepare your data, but your data still might not be ready to train the algorithm. Some algorithms require your data to be in a specific format. You should be aware of the following formatting steps as a way to prepare for model training.

Some algorithms might not be able to work with training data in a DataFrame format. In general, the data that you use for ML training can be stored in various formats, depending on your use case and algorithm. Various algorithms commonly use some file formats, such as CSV. Many Amazon SageMaker algorithms support training with data in CSV format. For use with Amazon SageMaker, CSV files can't have a header record, and the target variable must be in the first column.

Most Amazon SageMaker algorithms work best when you use the optimized protobuf recordIO format for the training data. In the protobuf recordIO format, Amazon SageMaker converts each instance in the dataset into a binary representation as a set of 4-byte floats, then loads it in the protobuf values field. This format enables you to take advantage of Pipe mode when you train the algorithms that support it. In Pipe mode, your training job streams data directly from Amazon S3. In contrast, File mode loads all your data from Amazon S3 to the training instance volumes. Streaming with Pipe mode can provide faster start times for training jobs and better throughput. You can also reduce the size of the Amazon Elastic Block Store (Amazon EBS) volumes for your training instances because pipe mode needs only

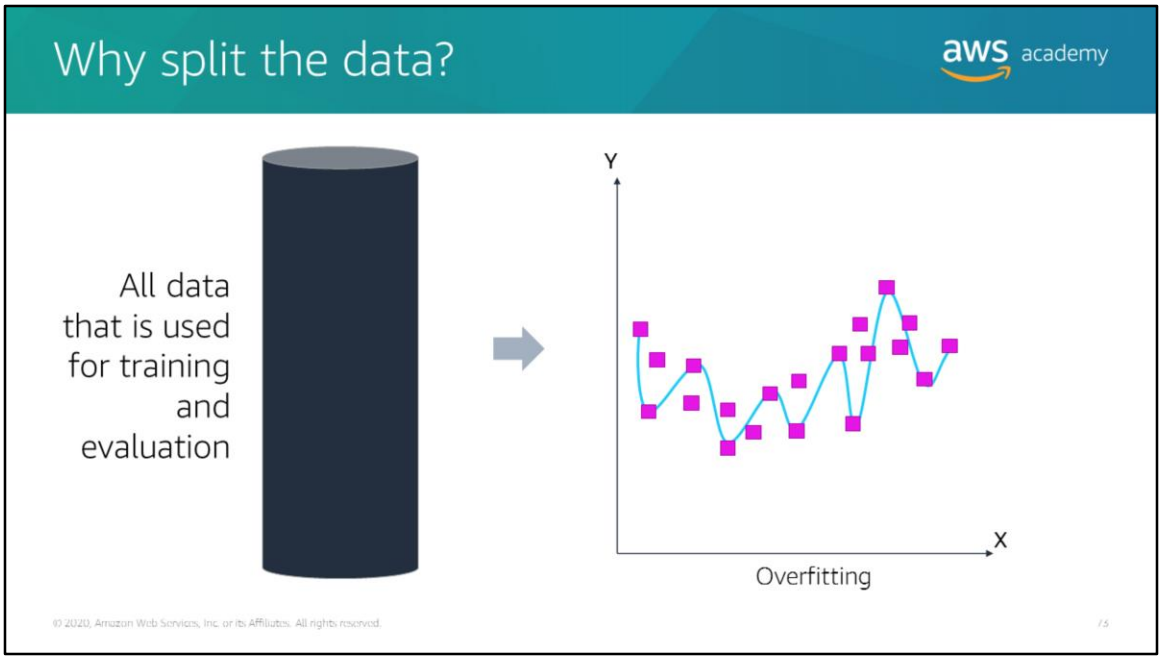
enough disk space to store your final model artifacts. In contrast, File mode needs disk space to store both your final model artifacts and your full training dataset.

The target variable in your training dataset should be the first column on the left. Your features should be to the right of the target variable column.

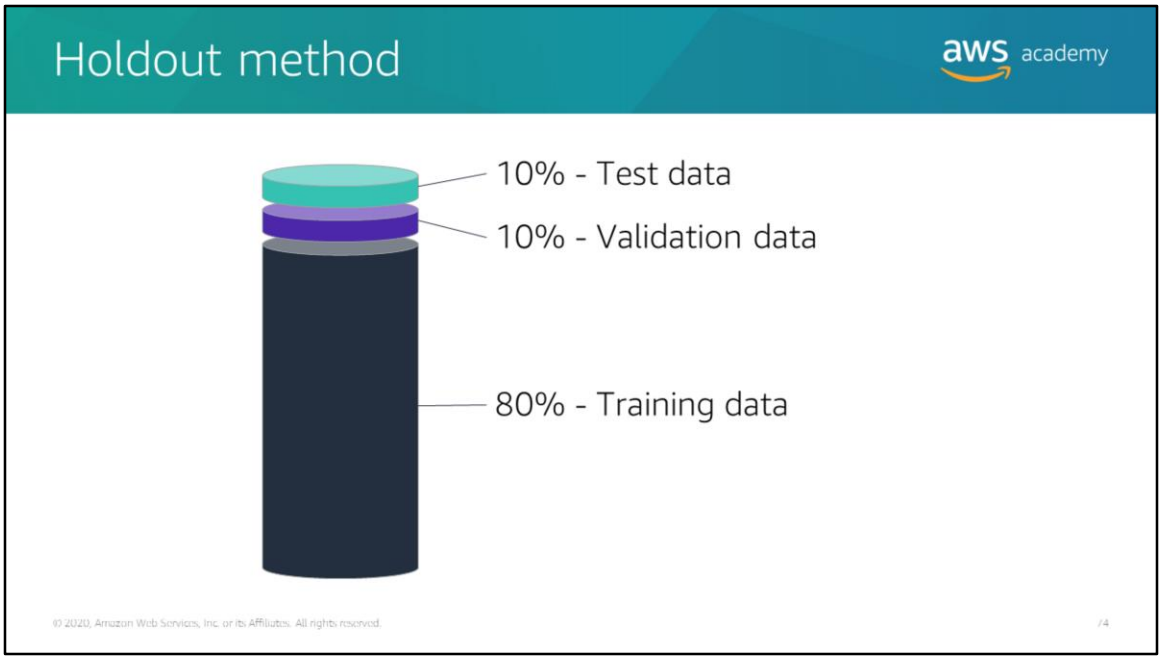


Target	Feature1	Feature 2	Feature 3	Feature 4	Feature 5	...
2	8.39	92.3	1	0	5	...
3	7.82	201.39	0	1	3	...
3	2.39	245.21	0	0	10	...
2	8.48	183.92	1	1	1	...
1	5.80	28.2	0	1	1	...

The target variable in your training dataset should be the first column on the left. Your features should be to the right of the target variable column.



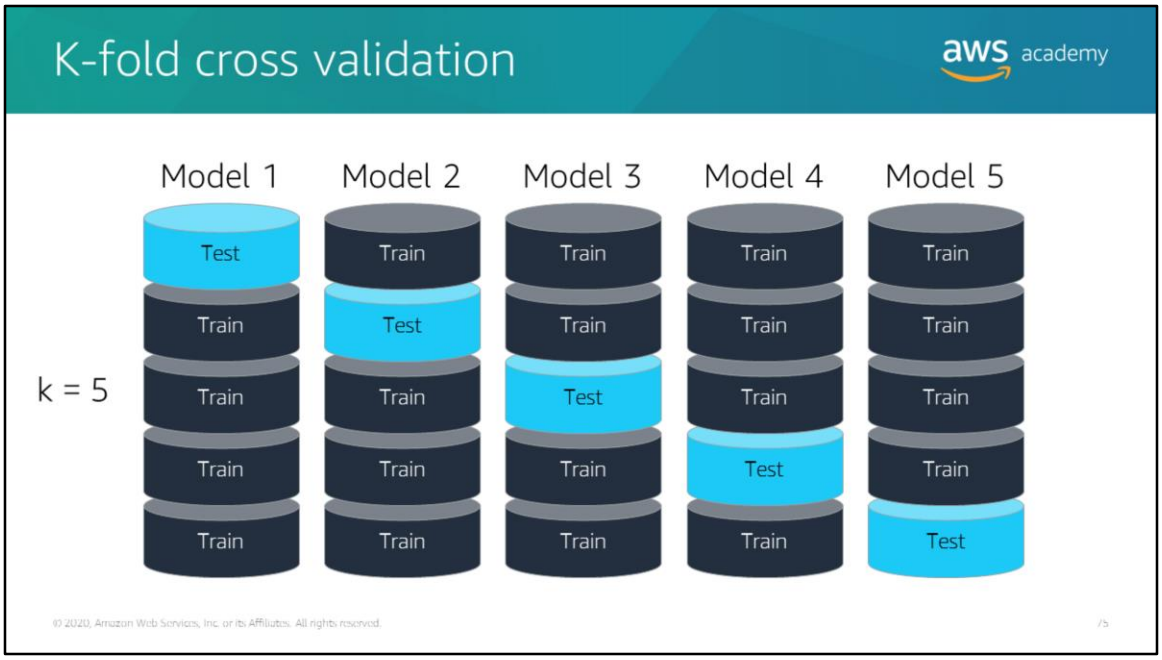
Evaluating a model with the same data that it trained on leads to *overfitting*. Recall that overfitting is where your model learns the particulars of a dataset too well. It essentially memorizes the training data, instead of learning the relationships between features and labels. Thus, the model can't use those relationships and patterns to apply to new data in the future.



Hold-out is when you split your data into multiple sets. These sets are usually training data, validation data, and testing data.

Training data, which includes both features and labels, feeds into the algorithm that you selected to produce your model. The model is then used to make predictions over the validation dataset. The *validation dataset* is where you might notice things that you will want to tweak, tune, and change. Then, when you are ready, you run the *test dataset*—which includes only features, because you want the labels to be predicted. The performance you get here with the test dataset is what you can reasonably expect to see in production.

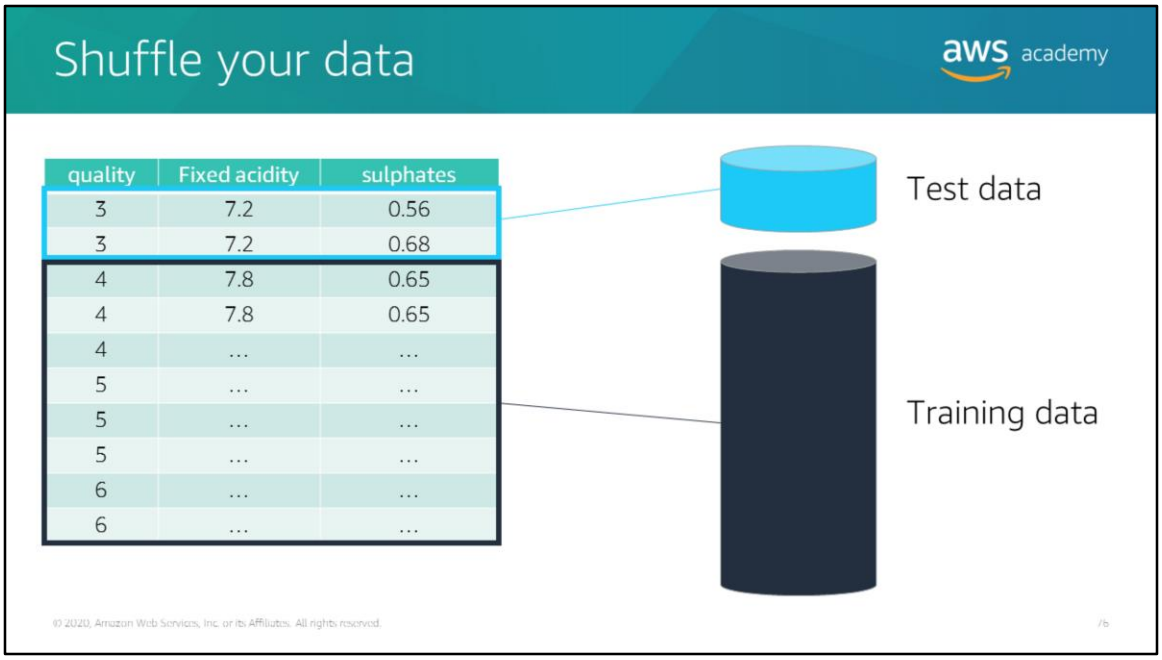
A common split when you use the hold-out method is as follows: use 80 percent of the data for a training set, 10 percent for validation, and 10 percent for test. Or, if you have a large amount of data, you can split it into 70 percent training, 15 percent validation, and 15 percent test.



For a small dataset, you can use k-fold cross validation to use as much data as possible. This technique provides good metrics to choose which model is better. K-fold cross validation randomly partitions the data into K different segments. For each segment, you use the rest of the data outside the segment for training to do a validation on that particular segment.

You do this process for each K segment. You apply different models to different pieces of the validation dataset to have some idea about how well the models perform. As a result, you use all the data for training. The cross-validation results are averaged to give you an idea about the performance of the model. You can use k-fold cross validation to apply different ML models to the training data. Thus, you can compare the performance of the models, which is based on the averaged result.

Consider an example where you have a 5-Fold Cross Validation. The available training data is separated into five different chunks. When you train the first model, you use all those chunks as the training data, and then calculate the metrics on this test piece. For the second model, you will use these pieces as training. After the model is trained, you apply it to this test piece. You repeat this process five times. You use all the training data, and you test it on five different models on different chunks of the test data. Eventually, you test it on all data points.



You should also be aware of one other potential problem with splitting your data. Data that's in a specific order can lead to biases in your models, and this is especially true when you work with structured data. Your data might be in a specific order, for example, by the *quality* column. If so, when you run your model against your test data, this ordered pattern is applied, which biases the model. It might also mean that some targets are missing from the training data.

Typically, randomizing your dataset before splitting is sufficient, and many libraries provide functions for randomization. With smaller sets, it is sometimes useful to use *stratified sampling*. Stratified sampling ensures that the training and test sets have approximately the same percentage of samples of each target class as the complete set.

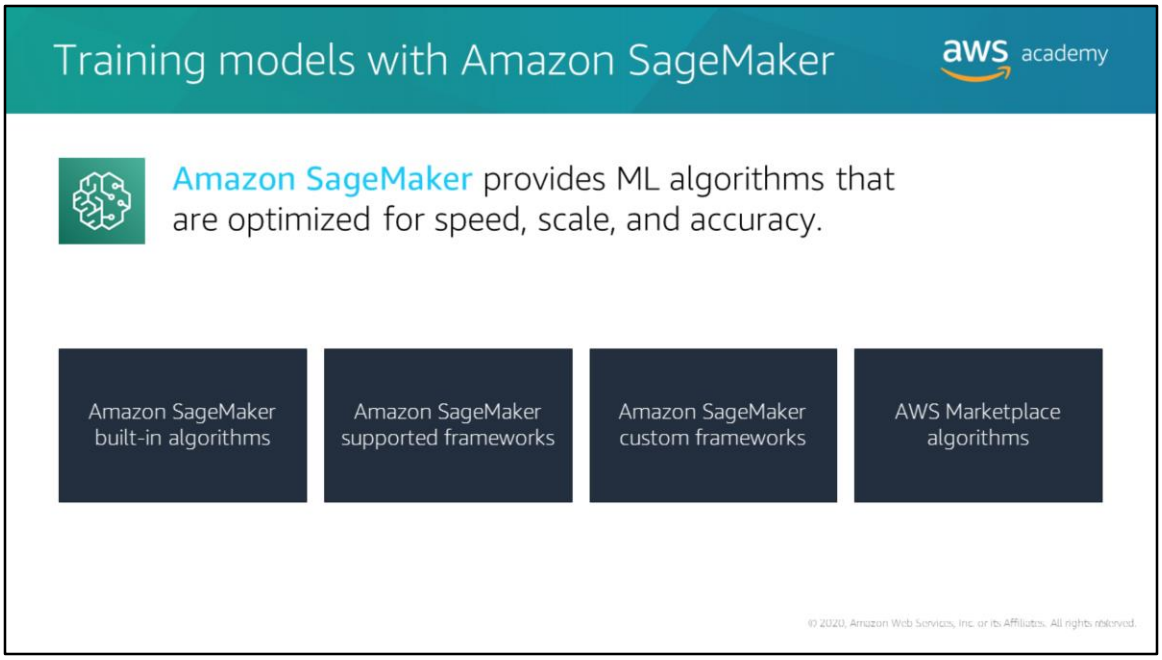
An internet search provides many ways to shuffle and split the data. One of the easiest ways is to use the train_test_split function from sklearn.

```
from sklearn.model_selection import train_test_split
training_data, test_data =
train_test_split(all_the_data, test_size=0.2, random_state=42, stratify=all_the_data[target'])
```

Note: The use of *stratify* ensures that the data—when it's split—maintains the ratio in the

In The Target Column.

target column.

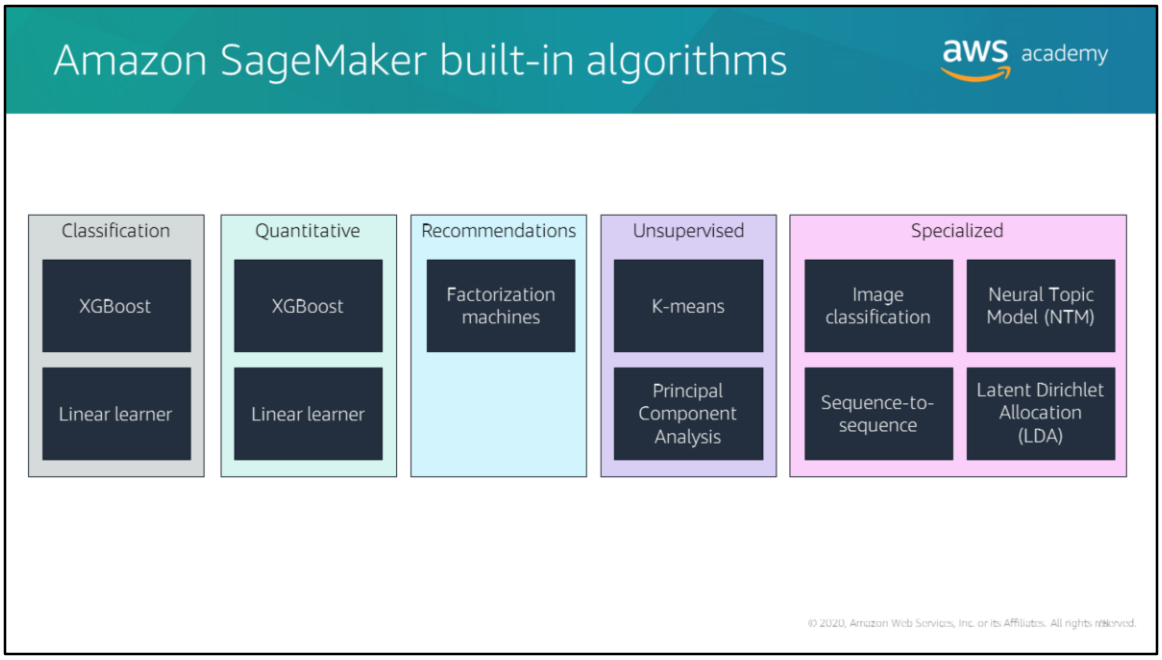


Amazon SageMaker provides four different ways that you can train models.

- Amazon SageMaker built-in algorithms – You can train and deploy these algorithms from Amazon SageMaker console, AWS Command Line Interface (AWS CLI), a Python notebook, or the Amazon SageMaker Python SDK. The available built-in algorithms are itemized and described in the next slide. Containers are used in the background when you use one of the Amazon SageMaker built-in algorithms, but you do not deal with them directly.
- Amazon SageMaker supported frameworks – Amazon SageMaker provides prebuilt containers to support deep learning frameworks such as Apache MXNet, TensorFlow, PyTorch, and Chainer. It also supports ML libraries, such as scikit-learn and SparkML by providing prebuilt Docker images. If you use the Amazon SageMaker Python SDK, they are deployed by using their respective Amazon SageMaker SDK Estimator class. In this case, you supply the Python code that implements your algorithm, and configure the prebuilt image to access your code as an entry point. You might have functional requirements for an algorithm or model that you developed in a framework, which a prebuilt Amazon SageMaker Docker image doesn't support. If so, you can modify an Amazon SageMaker image to satisfy your needs.
- Amazon SageMaker custom frameworks – If you have no prebuilt Amazon SageMaker container image to use or modify for an advanced scenario, you can package your own script or algorithm. You can then use this script or algorithm with Amazon SageMaker. You

can use any programming language or framework to develop your container. For example, your team might work and build ML models in R. If so, you can build your own containers to train and host an algorithm in R.

- AWS Marketplace algorithms – A third-party organization might have developed and tuned a model. It's worth looking in the AWS Marketplace to see what ready-to-use algorithms and models are available.



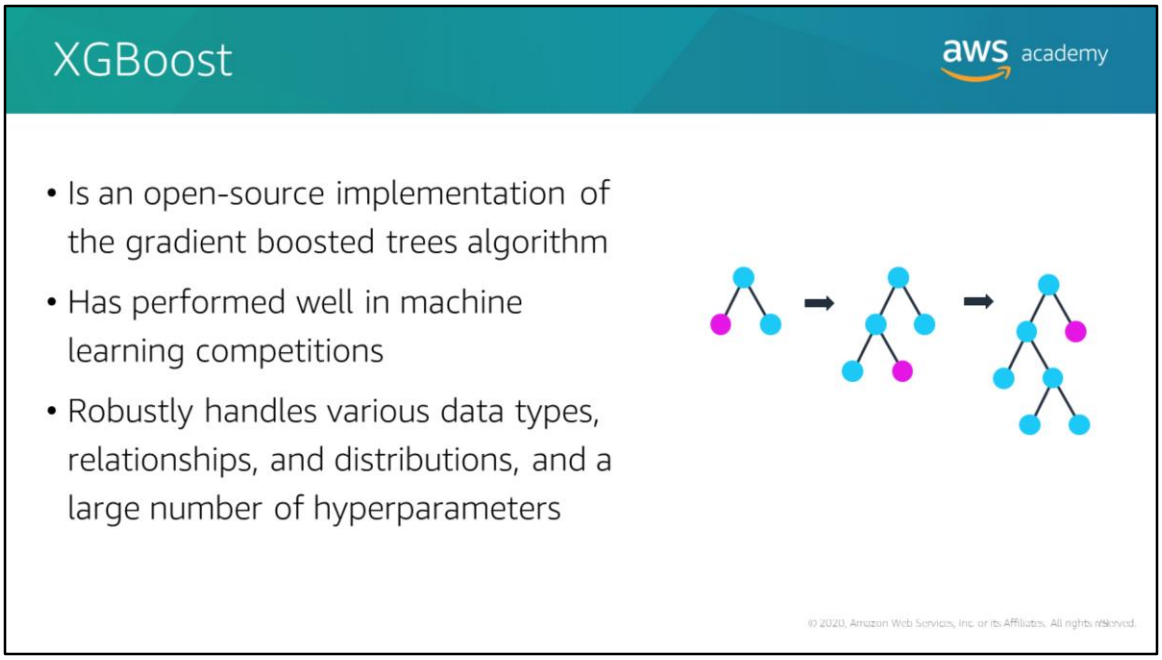
Amazon SageMaker provides high-performance, scalable ML algorithms that are optimized for speed, scale, and accuracy.

For supervised learning, Amazon SageMaker includes eXtreme Gradient Boosting (XGBoost) and linear learner algorithms for classification and quantitative (regression) problems. It also has a factorization machine to address recommendation and time-series prediction problems.

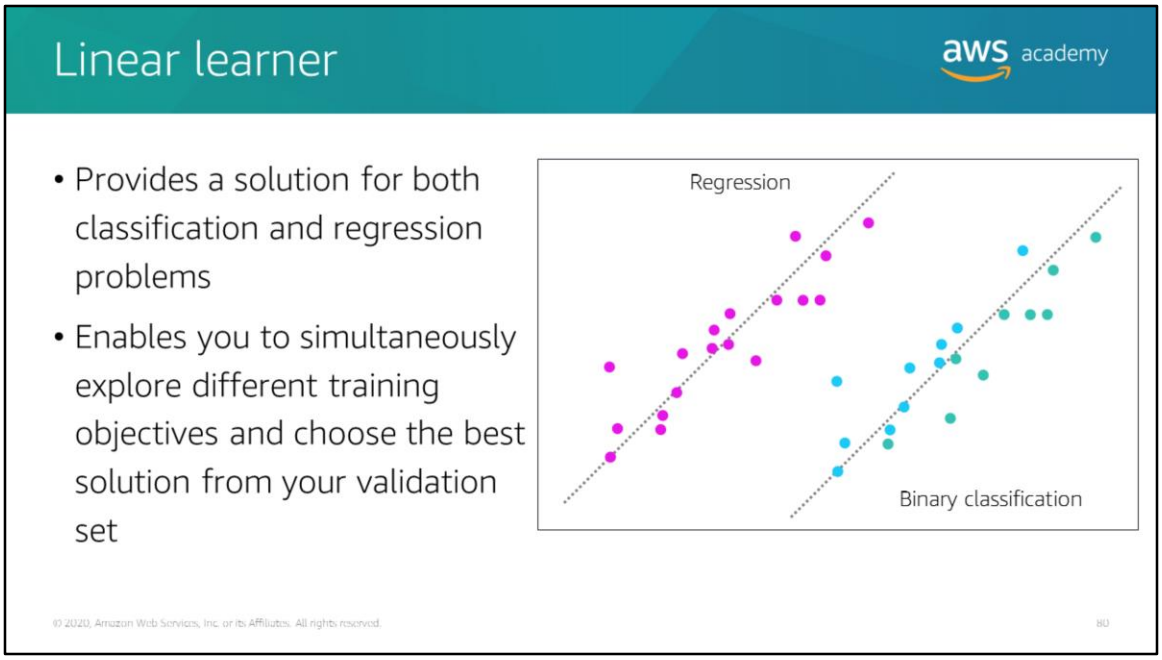
Amazon SageMaker also includes support for unsupervised learning, such as k-means clustering and principal component analysis (PCA). This learning is used to solve problems like identifying customer groupings that are based on purchasing behavior.

Finally, it has a selection of specialized algorithms for processing images and other deep learning tasks.

Next, you will look more closely at three of the most commonly used built-in algorithms and their use cases.



XGBoost is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm that attempts to accurately predict a target variable. It attains its prediction by combining an ensemble of estimates from a set of simpler, weaker models. XGBoost has done well in machine learning competitions. It robustly handles various data types, relationships, and distributions, and the many hyperparameters that can be tweaked and tuned for improved fit. This flexibility makes XGBoost a solid choice for problems in regression, classification (binary and multiclass), and ranking.



The Amazon SageMaker linear learner algorithm provides a solution for both classification and regression problems. With the Amazon SageMaker algorithm, you can simultaneously explore different training objectives and choose the best solution from your validation set. You can also explore many models and choose the best one for your needs.

The Amazon SageMaker linear learner algorithm compares favorably with methods that provide a solution for only continuous objectives. It provides a significant increase in speed over naive hyperparameter optimization techniques.

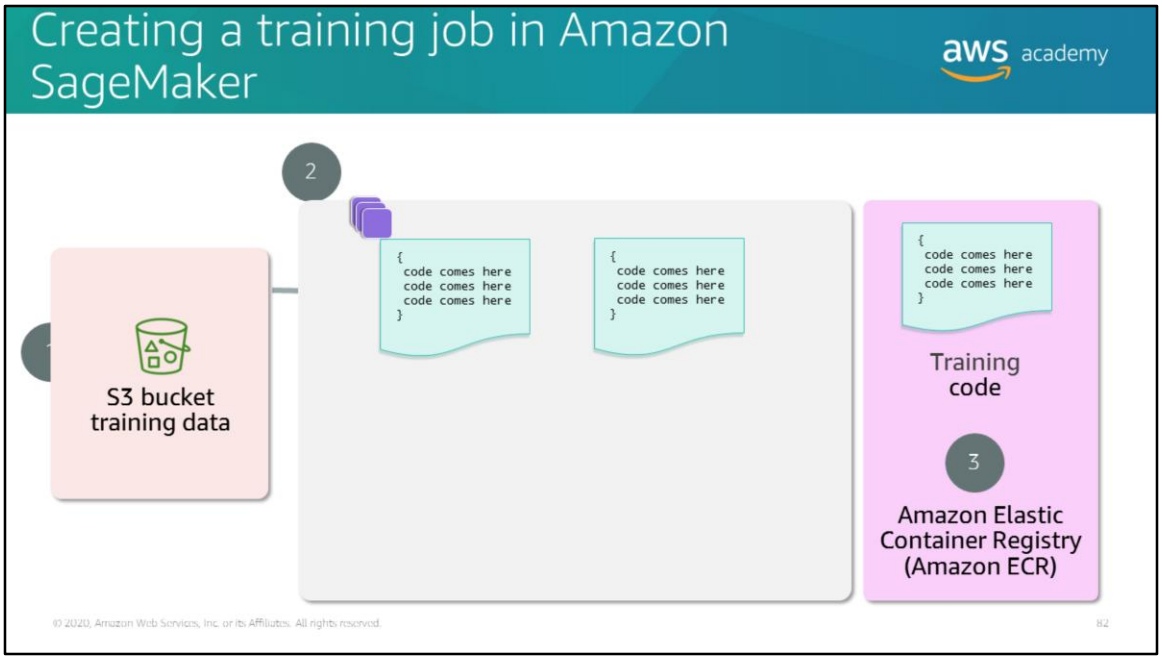
K-means

- It attempts to find discrete groupings within data, where members of a group are as similar as possible.
- The *means* in *k-means* is the averaging of the data. This averaging helps to find the center of the grouping.

K-means is an unsupervised learning algorithm. It attempts to find discrete groupings within data. In such groupings, members of a group are as similar as possible to one another, and as different as possible from members of other groups. You define the attributes that you want the algorithm to use to determine similarity.

K means plus plus is a model that tells you what percentage the point could be for each cluster and this is mainly for when a point is close enough to be in more than 1 cluster.

Lasso and Ridge and regularization techniques for when we want to imrove our model's performance if overfitting.

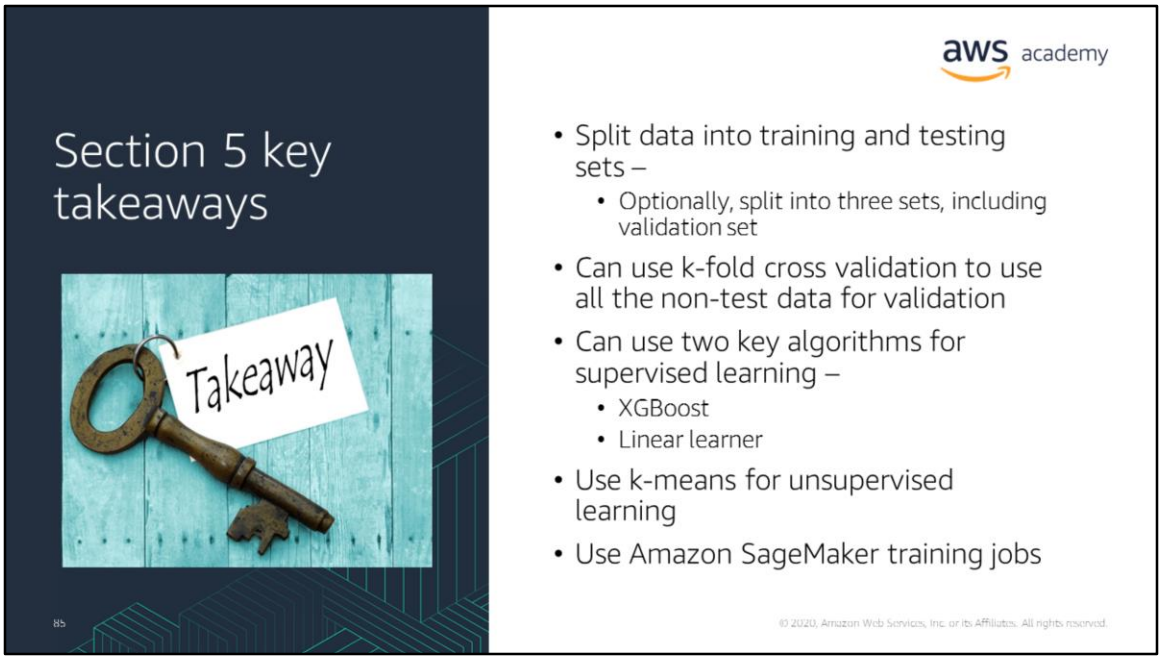


To train a model in Amazon SageMaker, you create a *training job*. The training job includes the following information:

- The URL of the Amazon S3 bucket where you stored the training data.
- The compute resources that you want Amazon SageMaker to use for model training. Compute resources are ML compute instances that Amazon SageMaker manages. Amazon SageMaker provides a selection of instance types that are optimized to fit different ML use cases. Instance types comprise varying combinations of CPU, GPU, memory, and networking capacity. You have the flexibility to choose the appropriate mix of resources for building, training, and deploying your ML models. Each instance type includes one or more instance sizes, and this feature enables you to scale your resources to the requirements of your target workload.
- The URL of the S3 bucket where you want to store the output of the job.
- The Amazon Elastic Container Registry (Amazon ECR) path where the training code is stored.

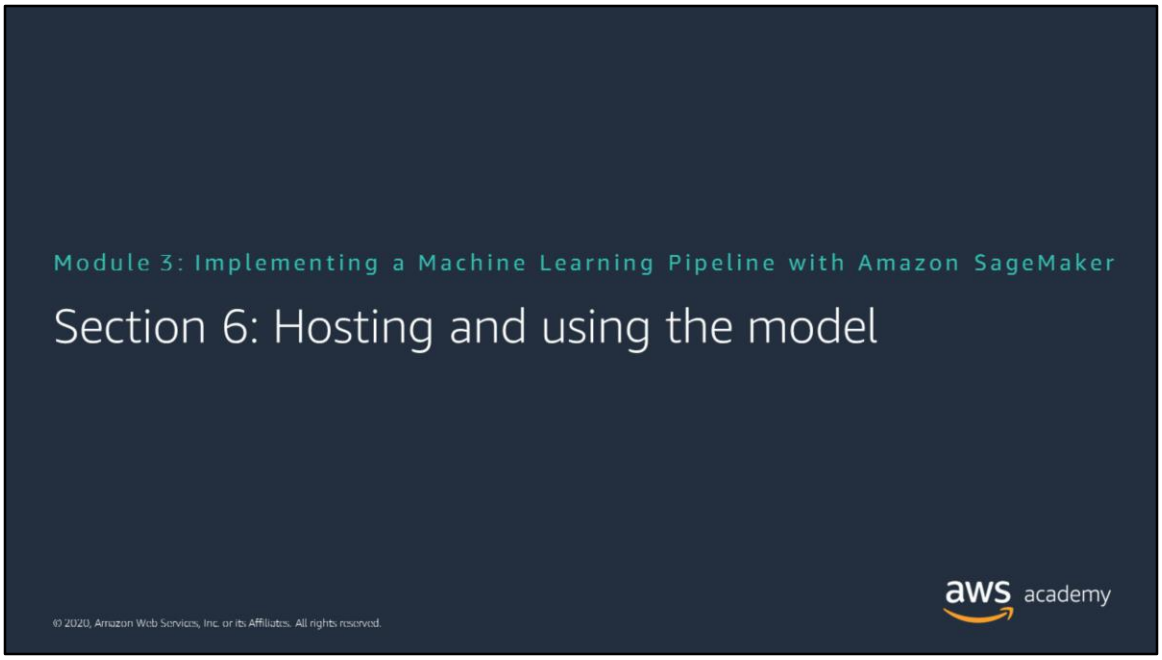
Your instructor will now either demonstrate how to train a model with Amazon SageMaker or provide you with access to a recorded demonstration.

You will now complete Lab 4: Splitting Data and Training a Model with XGBoost.

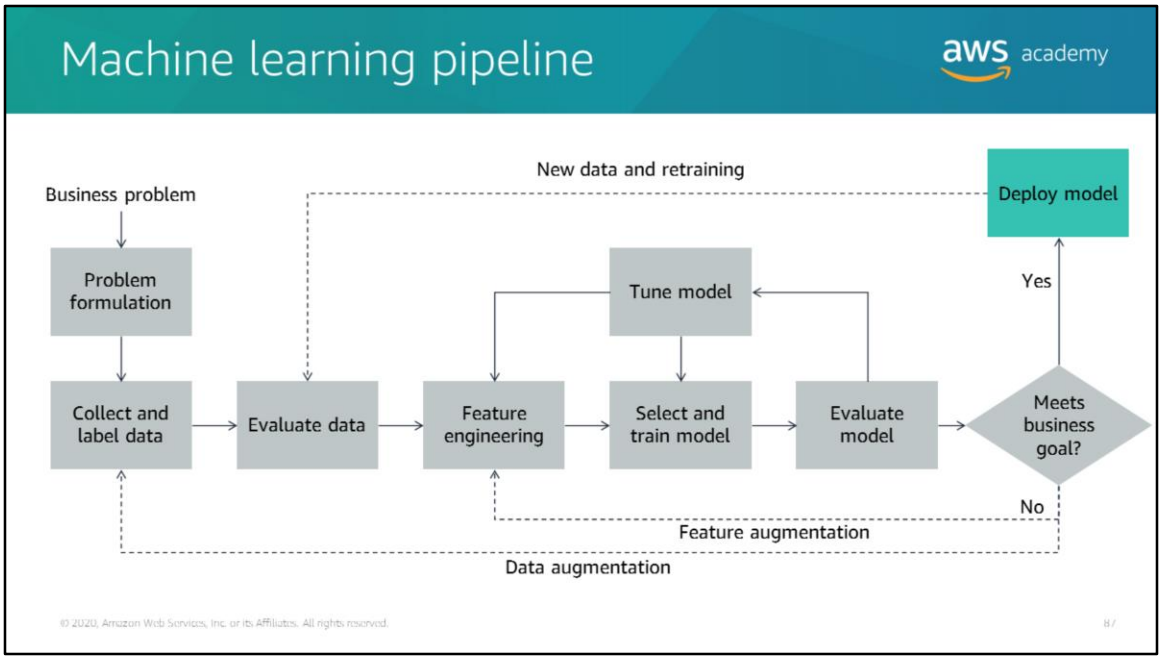


Some key takeaways from this section of the module include:

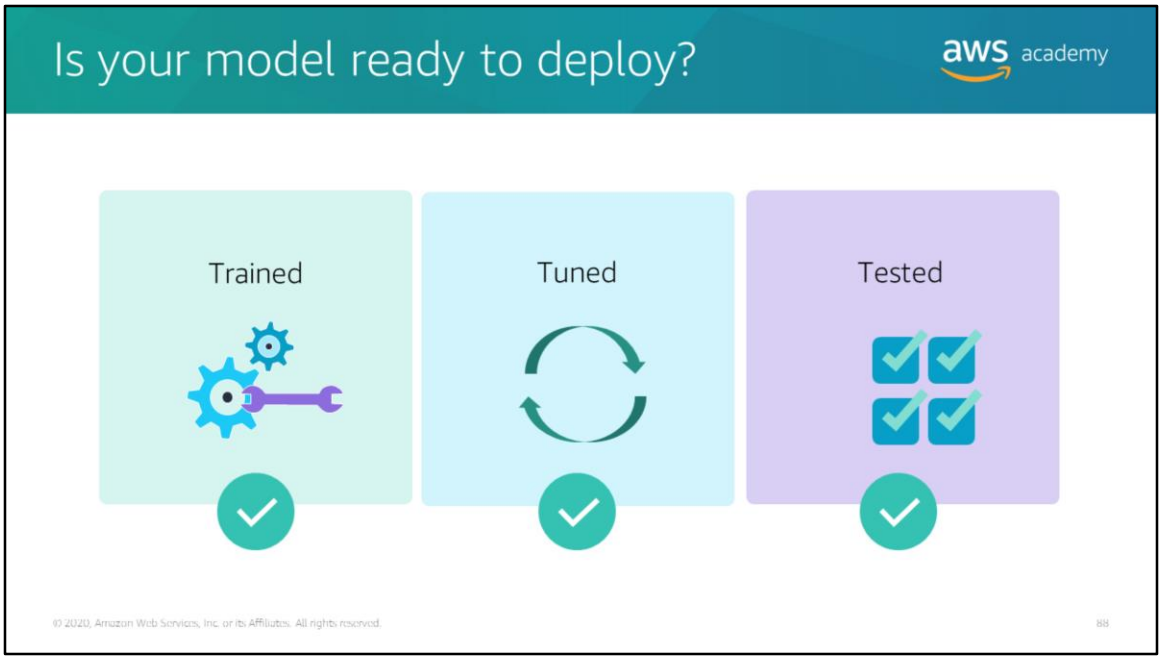
- Split data into training, testing and validation sets to help you validate the models accuracy
- Can use K-fold cross validation can help with smaller datasets
- Can use two key algorithms for supervised learning—XGBoost and linear learner
- Use k-means for unsupervised learning
- Use Amazon SageMaker training jobs to train models



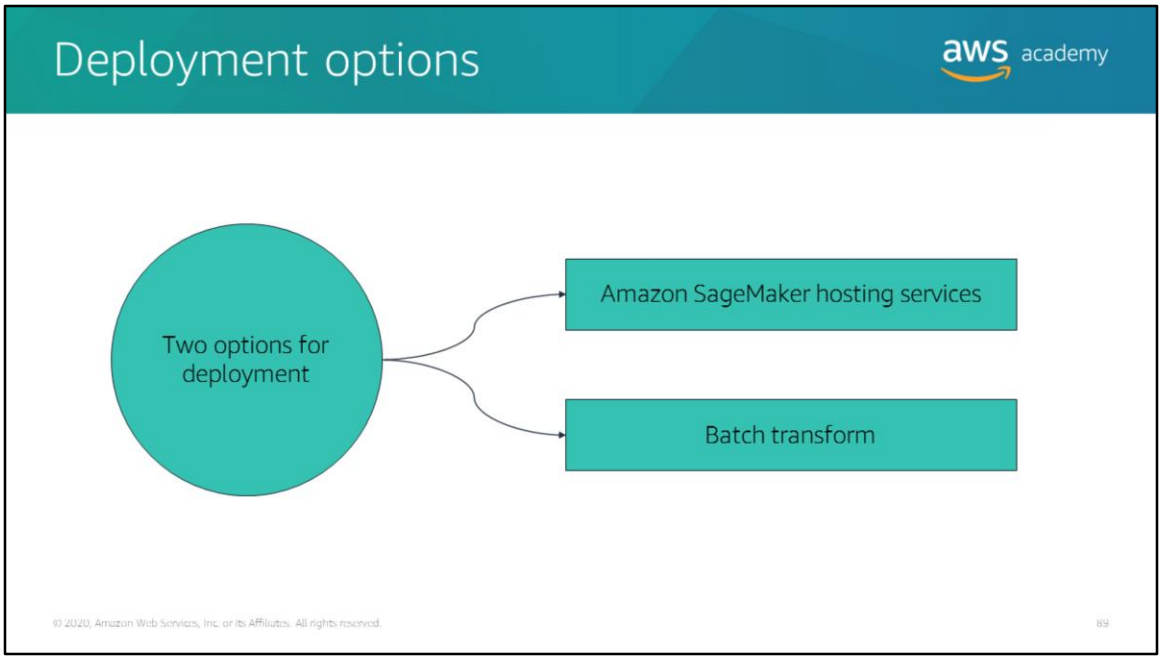
Introducing Section 6: Hosting and using the model.



In this section, you will learn how you can deploy your trained model for consumption by applications.



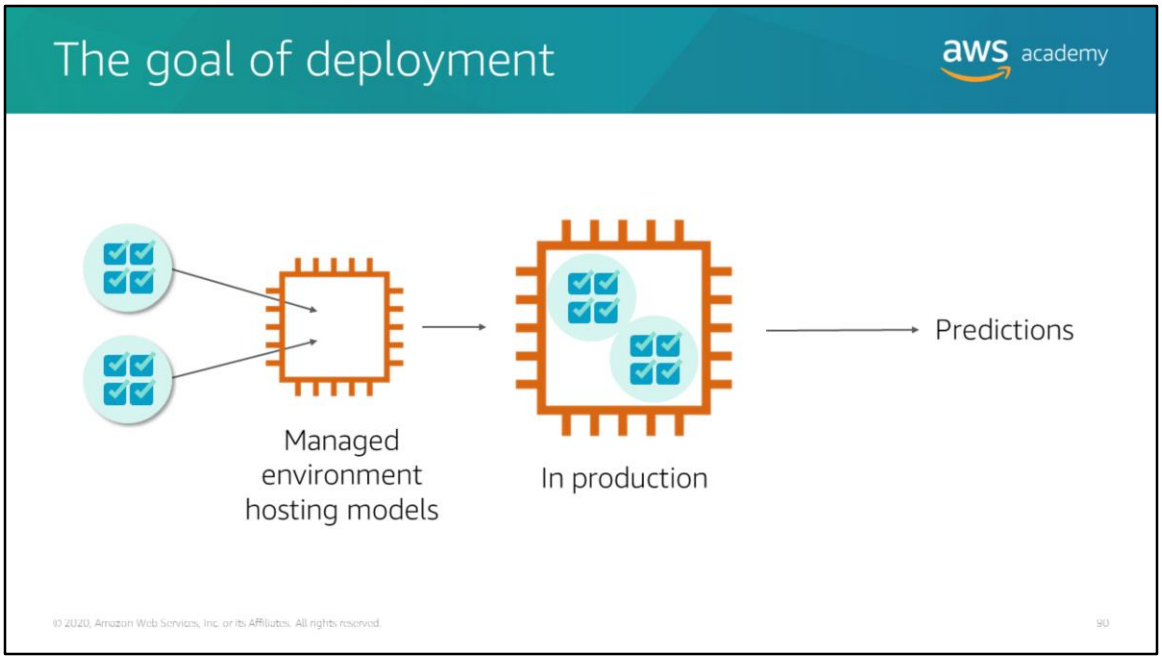
After you have trained, tuned, and tested your model, you are ready to deploy it. (Note: You will learn about tuning in the next section.) You might think that you are looking at the phases in the ML pipeline out of order. To test and get performance metrics from your model, you must make inferences or predictions from the model—which typically requires deployment. Deployment for testing is different from production, although the mechanics are the same. Amazon SageMaker provides everything necessary to host your model for simple testing and evaluation. These assessments can involve anything from a few requests to deployments that handle tens of thousands of requests.



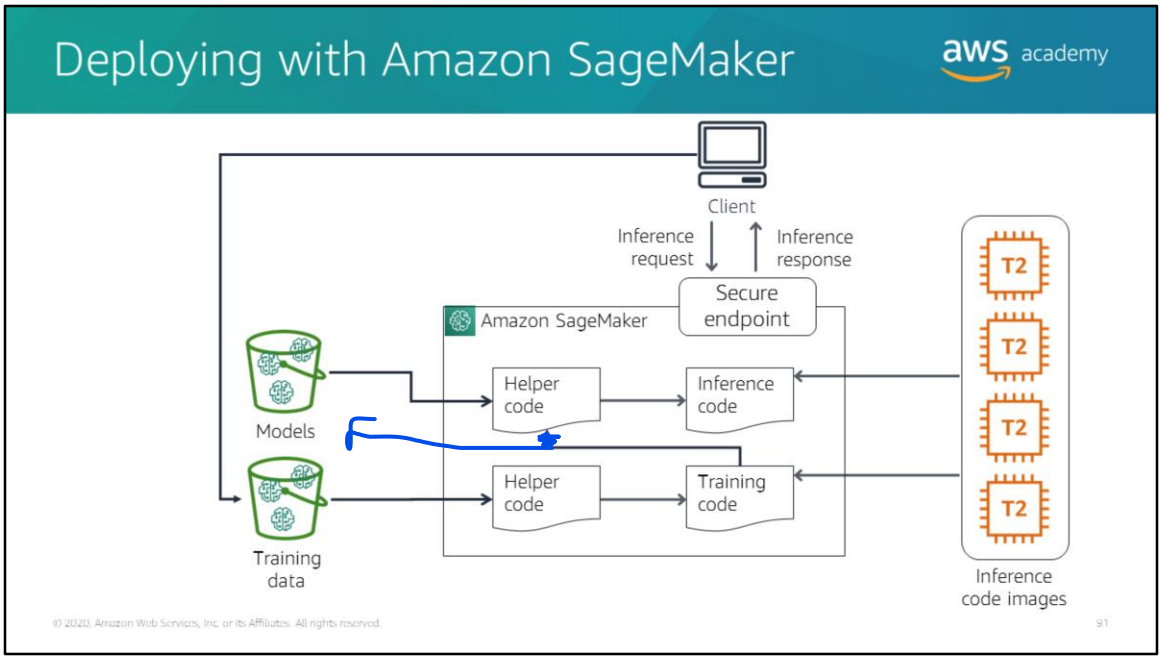
You can deploy your model in two ways.

For single predictions, deploy your model with Amazon SageMaker hosting services. Amazon SageMaker deploys multiple compute instances that run your model behind a load balanced endpoint. Applications can call the API at the endpoint to make predictions. With this model, you can scale the number of instances up or down, based on demand.

To get predictions for an entire dataset, use Amazon SageMaker batch transform. Instead of deploy and maintain a permanent endpoint, Amazon SageMaker spins up your model and performs the predictions for the entire dataset that you provide. It stores the results in Amazon S3 before it shuts down and terminates the commute instances. Performing batch predictions when you test the model is useful because you can quickly run your entire validation set against the model. You don’t need to write any code to process and collate the individual results.



The goal of the deployment phase is to provide a managed environment to host models for providing inference both securely and with low latency. After deployment into production, remember to monitor your production data. If necessary, you must retrain your model, because newly deployed models must reflect the current production data. New data is accumulated over time, and it can potentially identify alternative or new outcomes. Therefore, deploying a model is a continuous process, not a one-time exercise.



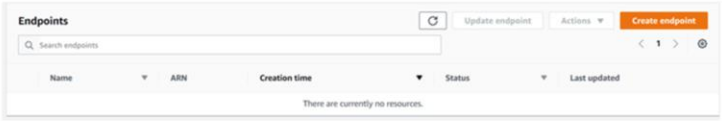
For high availability and redundancy, you can easily deploy your model to automatically scaling Amazon ML instances across multiple Availability Zones. Specify the type of instance, and the maximum and minimum number that you want—Amazon SageMaker takes care of the rest. It launches the instances, deploys your model, and sets up the secure HTTPS endpoint for your application. Your application must include an API call to this endpoint to achieve low latency and high-throughput inference. This architecture enables you to integrate your new models into your application in minutes because model changes no longer require changes to application code.

Amazon SageMaker manages your production compute infrastructure on your behalf. It performs health checks, applies security patches, and conducts other routine maintenance. It does all of these tasks with built-in Amazon CloudWatch monitoring and logging.

Creating an endpoint

aws academy

You can create an endpoint in the console:



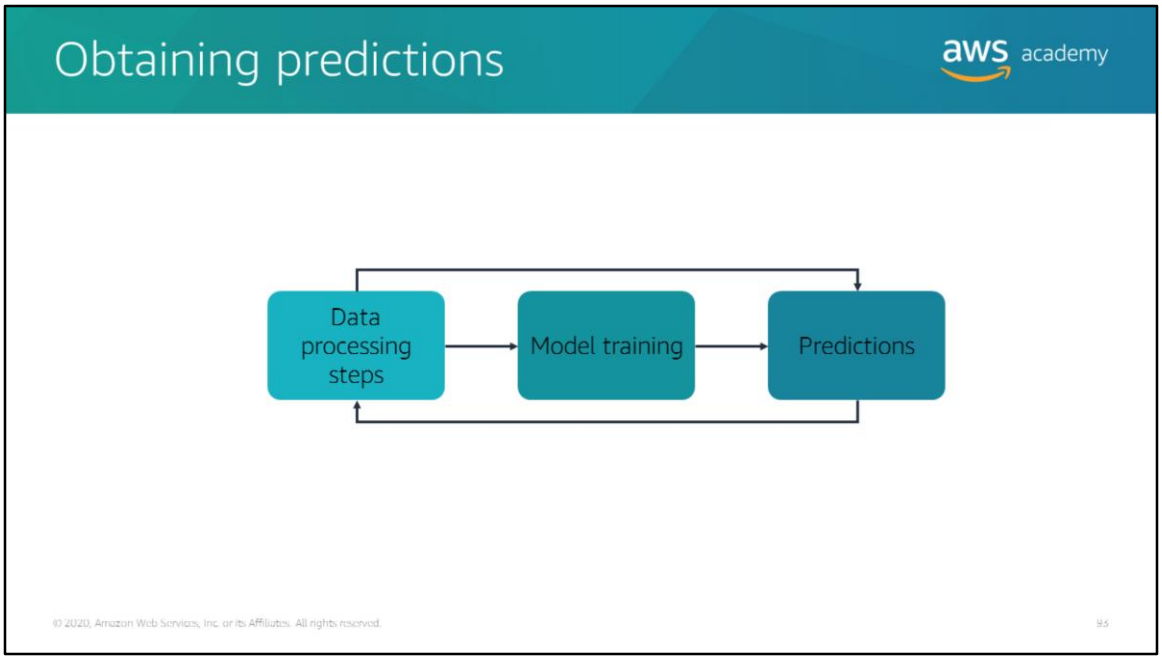
Or, you can use the Amazon SageMaker SDK:

```
xgb_predictor = xgb_model.deploy(initial_instance_count=1,
                                 content_type='text/csv',
                                 instance_type='ml.t2.medium'
                                 )
```

 92

After you train the model, you can create the endpoint either in code or by using the Amazon SageMaker console. If you plan to host only a single model, you can create an endpoint for that model. However, if you plan to host multiple models, you must create a multi-model endpoint.

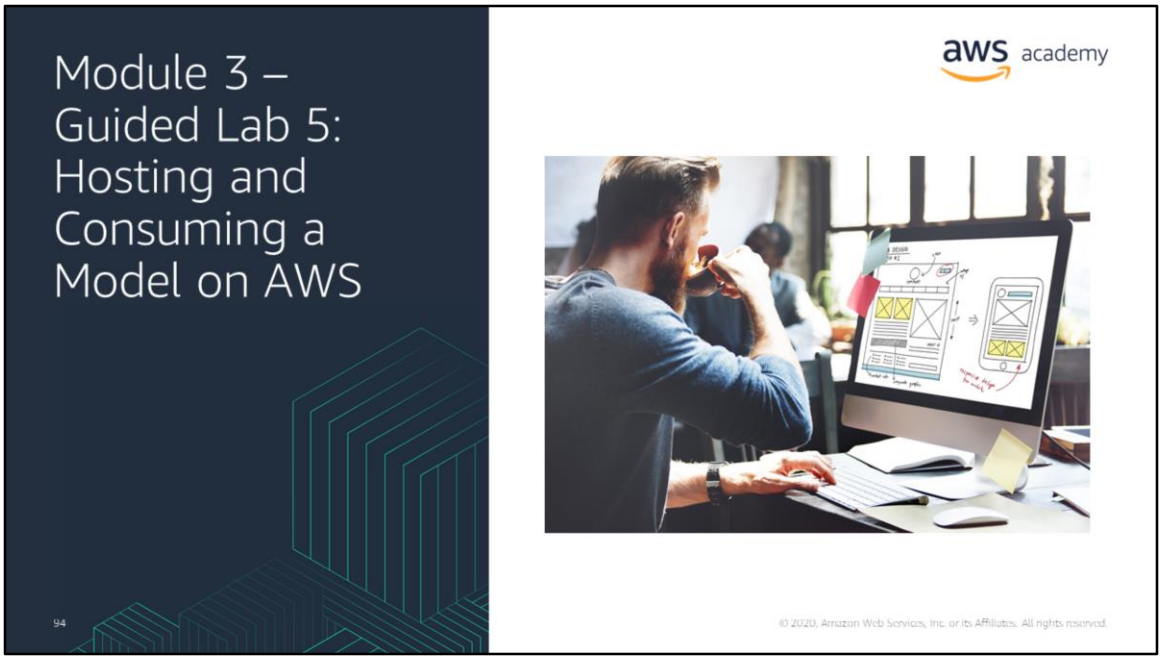
Multi-model endpoints provide a scalable and cost-effective solution for deploying large numbers of models. They use a shared serving container that is enabled to host multiple models. This process reduces hosting costs by improving endpoint utilization, compared to single-model endpoints. It also reduces deployment costs because Amazon SageMaker loads models in memory, and scales them according to traffic patterns.



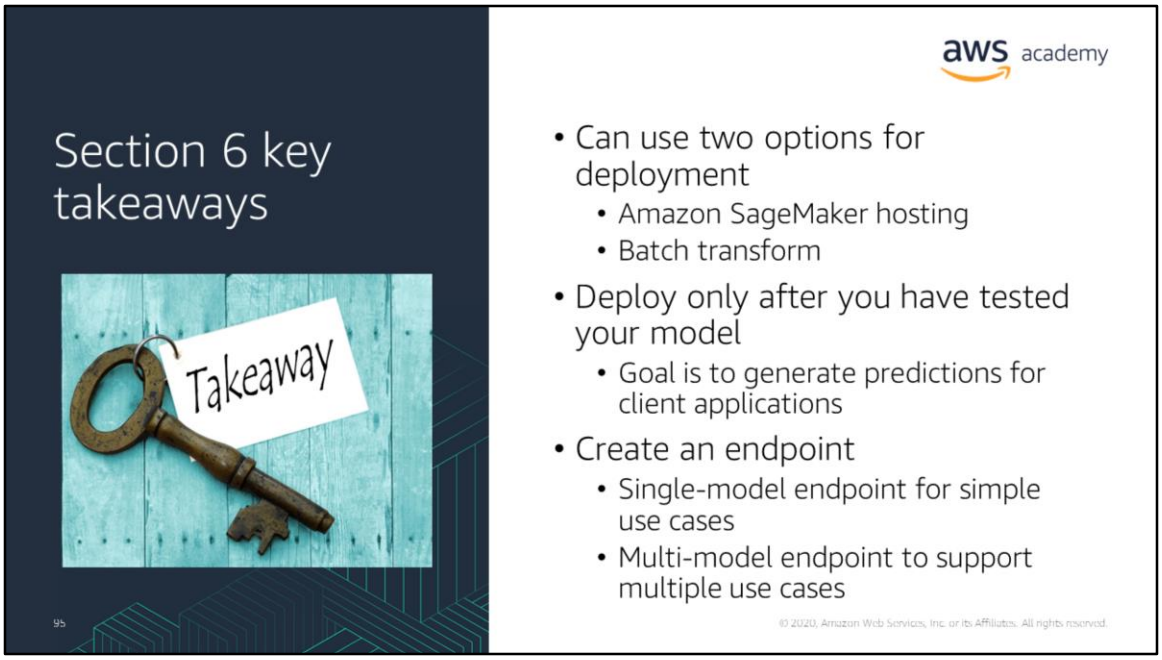
When you deploy machine learning models into production so you can make predictions on new data, consistency is important. You must ensure that the same data processing steps that were used in training are also applied to each inference request. Otherwise, you might get incorrect prediction results.

By using inference pipelines, you can reuse the data processing steps that were applied in model training to inference. It isn't necessary to maintain two separate copies of the same code. This reuse of code helps with the accuracy of your predictions and reduces development cost.

Because Amazon SageMaker is a managed service, inference pipelines are completely managed. When you deploy the pipeline model, the service installs and runs the sequence of containers on each EC2 instance in the endpoint, or in the batch transform job. Additionally, the sequence of feature processing and inferences runs with low latency because the containers are collocated on the same EC2 instances.

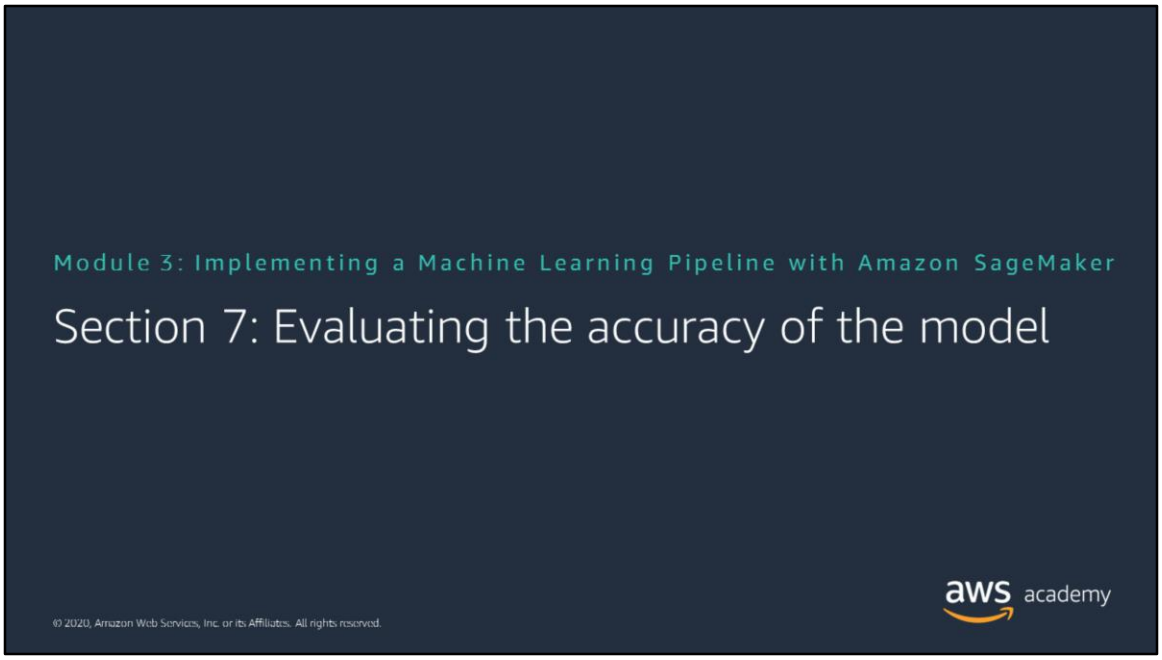


You will now complete Module 3 – Guided Lab 5: Hosting and Consuming a Model on AWS.

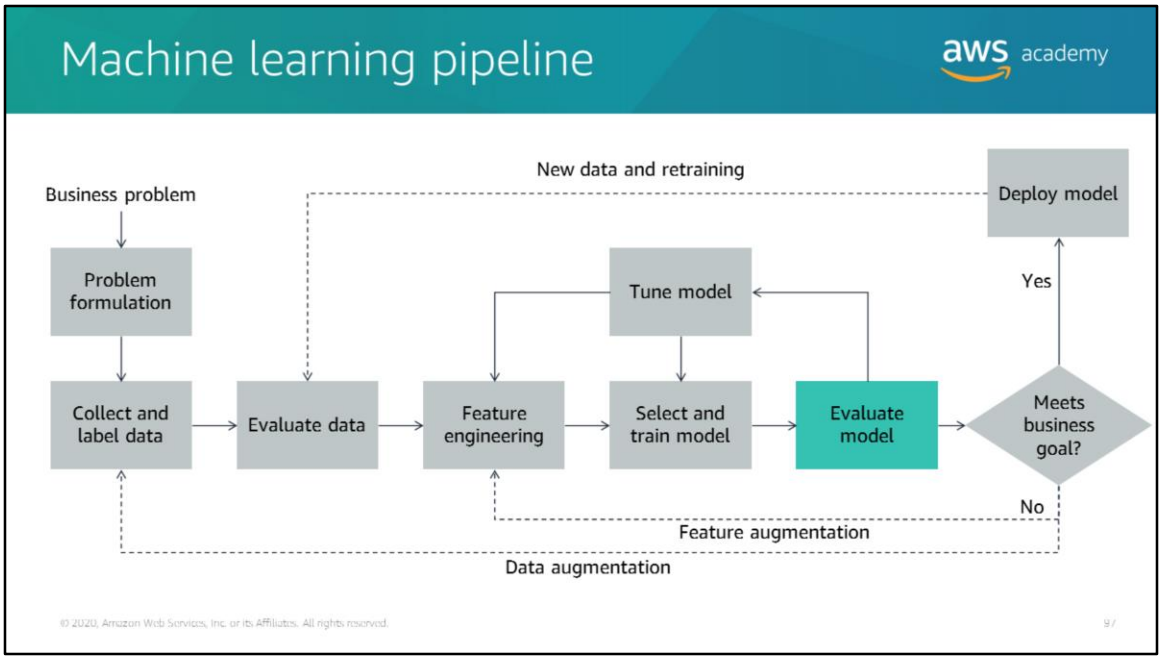


Some key takeaways from this section of the module include:

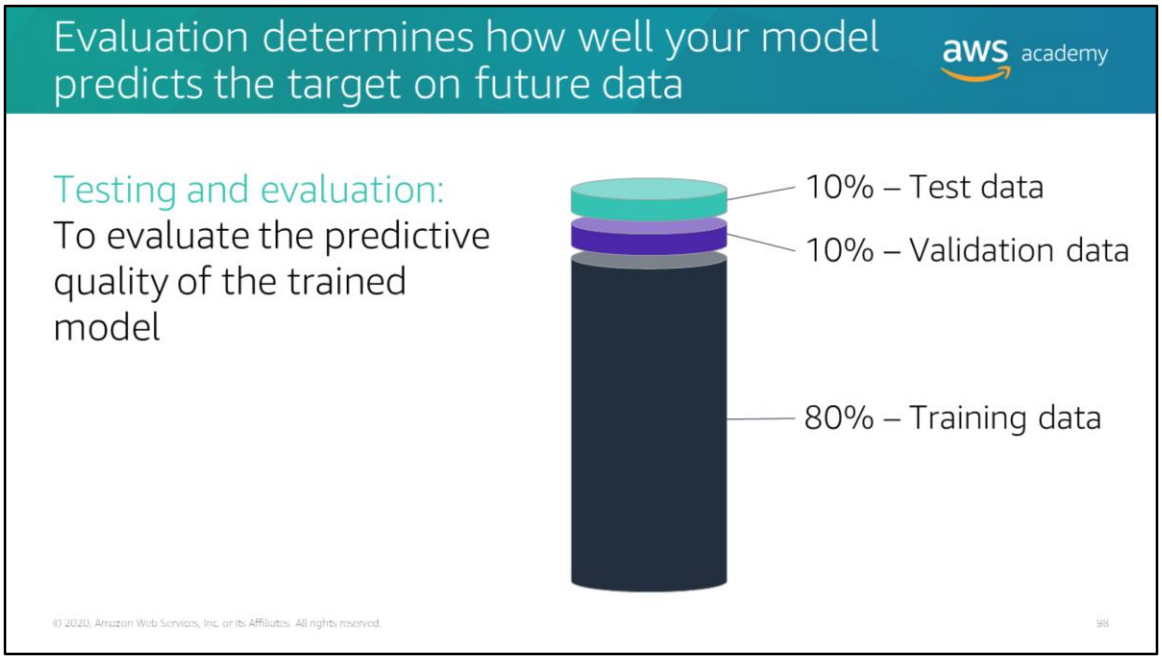
- You can deploy your trained model by using Amazon SageMaker to handle API calls from applications, or to perform predictions by using a batch transformation.
- The goal of your model is to generate predictions to answer the business problem. Be sure that your model can generate good results before you deploy it to production.
- Use Single-model endpoints for simple use cases and use multi-model endpoint support to save resources when you have multiple models to deploy.



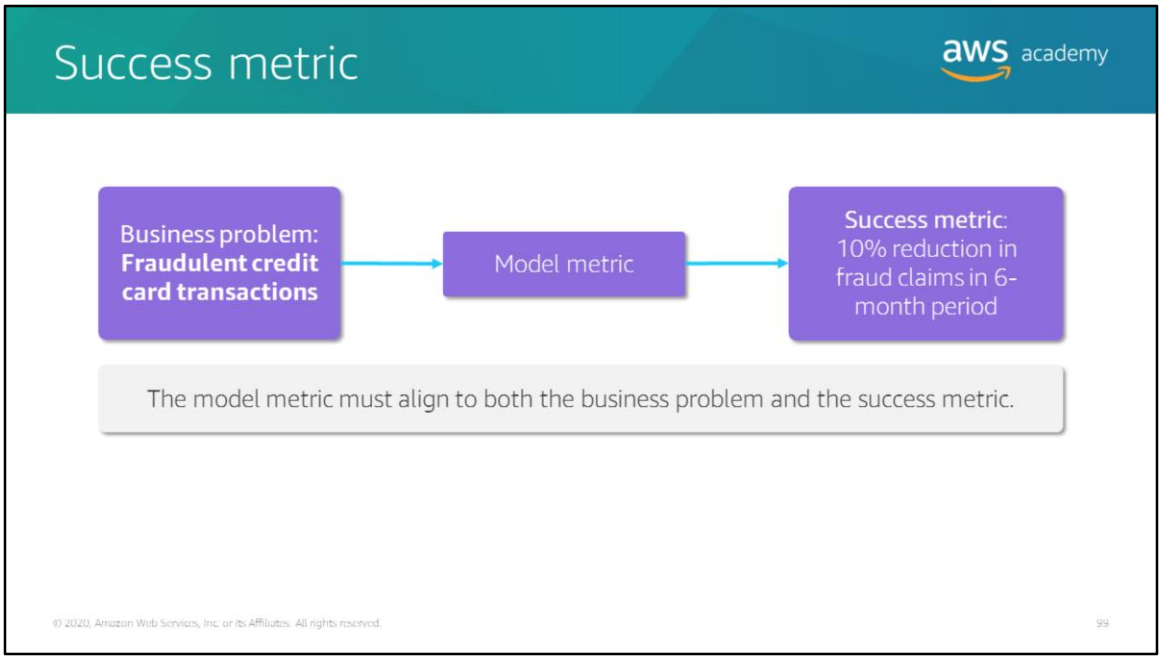
Introducing Section 7: Evaluating the accuracy of the model.



In this section, you will learn how you can evaluate your model's success in predicting results.

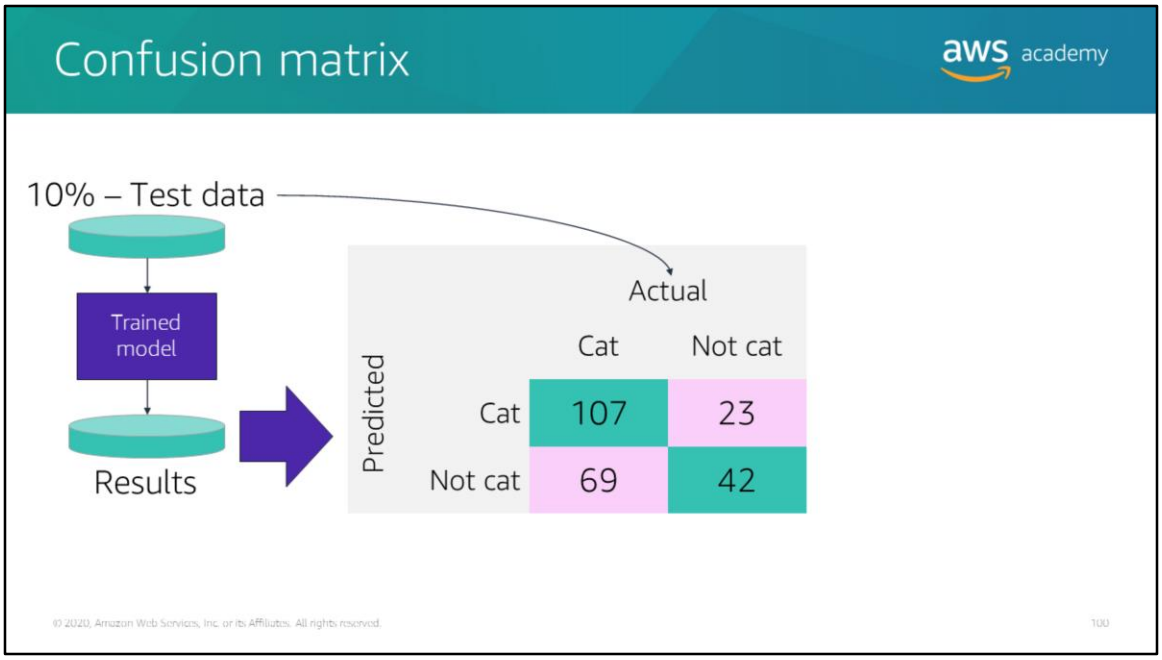


After you train and tune your models and decide which model is the best for your business problem, you must evaluate your model. You want to determine whether it will perform well in predicting the target on new and future data. Because future instances have unknown target values, you must assess how the model will perform on data for which you know the target answer. Then, use this assessment as a proxy for performance on future data. For this reason, you hold out a sample of your data for evaluation or testing purposes.



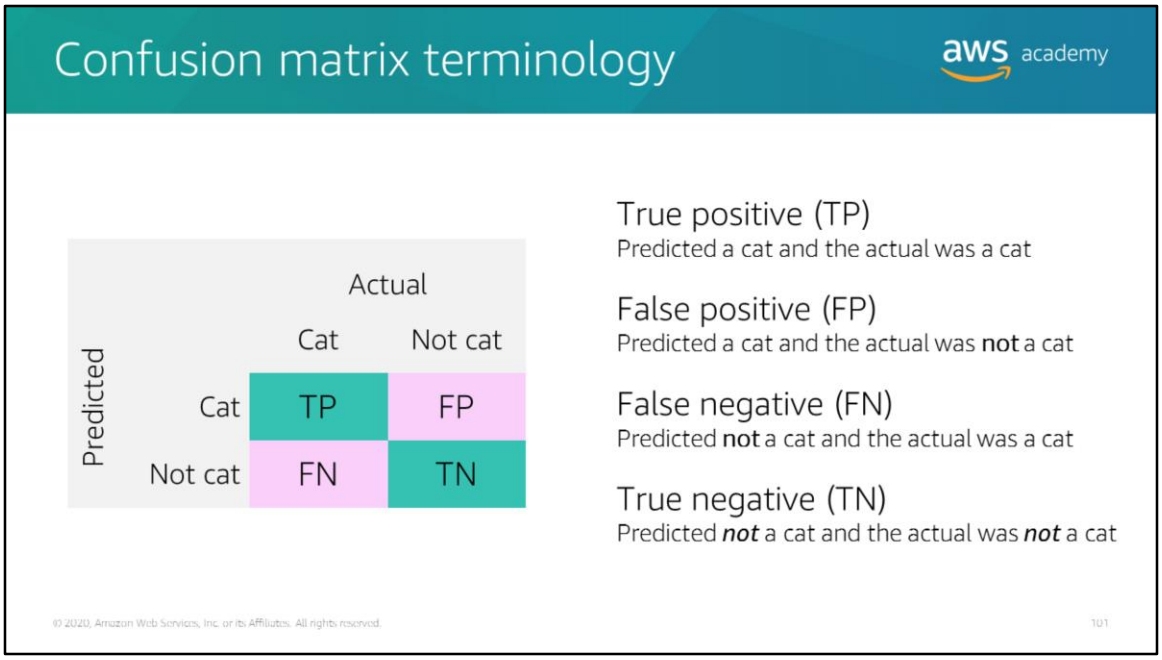
An important part of this phase is to choose the most appropriate metric for your business situation. Think back to the earlier section on problem formulation. During that phase, you defined your business problem and outcome, and you crafted a business metric to evaluate success. The model metric that you choose at this phase should be linked to that business metric as closely as possible. Frequently, the two metrics have a high correlation.

In addition to considering your business problem and success metric, the type of ML problem you work with influences the model metric that you choose. Throughout the rest of this module, you will see examples of common metrics that are used in classification problems and in regression problems. You will start by looking at a simple binary classification problem.



Now, you will consider a specific example. Suppose that you have a simple image recognition model that labels data as either *cat* or *not cat*. After the model is trained, you can use your test dataset that you held back to perform predictions. To help examine the performance of the model, you can compare the predicted values with the actual values.

If you plot the values into a table as shown, you can start to get some insights into how well the model performed.

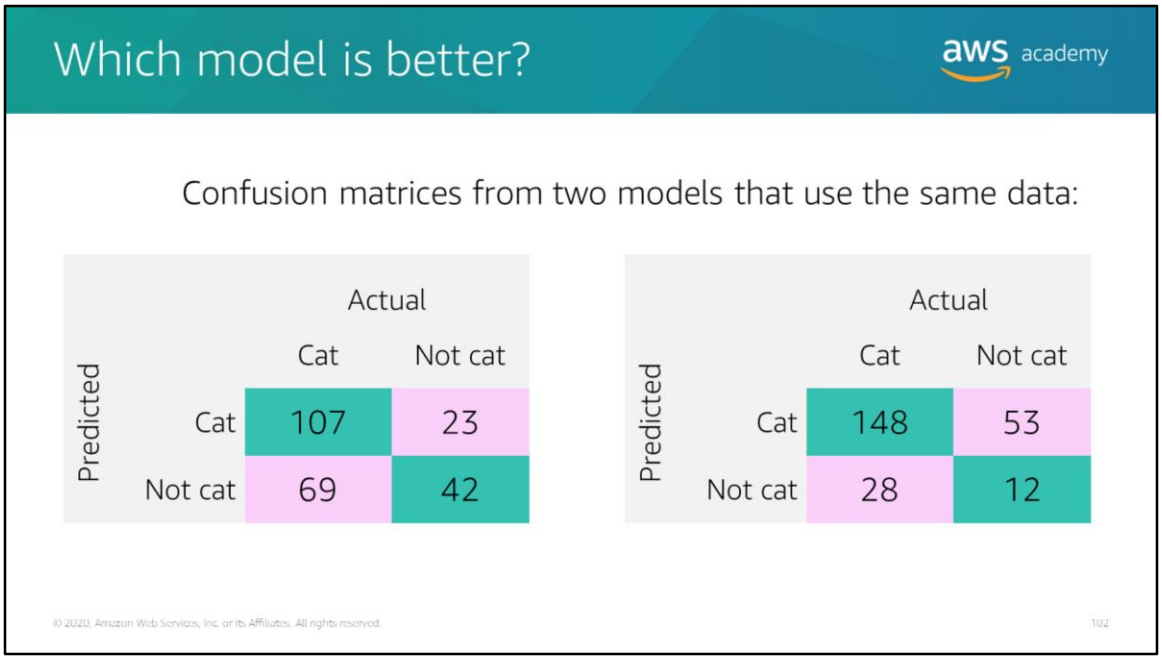


In a confusion matrix, you can get a high-level comparison of how the predicted classes matched up against the actual classes.

Suppose that the actual label or class is *cat*, which is identified as *P* for *positive* in the confusion matrix. And suppose that the predicted label or class is also *cat*. Then, you have a true positive result, which is a good outcome for your model.

Similarly, suppose that you have an actual label of *not cat*, which is identified as *N* for *negative* in the confusion matrix. And suppose that the predicted label or class is also *not cat*. Then, you have a true negative, which is also a good outcome for your model. In both of these cases, your model—by using the testing data—predicted the correct outcome.

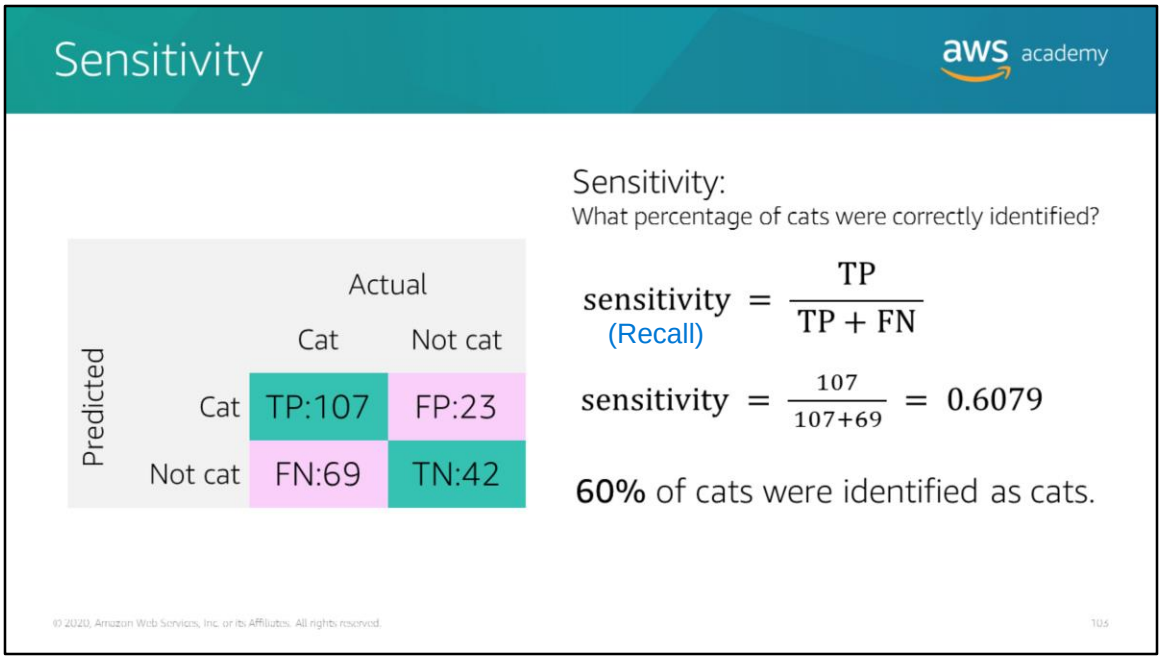
Two other outcomes are possible, both of which are less than ideal. The first outcome is when the actual class is negative, so *not cat*, but the predicted class is positive, so *cat*. This outcome is called a *false positive* because the prediction is positive, but incorrect. A *false negative* occurs when the actual class is positive, so *cat*, but the predicted class is negative, so *not cat*.



The diagram shows the confusion matrices from the performance of two different models on the same data. Can you tell which model is better?

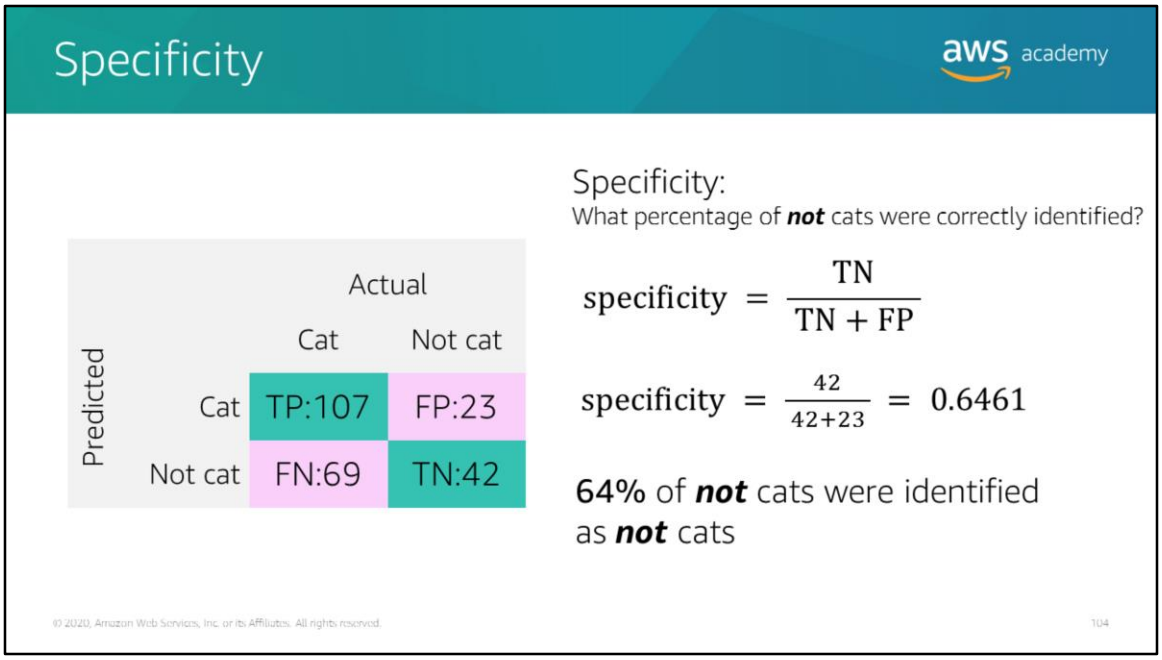
Which model is better? isn't the question to ask. What do you mean by *better*? Is *better* making sure that you find all the cats, even if it means many false positives? Or is *better* making sure the model is the most accurate?

It's difficult to see by looking at the two charts. What if you try several models, use multiple folds, and have hundreds to compare? To do these things, you must calculate some more metrics.



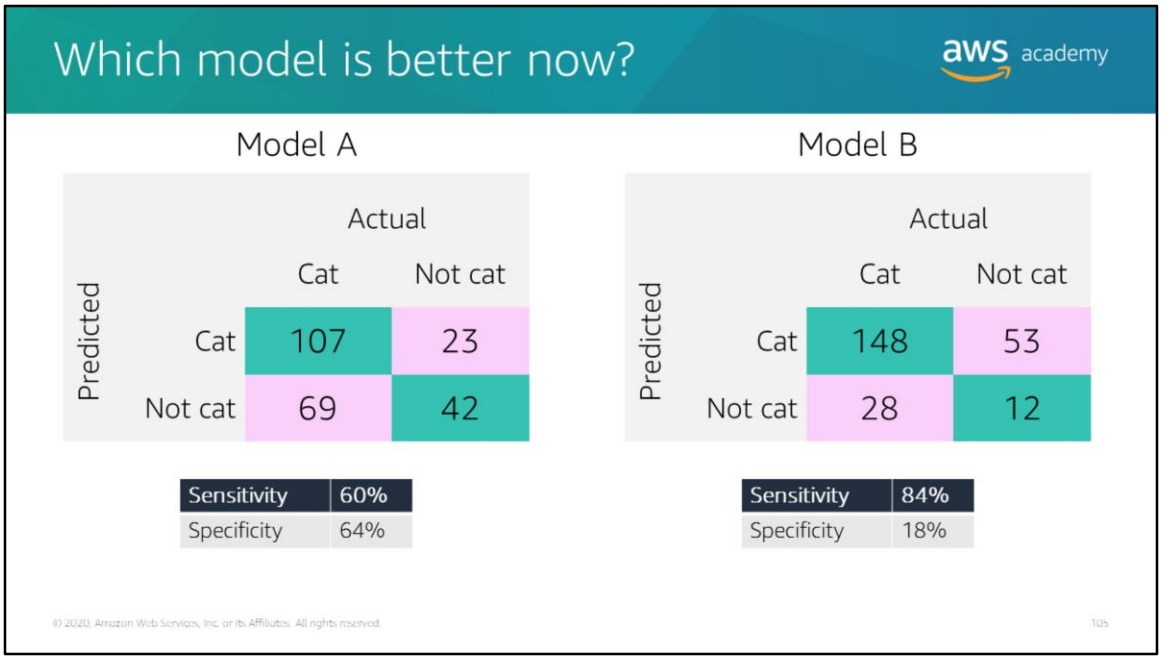
The first metric is *sensitivity*, which is sometimes referred to as *recall*, *hit rate*, or *true positive rate*. This metric is the percentage of positive identifications. In the cat example, it represents what percentage of cats are correctly identified.

To calculate sensitivity, you take the number of true positives (positive identifications of cats) and divide that number by the total number of actual cats.



Specificity, sometimes referred to as *selectivity* or *true negative rate*, is the percentage of negatives that were correctly identified. In the cat example, it is the number of images that were not cats that were identified as not cats.

To calculate specificity, you take the number of true negatives and divide that by the total number of actual negatives. In this example, that value is the number of not cats that were correctly identified, divided by the total number of actual not cats.



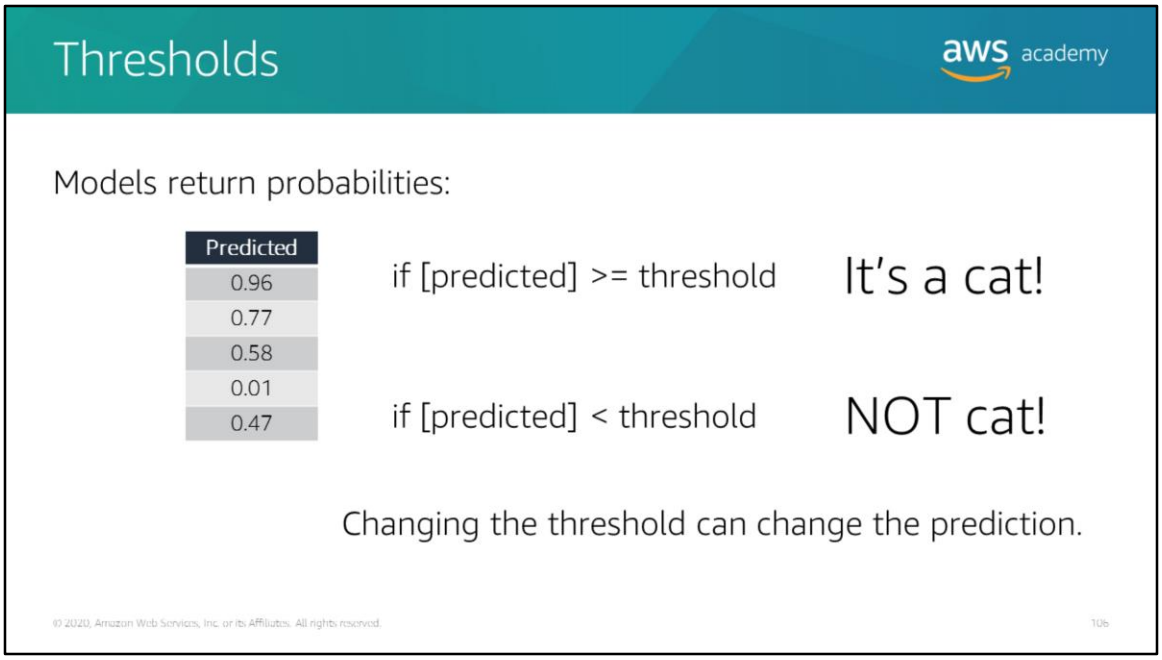
Now that you have these metrics for each model, it is easier to decide what your business goal is.

Which model do you choose if you want to identify as many cats as possible? Model B is a good answer, if you are not concerned about many false positives—that is, not cats that were incorrectly identified.

Which model do you choose if you want to make sure that you identify animals that are not cats? Model A might work, depending upon how many false negatives you can handle.

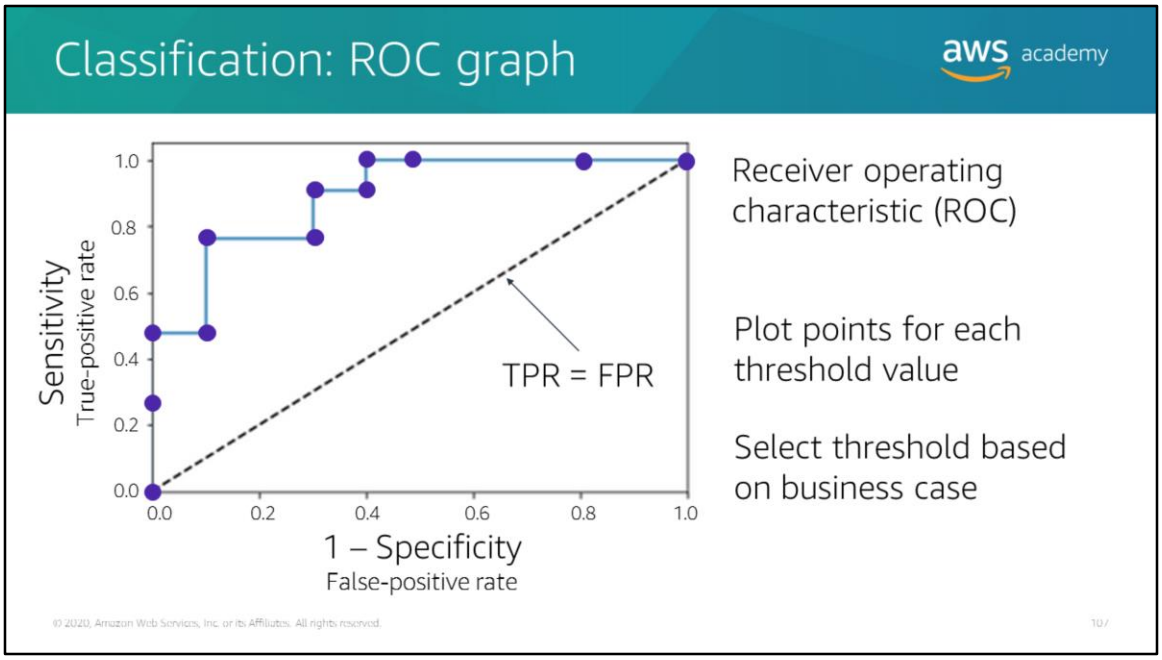
If you are classifying patients that have heart disease or not, which model is best? This scenario is where it gets interesting. A fun website might get a bad reputation if it can't identify cats correctly. However, if you are trying to diagnose patients, your focus might be different. It's important to understand the tradeoffs.

You can use more metrics to help with your decision making.



Classification models are going to return a probability for the target. This probability is a value between 0 and 1 of the input that belongs to the target class. To convert the value to a class, you must determine the threshold to use. You might think that this threshold is 50 percent, but you might change that figure to be lower or higher to improve your results.

Like with sensitivity and specificity, *classification* involves a tradeoff between correctly and incorrectly identifying classes. Changing the threshold can affect the outcome. Next, you will look at how you can visualize this.



A receiver operating characteristic (ROC) graph summarizes all the confusion matrices that each threshold produced. To build one, you calculate the sensitivity (or true-positive rate) against the false-positive rate for each threshold value. You then plot this value on a graph. You can calculate the false-positive rate by subtracting the specificity from 1. When those points are plotted, you can draw a line between them.

The dotted black line from (0,0) to (1,1), which is the line that represents the sensitivity-to-TPR ratio, is equal to the false-positive rate.

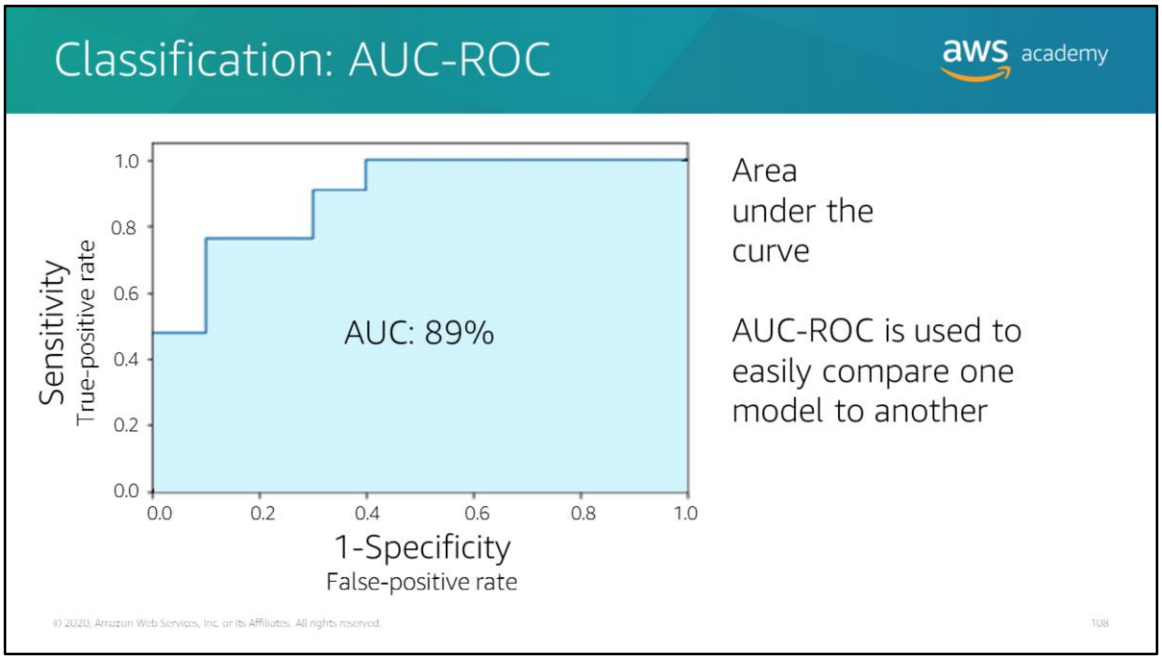
The point at (1,1) means that you would have correctly identified all the cats, but also incorrectly identified all the not cats. This result is not good. Any point on this line means that the proportion of correctly classified samples is the same as the proportion of incorrectly classified samples.

The point at (0,0) represents zero true positives and zero false positives.

A model that has high sensitivity and a low false-positive rate is usually your goal. It's better when the line between the threshold recordings is closer to the top-left corner.

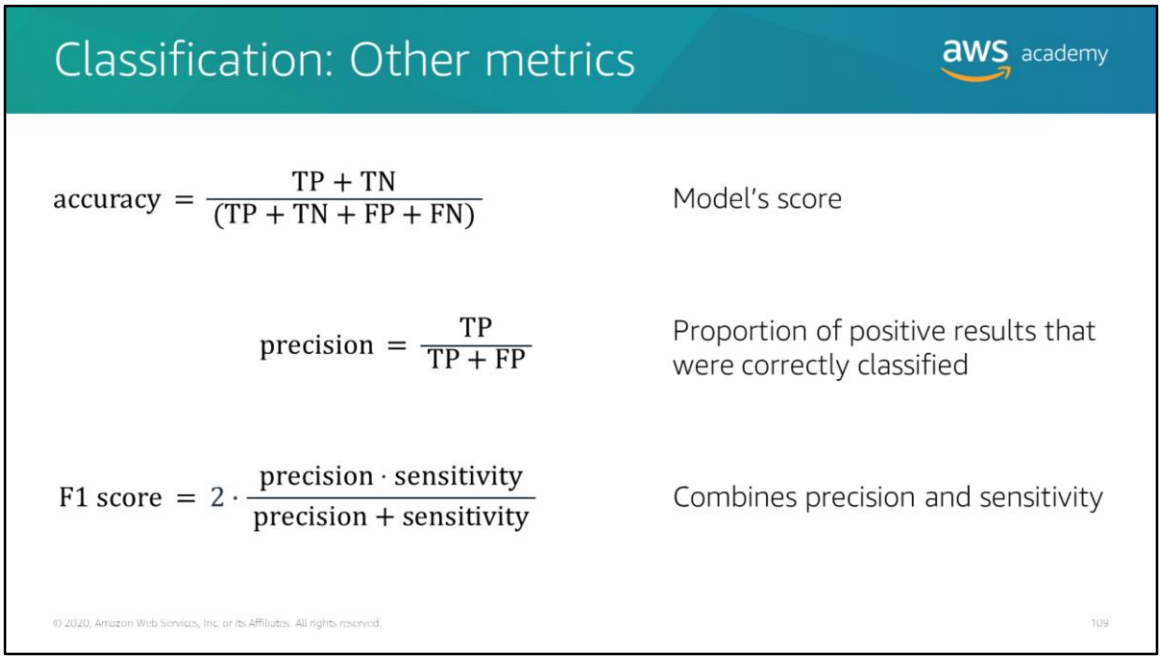
If you have the data from two models, you can plot out the ROC curve for each model, and

compare them. However, that can be tedious, so you can use the area under the curve (AUC)-ROC instead.



Area under the curve-receiver operator curve (AUC-ROC) is another evaluation metric. The AUC part is the area under the plotted line. The higher the AUC, the better the model is at predicting *cats* as cats and *not cats* as not cats.

You can use the AUC to quickly compare models with each other.



With the four numbers from the confusion matrix, you can calculate the model's *accuracy* (which is also known as its *score*). You can do this calculation by adding the correct predictions and then dividing that number by the total number of predictions.

Although accuracy is a widely used metric for classification problems, it has limitations. This metric is less effective when your dataset has many true negative cases. Think about the *cat/not cat* example. If the majority of your accuracy is based on true negatives, it says only that your model is good at predicting what isn't a cat. In this case, you can't feel confident in your model's ability to predict cats when you roll it out into production.

This problem leads to the importance of ensuring that the metric you choose for model evaluation aligns with your business goal. Think about the credit card fraud example. Using accuracy as your main metric is probably not a good idea in this case, when you have many true negatives. Your high true-negative number might hide the fact that your model's ability to identify cases of fraud (true positives) is not ideal. As a credit card company, it's probably unacceptable to have less-than-almost-perfect performance in identifying fraud cases, because it would drive customers away. Such a result would be the opposite of what you want to achieve from a business standpoint.

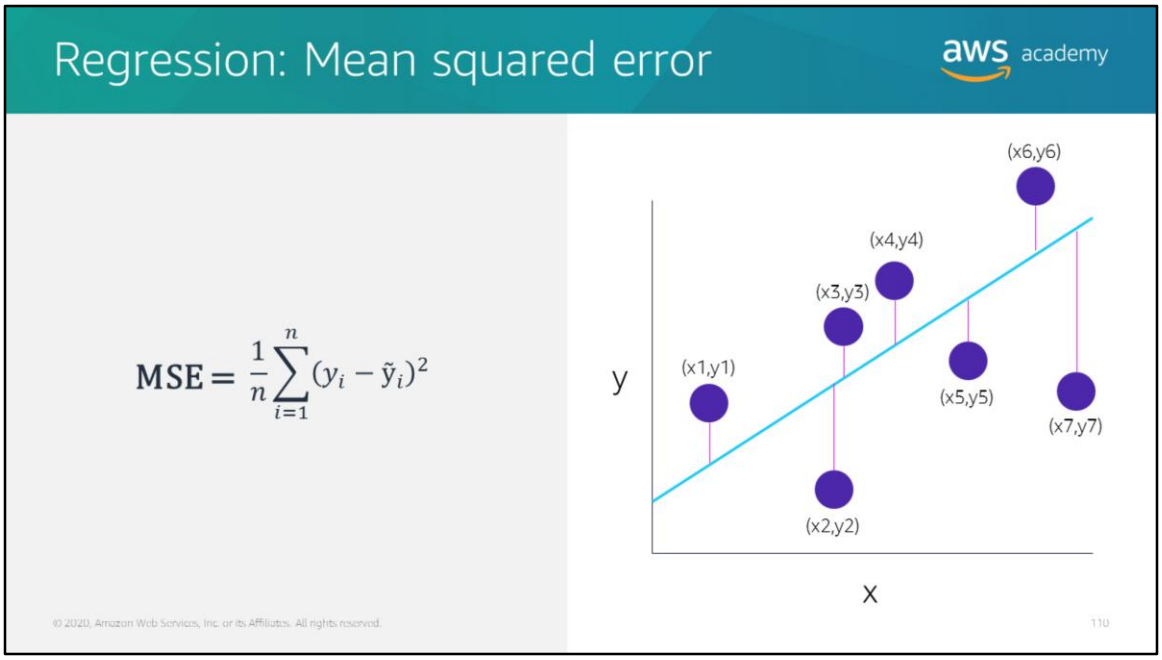
For this reason, two other metrics are often used in these situations. The first one is

precision, which removes the negative predictions from consideration. Precision is the proportion of positive predictions that are actually correct. You can calculate precision by taking the true positive and dividing it by the true positive plus the false positive.

When the cost of false positives is high in your particular business situation, precision might be a good metric. Think about a classification model that identifies emails as spam or not. You don't want your model to label an email as spam when the email is legitimate. In that case, it would prevent your users from seeing that email, which might be important.

Consider a model that must predict whether a patient has a terminal illness or not. In this case, the use of precision as your evaluation metric doesn't account for the false negatives in your model. It is vitally important and essential to the success of the model to identify the illness in a patient who actually has that illness. In this situation, sensitivity is a better metric to use.

However, it doesn't always need to be one or the other. The *F1 score* combines precision and sensitivity to give you one number that quantifies the overall performance of a particular ML algorithm. You might want to use the F1 score when you have a class imbalance, but you want to preserve the equality between precision and sensitivity.

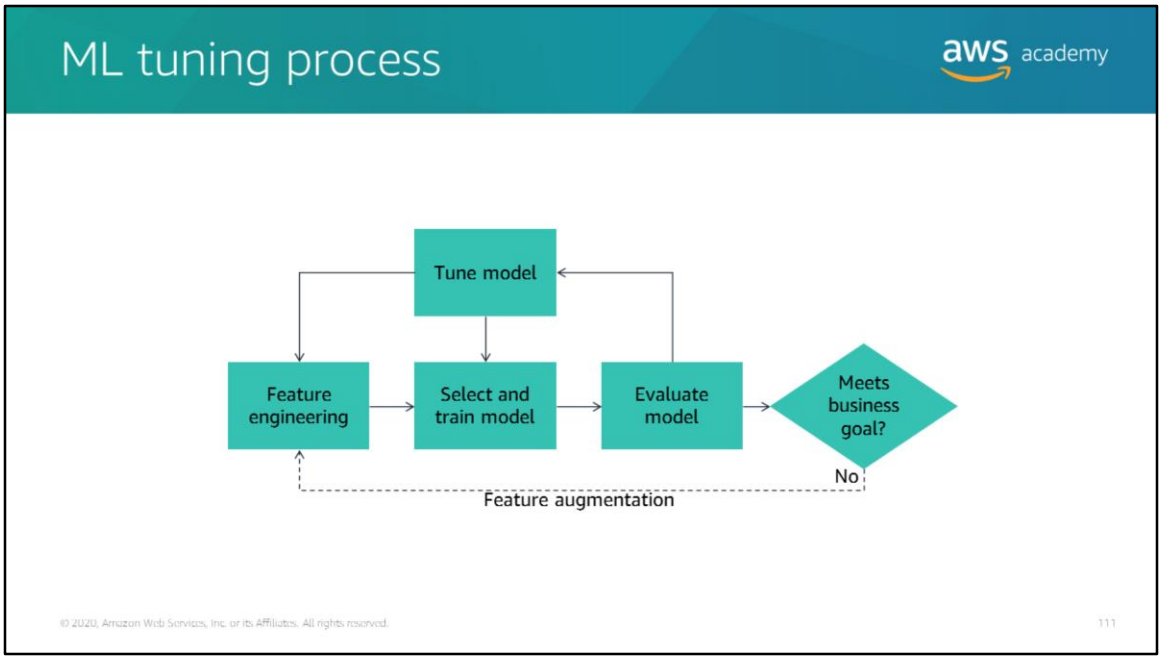


But what if you are dealing with a regression problem? In that case, you can use other common metrics to evaluate your model, including *mean squared error*.

Mean squared error is commonly used. Its general purpose is the same as what you saw with classification metrics. You determine the prediction from the model, and you compare the difference between the prediction and the actual outcome.

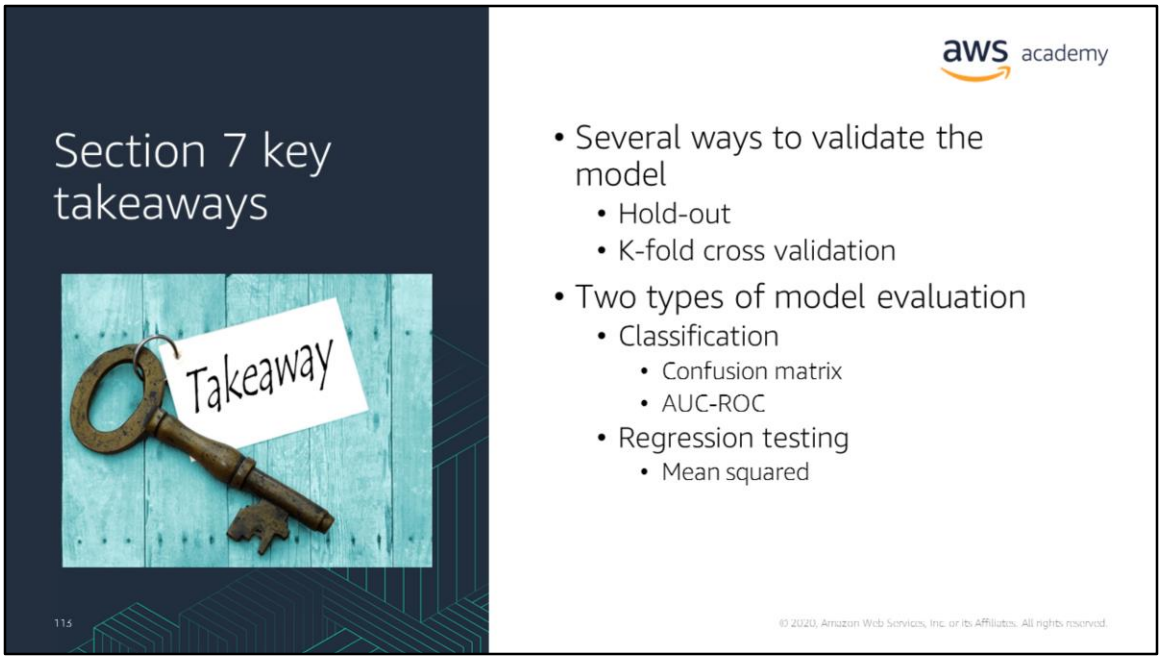
More specifically, you take the difference between the prediction and actual value, square that difference, and then add all the squared differences for all the observations. In scikit-learn, you can use the mean squared error function directly from the metrics library.

You can use other metrics for linear models, including R squared.



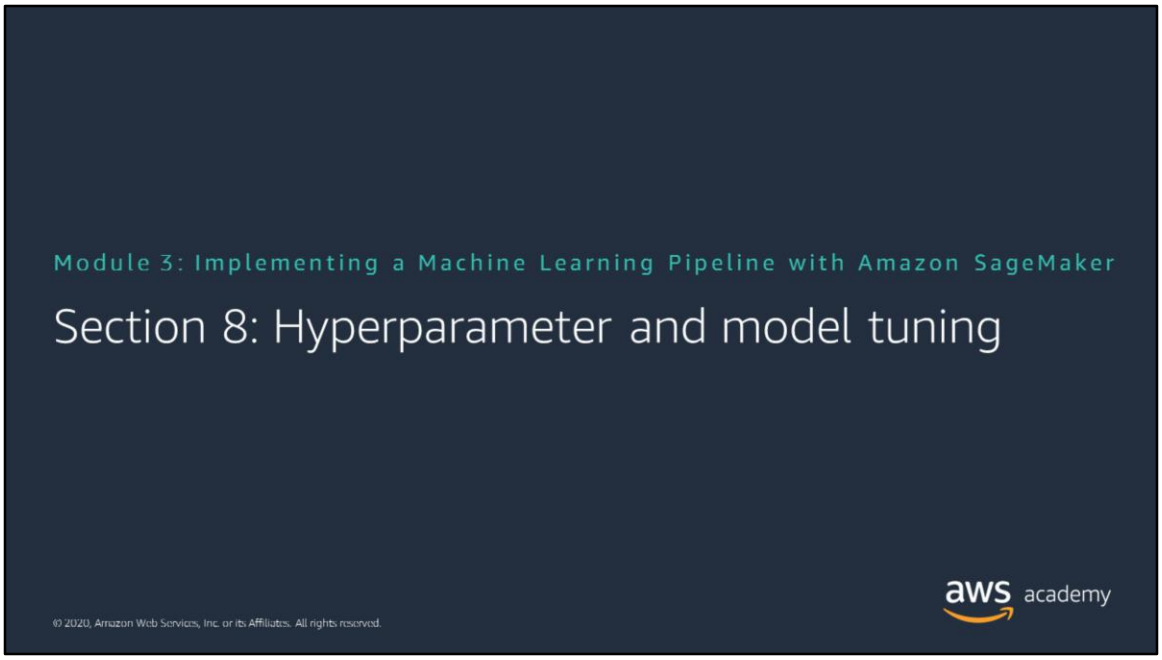
You have trained your model, performed a batch transformation on your test data, and calculated your metrics. Now, you can use these metrics to help you tune the model. You might select a different set of features and train the model again. Which was the better model? The metrics can help to inform you. You might also use different data and retrain the model with the same features. Recall the k-fold cross validation. Finally, you might tune the parameters of the model itself, which is the subject of the next section.

You will now complete Lab 6: Evaluating Model Accuracy.

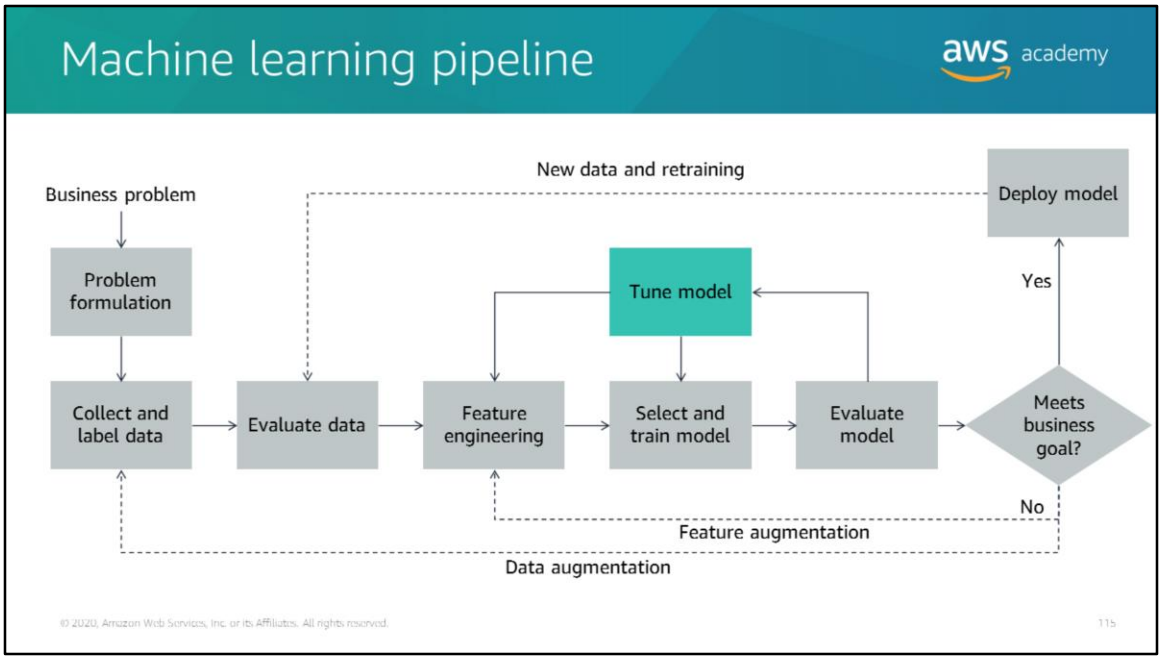


Some key takeaways from this section of the module include:

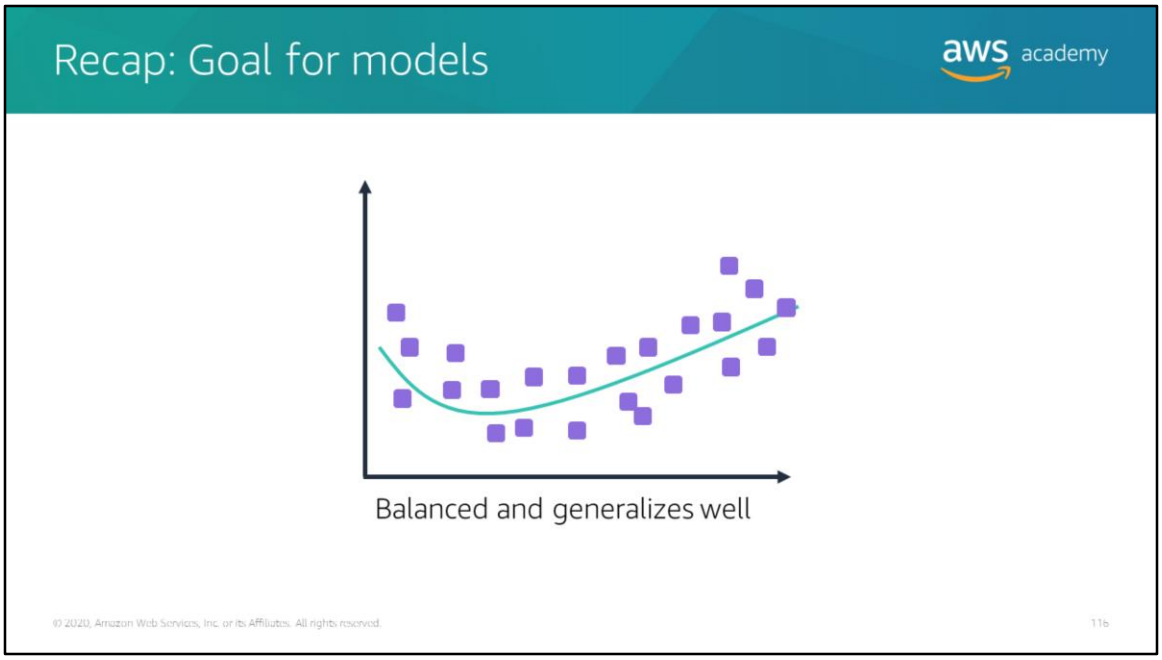
- To evaluate the model, you must have data that the model hasn't seen, through either a hold-out set or by using k-fold cross validation.
- Different machine learning models use different metrics.
    - Classification can use confusion matrix, and the AUC-ROC that can be generated from it.
    - Regression can use mean squared.



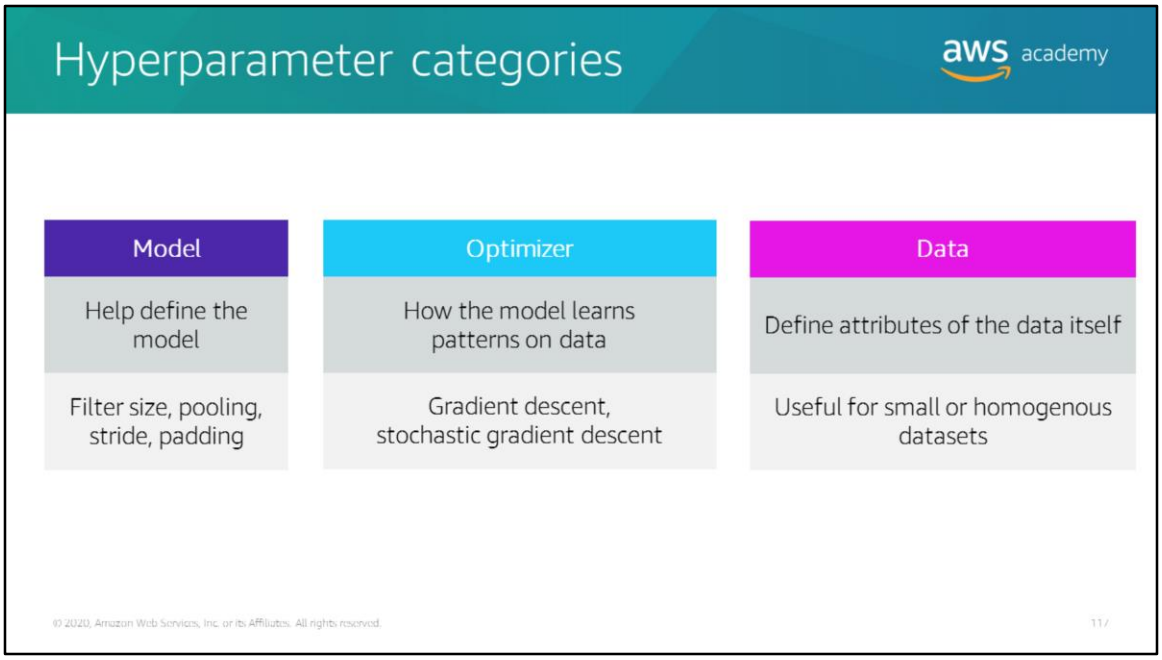
Introducing Section 8: Hyperparameter and model tuning



In this section, you will learn how to tune the model's hyperparameters to improve model performance.



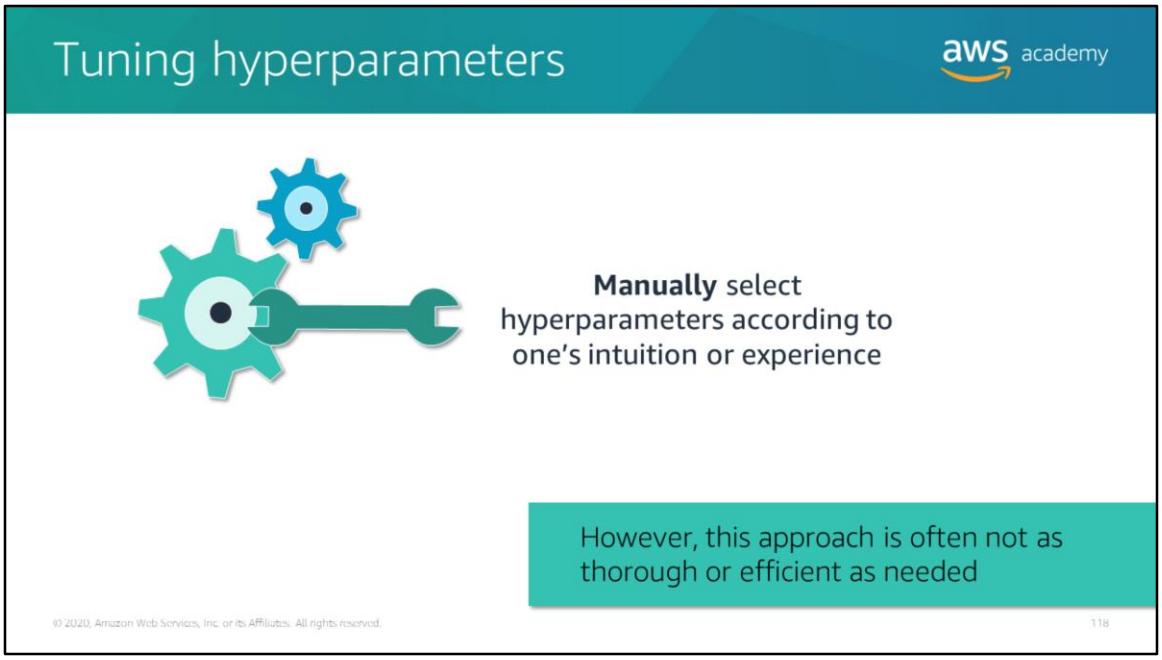
Recall from an earlier module that hyperparameters can be thought of as the knobs that tune the machine learning algorithm to improve its performance. Now, you will look more explicitly at tuning your models. You will learn more specific details about the types of hyperparameters and how to optimize hyperparameters.



Hyperparameters have a few different categories. The first kind is *model* hyperparameters, which help define the model itself. For instance, some neural-network-based models require you to define an architecture before you can start training them. The architecture includes a specific number of layers in the neural network and the activation functions that are used within. For example, in a neural network for a computer vision problem, additional attributes of the architecture must be defined. Such attributes include filter size, pooling, and the stride or padding.

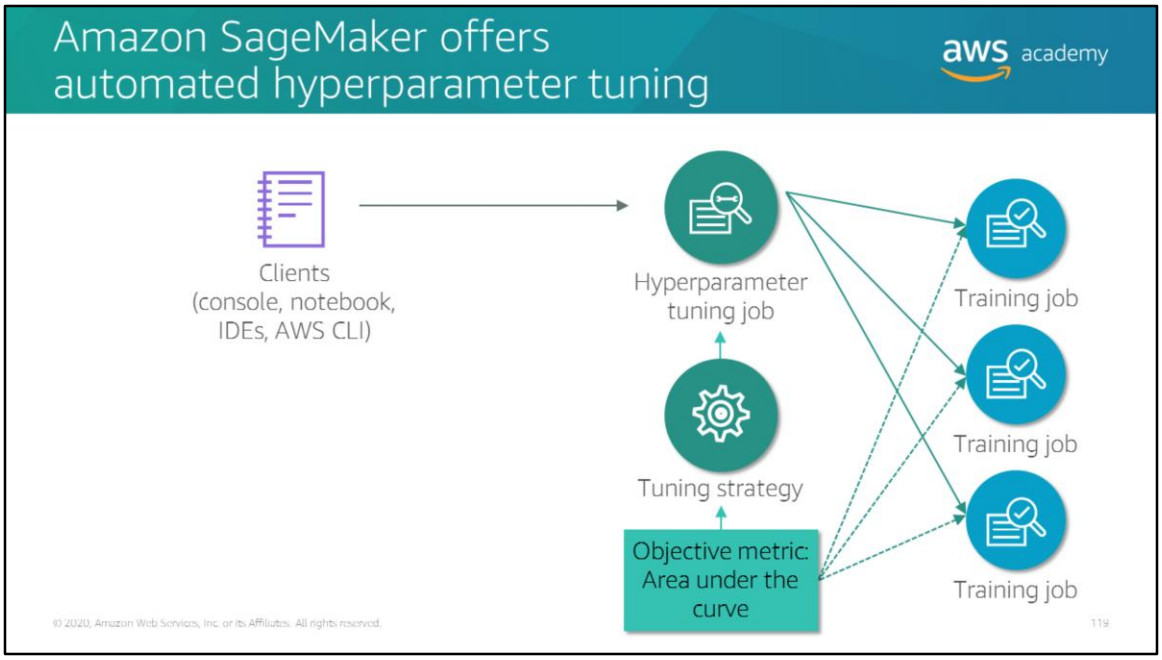
The second kind is *optimizer* hyperparameters. These hyperparameters are related to how the model learns the patterns that are based on data, and they are used for a neural-network model. These types of hyperparameters include optimizers like gradient descent and stochastic gradient descent. They can also include optimizers that use momentum like Adam, or initialize the parameter weights by using methods such as Xavier initialization or He initialization.

The third kind is *data* hyperparameters, which relate to the attributes of the data itself. They include attributes that define different data augmentation techniques, such as cropping or resizing for image-related problems. They are often used when you don't have enough data or enough variation in your data.

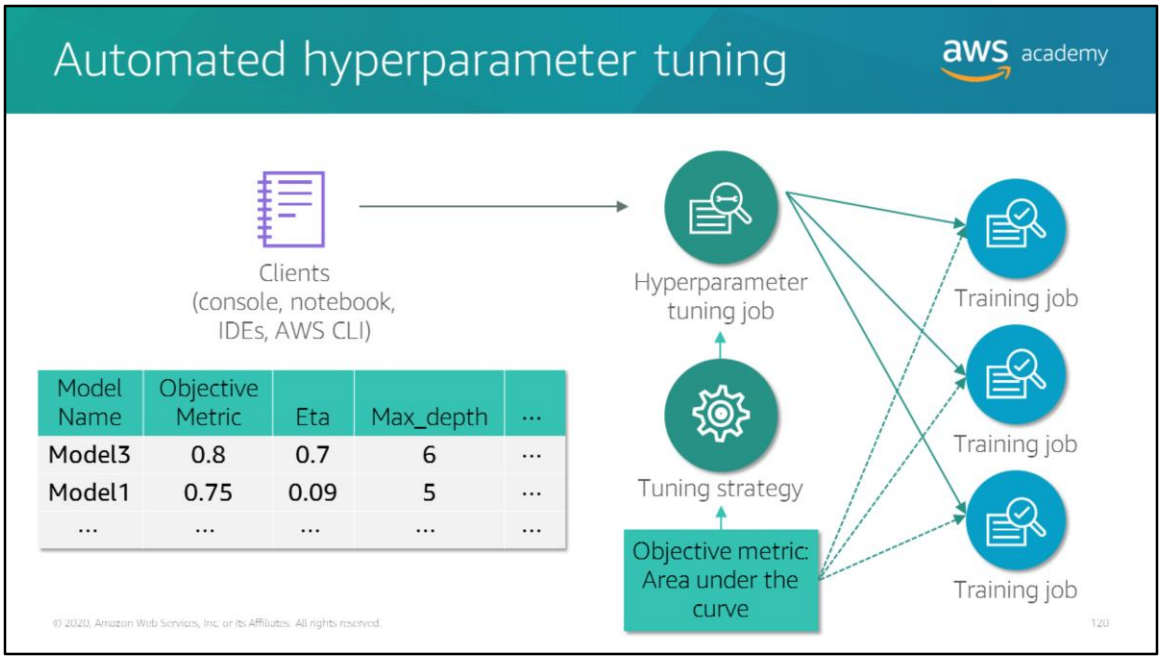


Tuning hyperparameters can be labor-intensive. Traditionally, this kind of tuning was done manually. Someone—who had domain experience that was related to that hyperparameter and use case—would manually select the hyperparameters, according to their intuition and experience. Then, they would train the model and score it on the validation data. This process would be repeated until satisfactory results were achieved.

This process is not always the most thorough and efficient way of tuning your hyperparameters.

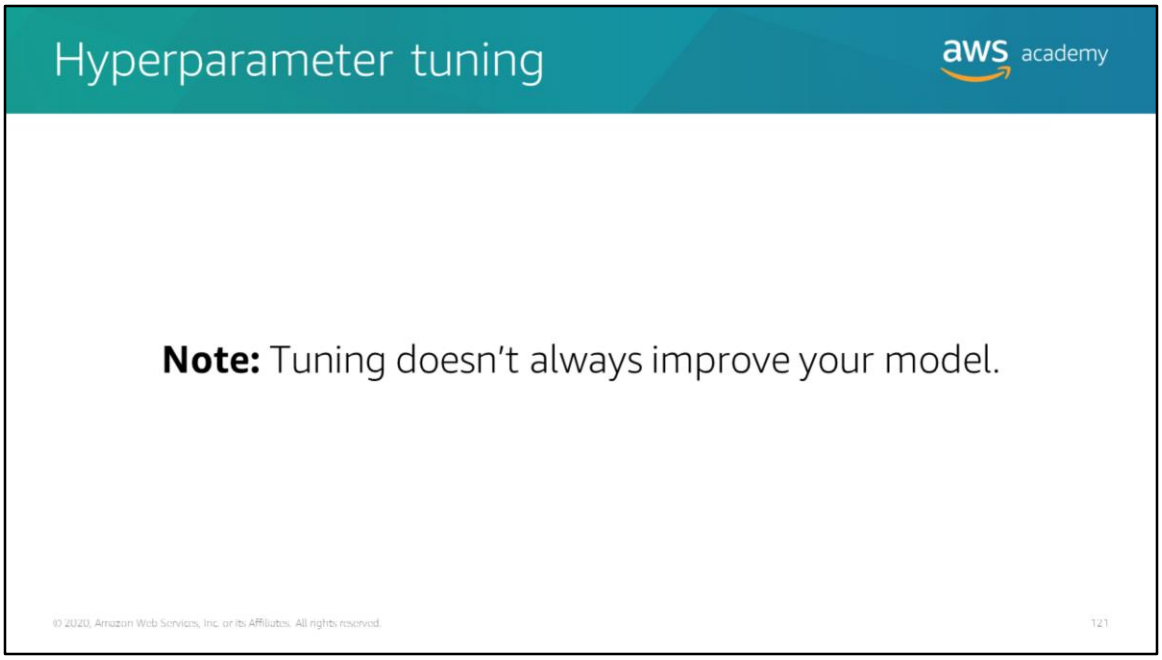


Amazon SageMaker enables you to perform automated hyperparameter tuning. Amazon SageMaker automatic model tuning, also known as *hyperparameter tuning*, finds the best version of a model by running many training jobs on your dataset. It uses the algorithm and ranges of hyperparameters that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by a metric that you choose. It uses Gaussian Process regression to predict which hyperparameter values might be most effective at improving fit. It also uses Bayesian optimization to balance exploring the hyperparameter space and exploiting specific hyperparameter values, when appropriate. And importantly, automatic model tuning can be used with the Amazon SageMaker built-in algorithms, prebuilt deep learning frameworks, and bring-your-own-algorithm containers.

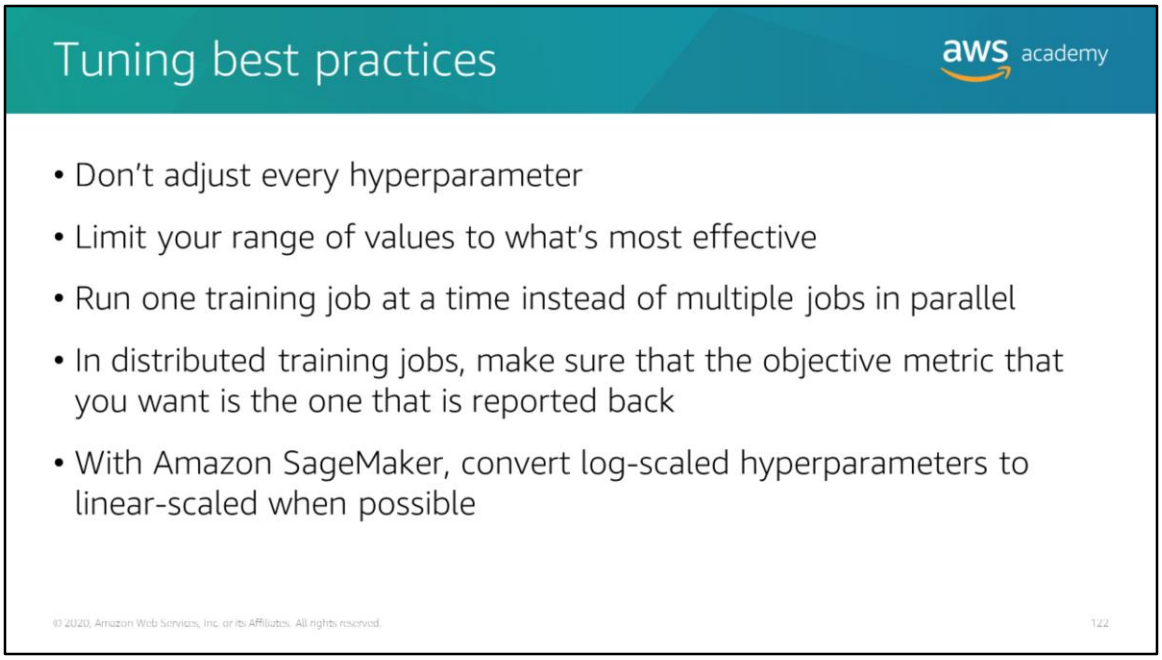


For example, suppose that you want to solve a binary classification problem on a fraud dataset. Your goal is to maximize the area under the curve (AUC) metric of the algorithm by training a linear learner algorithm model. You don't know which values of the *learning_rate*, *beta_1*, *beta_2*, and epochs to use to train the best model.

To find the best values for these hyperparameters, you can specify ranges of values for Amazon SageMaker hyperparameter tuning to search. It uses these ranges to run training jobs so it can find the combination of values that performs the best, as measured by the objective metric that you chose. It will then return the training job with the highest AUC.



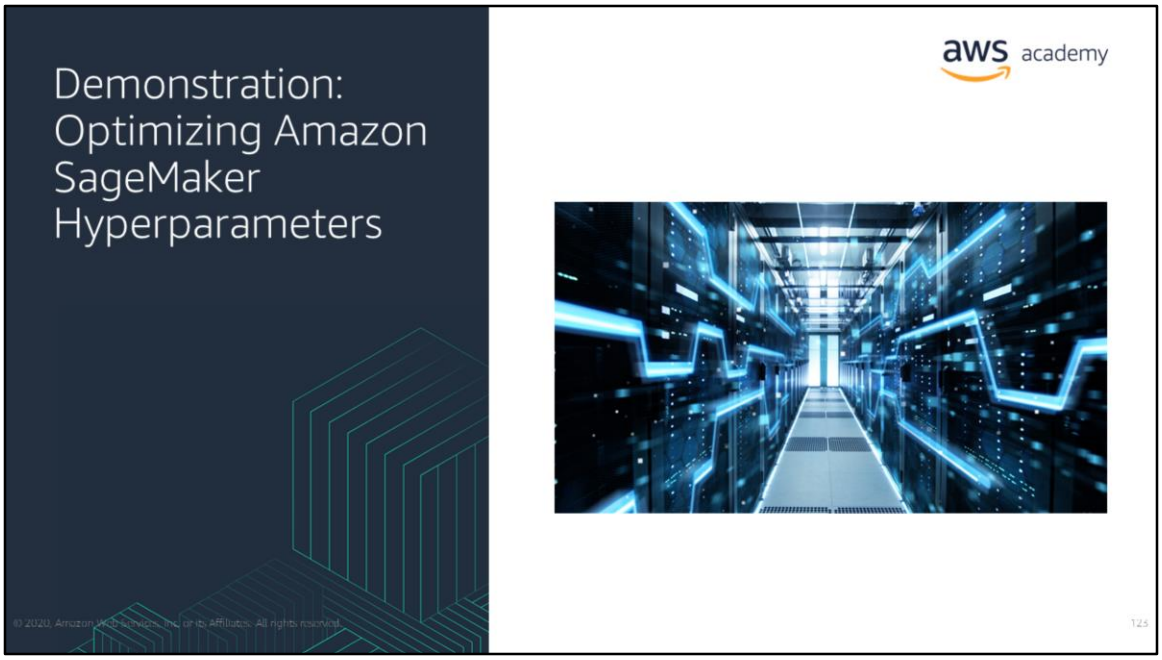
Hyperparameter tuning might not necessarily improve your model. It's an advanced tool for building machine solutions. As such, it should be considered part of the process of using the scientific method. When you build complex ML systems like deep learning neural networks, it is impractical to explore all the possible combinations.



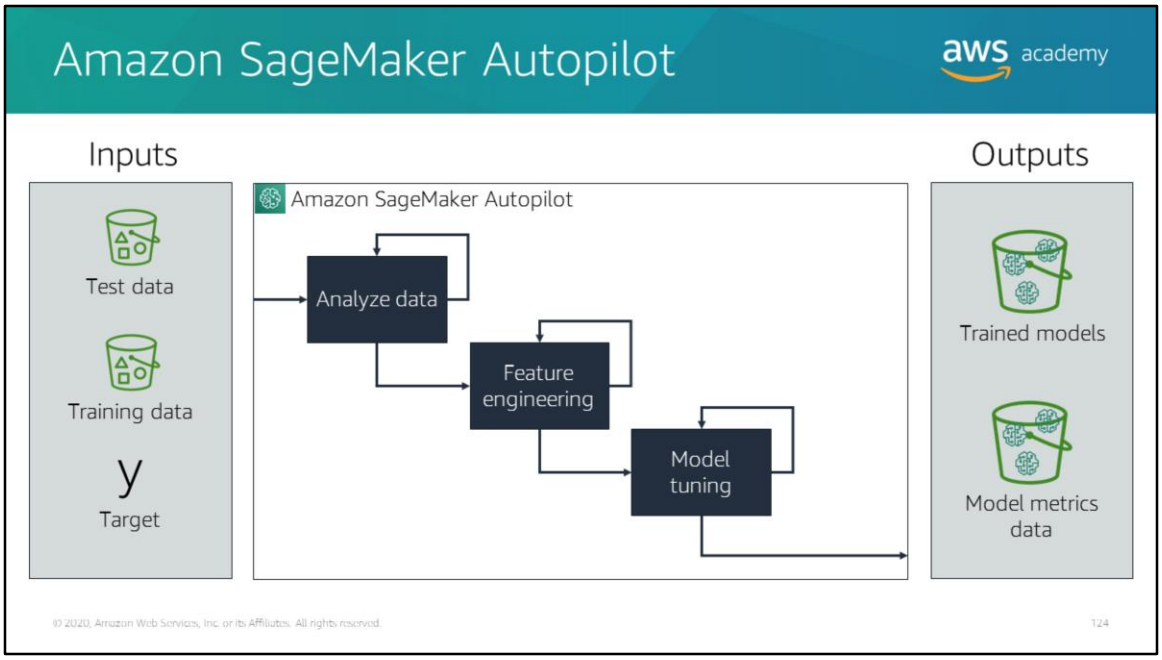
To improve optimization, use the following guidelines when you create hyperparameters.

- Limit the number of hyperparameters to the ones that you think are most likely to give you good results, instead of using all hyperparameters.
- The range of values for hyperparameters that you choose to search can significantly affect the success of hyperparameter optimization. You might want to specify a large range that covers every possible value for a hyperparameter. However, you get better results by limiting your search to a small range of values. If you get the best metric values within a part of a range, consider limiting the range to that part.
- Running more hyperparameter tuning jobs concurrently gets more work done quickly, but a tuning job improves only through successive rounds of experiments. Typically, running one training job at a time achieves the best results with the least amount of compute time.
- When a training job runs on multiple instances, hyperparameter tuning takes the last-reported objective metric from all instances of that training job. It then uses that metric as the value of the objective metric for that training job. You should design distributed training jobs so that you get the report of the objective metric that you want.
- During hyperparameter tuning, Amazon SageMaker attempts to determine whether your hyperparameters are log-scaled or linear-scaled. Initially, it assumes that hyperparameters are linear-scaled. If they should be log-scaled, it might take some time for Amazon

SageMaker to discover it. If you know that a hyperparameter should be log-scaled and can convert it yourself, doing so can improve hyperparameter optimization.

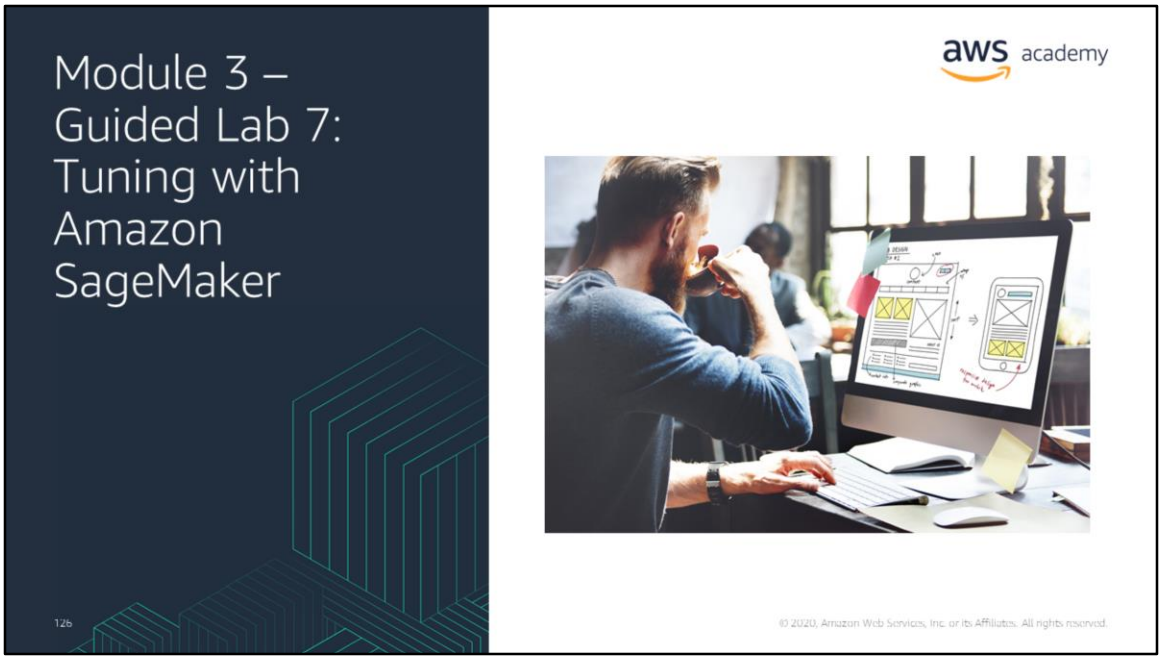


Your instructor will now either demonstrate how to optimize Amazon SageMaker hyperparameters or provide you with access to the recorded demonstration..

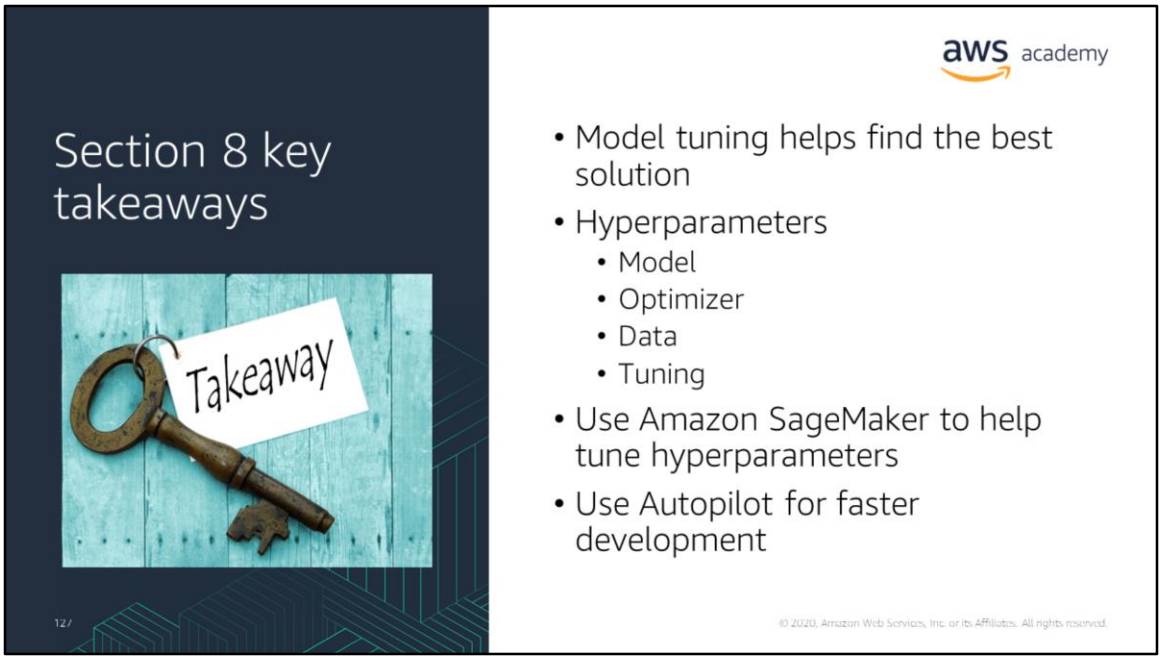


Now that you have walked through the end-to-end process of training and tuning an ML model, it's worth learning about Amazon SageMaker Autopilot. This service can help you find a good model, with little effort or input from you. With Autopilot, you create a job that supplies the test, training, and target. Autopilot analyzes the data, selects appropriate features, and then trains and tunes models. It documents the metrics and finds the best model that is based on the provided data. The results include the winning model, metrics, and a Jupyter notebook that you can use to investigate the results. This service doesn't remove your need to perform preprocessing of the data, but it can save time in feature selection and model tuning.

Your instructor will now either demonstrate how to run Amazon SageMaker Autopilot or provide you with access to the recorded demonstration.

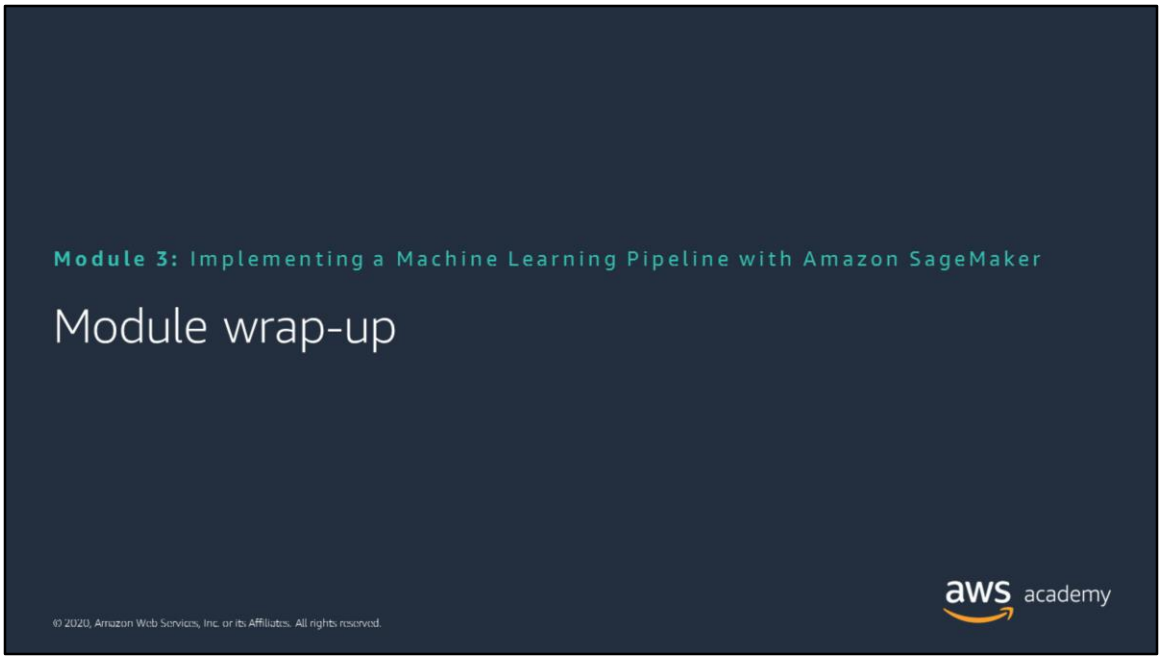


You will now complete Module 3 – Guided Lab 7: Tuning with Amazon SageMaker.

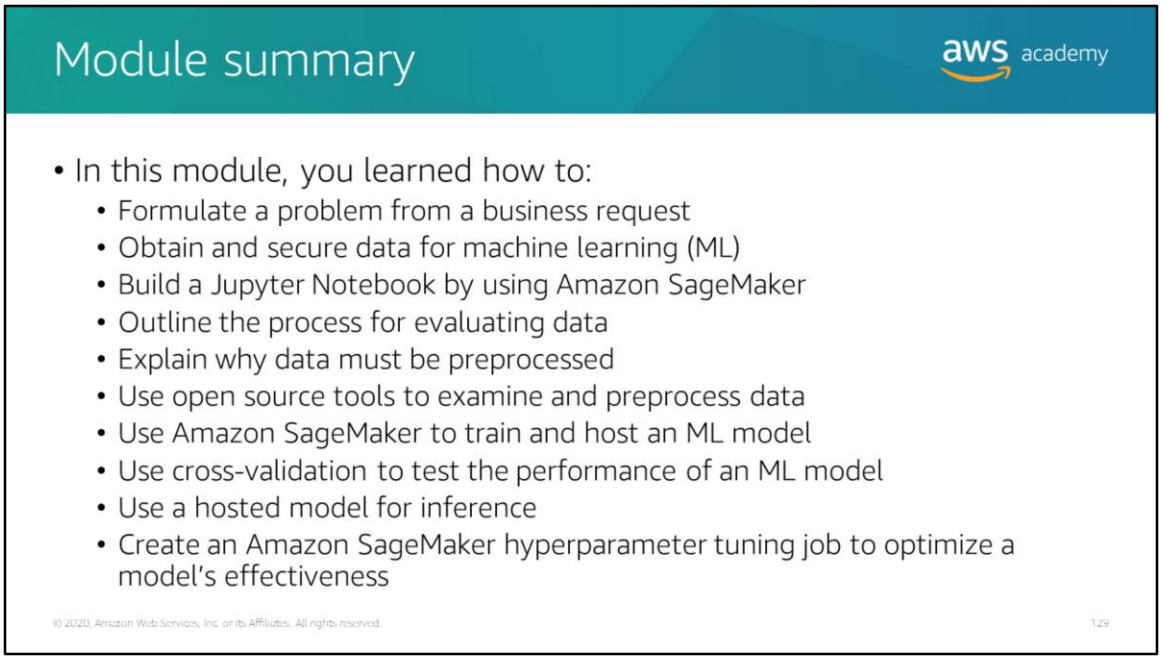


Some key takeaways from this section of the module include:

- Model tuning is important to find the best solution to your business problem.
- Hyperparameters can be tuned for the model, optimizer, and data.
- Amazon SageMaker can perform automatic hyperparameter tuning.
- Overall model development can be accelerated by using Autopilot.



It is now time to review the module and wrap up with a knowledge check.



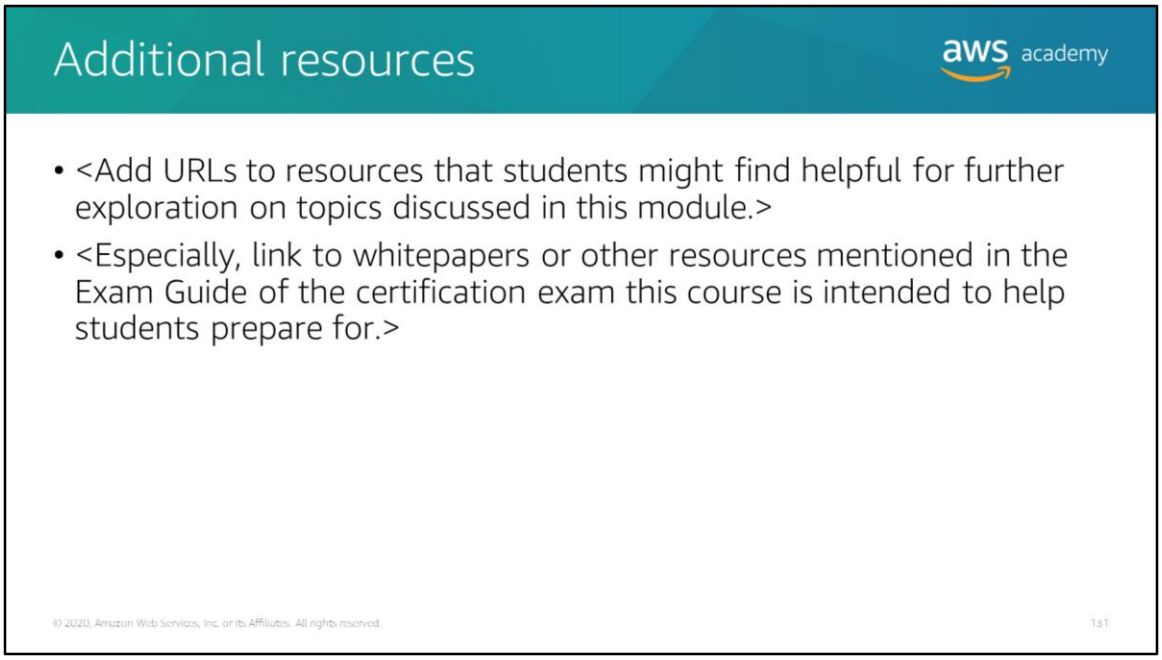
In summary, in this module, you learned how to:

- Formulate a problem from a business request
- Obtain and secure data for machine learning (ML)
- Build a Jupyter Notebook by using Amazon SageMaker
- Outline the process for evaluating data
- Explain why data must be preprocessed
- Use open source tools to examine and preprocess data
- Use Amazon SageMaker to train and host an ML model
- Use cross-validation to test the performance of an ML model
- Use a hosted model for inference
- Create an Amazon SageMaker hyperparameter tuning job to optimize a model's effectiveness

Complete the knowledge check

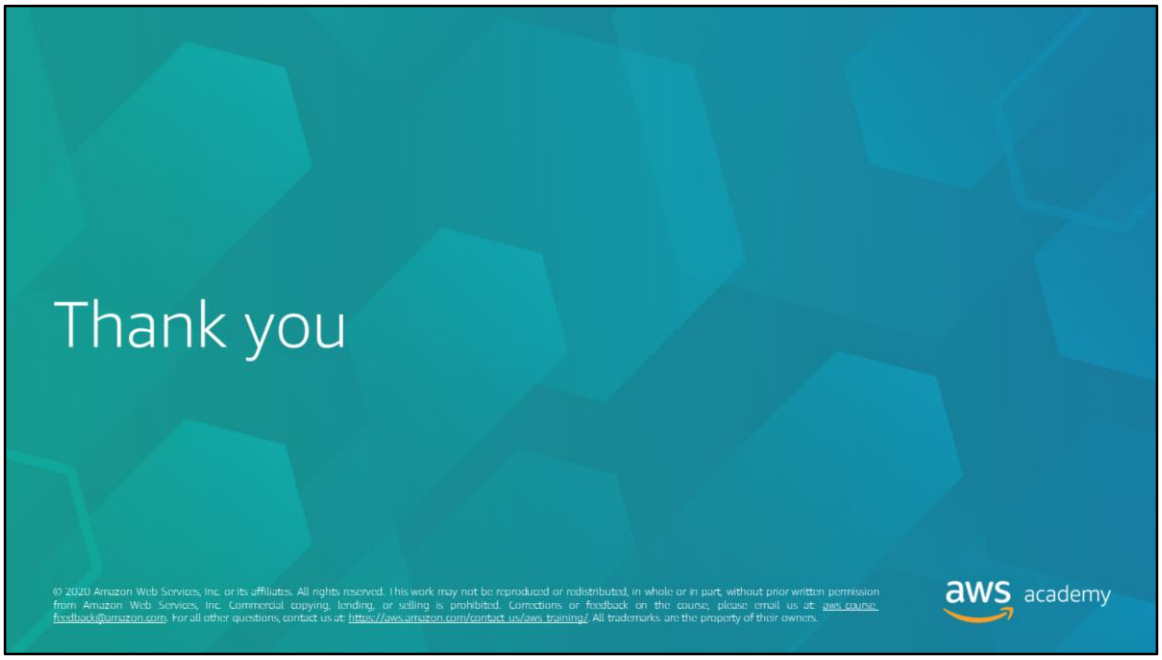
150

It is now time to complete the knowledge check for this module.



If you want to learn more about the topics that are covered in this module, you might find the following additional resources helpful:

- <insert hyperlink>



Thanks for participating!

