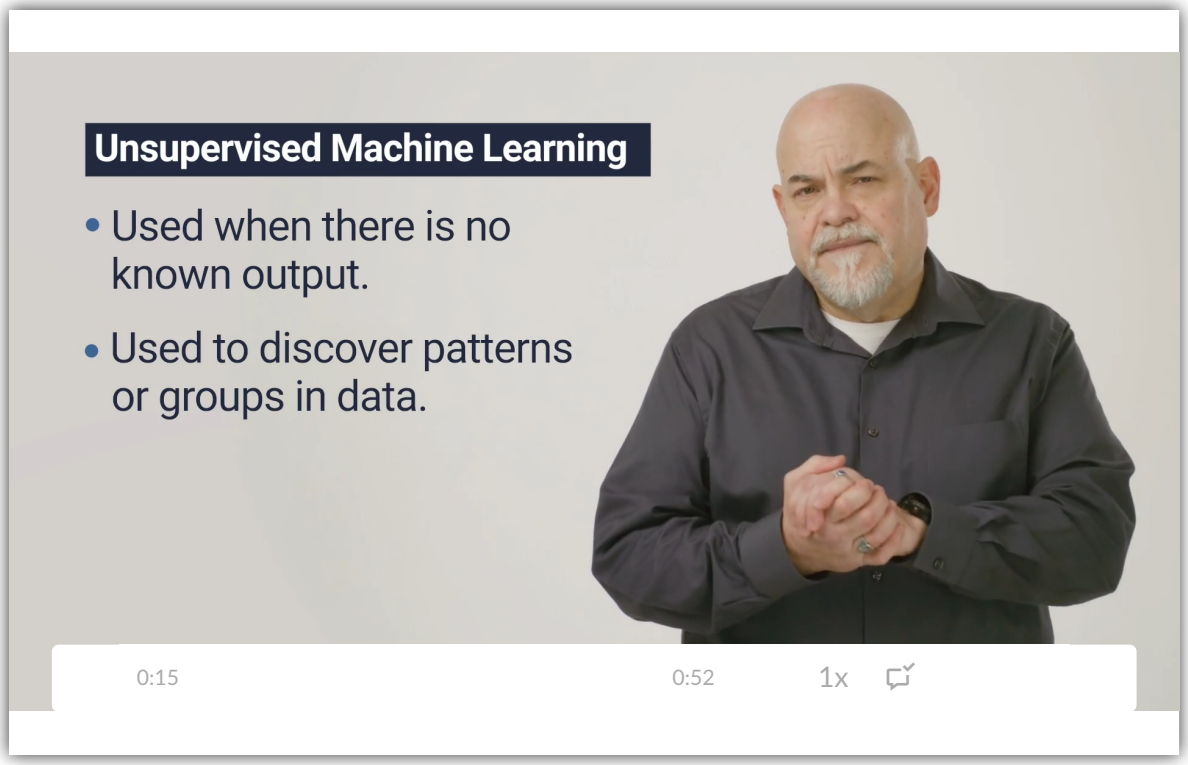
18.0.1 Using Unsupervised Learning to Discover Unknown Patterns



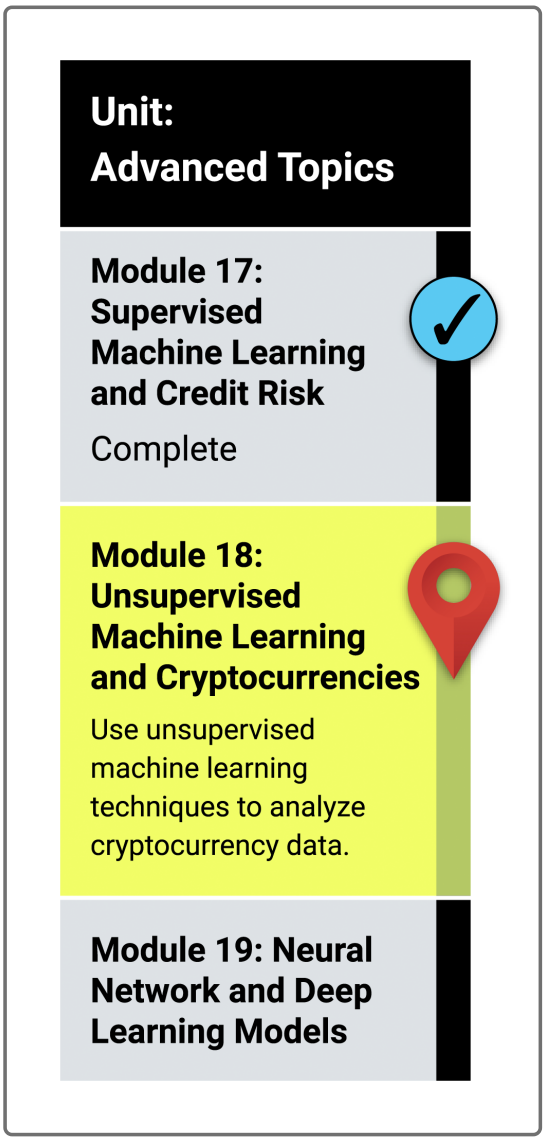
18.0.2 Module 18 Roadmap

Looking Ahead

This week we'll dive deeper into machine learning using unsupervised algorithms, which help us explore data when we're not sure what we're looking for. Now you can analyze data without a clear output in mind.

You'll work primarily with the K-means algorithm, the main unsupervised algorithm that groups similar data into clusters. We'll build on this by speeding up the process using principal component analysis (PCA), which employs many different features.

Before starting this module, you should have a strong understanding of training and testing datasets.



What You Will Learn

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

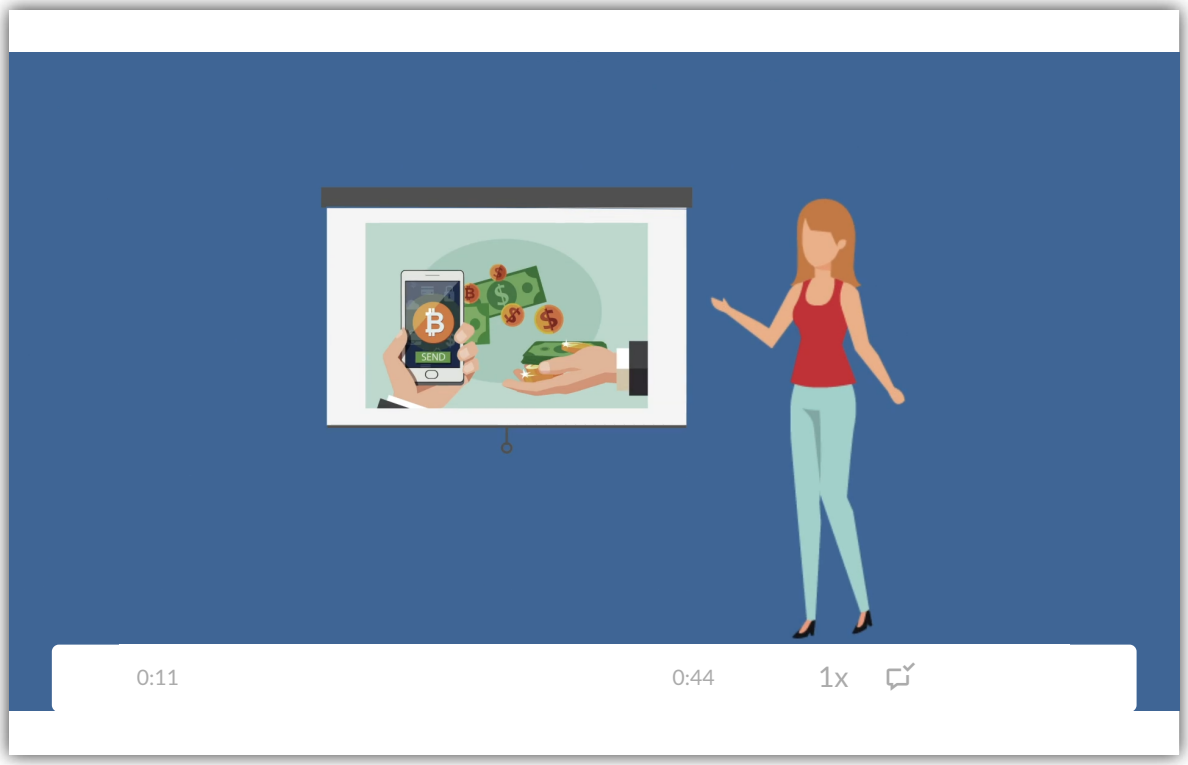
- Describe the differences between supervised and unsupervised learning, including real-world examples of each.
- Preprocess data for unsupervised learning.
- Cluster data using the K-means algorithm.
- Determine the best amount of centroids for K-means using the elbow curve.
- Use PCA to limit features and speed up the model.

Planning Your Schedule

Here's a quick look at the lessons and assignments you'll cover in this module. You can use the time estimates to help pace your learning and plan your schedule:

- Introduction to Module 18 (15 minutes)
- Supervised vs. Unsupervised Learning (1 hour)
- Data Preprocessing (2 hour)
- Clustering and the K-means Algorithm (1 hour)
- Using the Elbow Curve to Find Centroids (1 hour)
- Managing Data Features (2 hour)
- Hierarchical Clustering (2 hour)
- Application (5 hours)

18.0.3 Unsupervised Learning & Crypto



18.1.1 Supervised Learning Recap

Your childhood friend, Martha explained a project that she may get put on at work and wanted to pick your brain about machine learning. You start your conversation by going over the part of machine learning you are most familiar with, supervised learning.

Supervised learning is fairly simple: We use it when we know what we're looking for or what our output should be. Two of the most popular techniques are classification and regression. For example, if we want to see how Bitcoin performs over time, or if we want to figure out what kinds of coins a user is most likely to buy, we use supervised learning. However, we use unsupervised learning when we don't yet know the question we're asking of the data. In other words, we just want to figure out if there is anything at all the data can tell us.

REWIND

In supervised learning, first a model is initiated, or a template for the algorithm is created. Then it will analyze the data and attempt to learn patterns, which is also called fitting and training. After the data has been fit and trained, it will then make predictions.

Which of the following are types of supervised learning? Select all that apply.

- [ ] Classification
- [ ] Linear regression
- [ ] Clustering
- [ ] Logistic regression

Check Answer

Finish ▸

GITHUB

Create a new repository for this module named “Cryptocurrencies” and clone the empty repo into your class folder.

18.1.2 Unsupervised Learning

You and Martha are convinced: Unsupervised learning is the best way forward for your dataset because you are looking for any groupings, trends, or other information that could help you pitch cryptocurrencies to her firm.

Now the fun part of the project can start: learning how to use unsupervised algorithms. You know it might take a few tries to get it right, but if you can convince Accountability Accounting to invest in this cutting-edge financial system, you will truly be making a difference.

In supervised learning, the input data already has a paired outcome, which is plugged in to train the model to predict outcomes in new datasets. For example, we want to build a model that, when given unfamiliar data, can accurately predict the outcomes.

In **unsupervised learning**, there are two key differences from the above approach:

- There are no paired inputs and outcomes.
- The model uses a whole dataset as input.

Unsupervised learning is used in one of the following two ways:

- Transform the data to create an intuitive representation for analysis or to use in another machine learning model; or
- Cluster or determine patterns in a grouping of data, rather than to predict a classification.

For example, imagine a store owner is planning to stock up on back-to-school supplies. The owner's challenges are selecting from a wide range of school supplies, devoting considerable space to the inventory, and maintaining a fast inventory turnover rate, from stocking to selling.

The owner wants to predict the best time to start selling school supplies so staff can make room for the next big items to sell. The owner decides to focus on the previous year's data, which includes the number of items sold, when they were sold, and the area schools' start dates for 10, 20, and 30 days prior to the items' sell dates.

Since the owner knows what they are looking for—the best time to start selling school supplies—what would be the best approach?

- ○ Logistic regression
- ○ Classification
- ○ Unsupervised learning
- ○ Linear regression

Check Answer

Finish ▸

Now, imagine the owner notices that when customers come in to buy school supplies, they also purchase non-school items, giving other categories a bump in sales. There are a wide variety of products to offer different consumers, so the owner wants to know which consumers might buy more products and what items they might want to buy. We don't really

know *how* to group the data, but we know that grouping the data will be instructive. You'll need to factor in many data points, such as age, race, gender, location, and much more.

Since the owner doesn't know exactly how many groups or what kinds of customers those groups contain, what would be the best method for finding out?

- Logistic regression
- Classification
- Unsupervised learning
- Linear regression

Check Answer

Finish ▸

18.1.3 Types of Unsupervised Learning: Transformations and Clustering

Unsupervised learning appears to be a great choice! You and Martha decided to follow your intuition and research which type of unsupervised learning is best: transformations or clustering.

There are two types of unsupervised learning: transformations and clustering algorithms.

Transformations

We use **transformations** when we need to take raw data and make it easier to understand. Transformations also can help prepare data so that it can be used for other machine learning algorithms.

For example, if you had cable TV viewer data, it might contain location, viewing duration, and viewer churn and retention during commercials and after programs ended. Transformations can reduce the dimensional representation, which simply means we'll be decreasing the number of features used for the model or analysis. After doing so, the data can either

be processed for use in other algorithms or narrowed down so it can be viewed in 2D.

Clustering Algorithms

We use **clustering algorithms** to group similar objects into clusters. For example, if a cable service wants to group those with similar viewing habits, we would use a clustering algorithm.

Match the following types of unsupervised learning to their respective descriptions.

	Transformations	Clustering Algorithms
This type takes the data and transforms it in a way that becomes more readable. It also helps prepare the data for different machine learning algorithms.	○	○
This type can reduce the dimensional representation, decreasing features used for the model or analysis. After doing so, it can either process data for use in other algorithms or narrow down data so it can be viewed in 2D.	○	○
This type entails grouping similar objects into clusters.	○	○

Check Answer

Finish ▸

Challenges of Unsupervised Learning

IMPORTANT

> Unsupervised learning isn't the solution for every data analytic challenge. Just because supervised learning might not work for one

> situation doesn't mean unsupervised learning will work instead. Understanding the data and what can be done with it is an important first step before choosing an algorithm.

Recall that unsupervised learning does not take in any pairing of input and outcomes from the data—it only looks at the data as a whole. This can cause some challenges when running the algorithm. Since we won't know the outcome it's predicting, we might not know that the result is correct.

This can lead to issues where we're trying to decide if the model has provided any helpful information that we can use to make decisions in the real world. For example, our store owner might run a model that ends up grouping the type of people by how much they're buying. This could be useful in some contexts—for example, knowing who the top spenders are—but it might not help the store owner better organize the store for maximum purchases per person, or understand the differences in product preferences between top purchasers.

The only way to determine what an unsupervised algorithm did with the data is to go through it manually or create visualizations. Since there will be a manual aspect, unsupervised learning is great for when you want to explore the data. Sometimes you'll use the information provided to you by the unsupervised algorithm to transition to a more targeted, supervised model.

As with supervised learning, data should be preprocessed into a correct format with only numerical values, null value determination, and so forth. The only difference is unsupervised learning doesn't have a target variable—it only has input features that will be used to find patterns in the data. It's important to carefully select features that could help to find those patterns or create groups.

The next section will cover data preprocessing and data munging, and provide a refresher on Pandas and data cleaning. First, you'll need to install the necessary libraries for practice.

18.1.4 Install Your Tools

Before getting started, make sure that you're set with any libraries you'll be using while exploring unsupervised learning.

If you already have some libraries installed from previous modules, you may skip those parts of the installation instructions.

Scikit-learn

To install the Scikit-learn library, follow these steps:

1. Open your terminal and activate your PythonData environment.
2. Run the following command:

```
conda install scikit-learn
```

3. After installation, you're all set.

Plotly

To install the Python Plotly library, follow these steps:

1. Open your terminal and activate your PythonData environment.
2. Run the following command:

```
conda install plotly
```

3. After installation, you’re all set.

hvPlot

To install the hvPlot visualization library, follow these steps:

1. Open your terminal and activate your PythonData environment.
2. Run the following command:

```
conda install -c pyviz hvplot
```

3. After installation, you’re all set.

18.2.1 Steps for Preparing Data

After digging into unsupervised learning a bit, you realize that your first step in convincing Accountability Accountants to invest in cryptocurrency is to preprocess the data.

You and Martha open up the dataset to get started preprocessing it. Together, you will want to manage unnecessary columns, rows with null values, and mixed data types before turning your algorithm loose.

Data Selection

Before moving data to our unsupervised algorithms, complete the following steps for preparing data:

1. Data selection
2. Data processing
3. Data transformation

Data selection entails making good choices about which data will be used. Consider what data is available, what data is missing, and what data can be removed. For example, say we have a dataset on city weather that

consists of temperature, population, latitude and longitude, date, snowfall, and income. After looking through the columns, we can readily see that population and income data don't affect weather. We might also notice some rows are missing temperature data. In the data selection process, we would remove the population and income columns as well as any rows that don't record temperatures.

Data Processing

Data processing involves organizing the data by formatting, cleaning, and sampling it. In our dataset on city weather, if the date column has two different formats—mm-dd-yyyy (e.g., 01-23-1980) and month-data-year (e.g., jan-23-1980)—we would convert all dates to the same format.

Data Transformation

Data transformation entails transforming our data into a simpler format for storage and future use, such as a CSV, spreadsheet, or database file. Once our weather data is cleaned and processed, we would export the final version of the data as a CSV file for future analysis.

18.2.2 Pandas Refresher

When it comes to preprocessing data, you have good news for Martha. The Pandas Python library is really good at this! When Martha asks for a quick refresher on how to use Pandas for data munging, you know just the dataset to use—the iris dataset from the University of California, Irvine (UCI) Machine Learning Repository.

Pandas is a Python library that is excellent for data munging. We'll be using the **iris dataset from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/iris)** , a common dataset used throughout machine learning:

1. Store the raw **iris.csv (https://courses.bootcampspot.com/courses/138/files/26354/download?wrap=1)** **(https://courses.bootcampspot.com/courses/138/files/26354/download?wrap=1)** in a folder that is easy to access.
2. Open a new Jupyter Notebook.
3. Import your libraries:

```
import pandas as pd
```

4. To load the dataset in a Pandas DataFrame, enter the code below. Be sure to use the path to the stored CSV file (stored in an easy-to-access location):

```
file_path = "<folder path to stored data sets>/iris.csv"
iris_df = pd.read_csv(file_path)
iris_df.head()
```

5. Select the fields of data you want:

```
file_path = "Resources/iris.csv"
iris_df = pd.read_csv(file_path)
iris_df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Which field should we drop from the DataFrame?

- sepl_length
- sepal_width
- petal_length
- petal_width
- class

Check Answer

Finish ▸

NOTE

> Unsupervised learning will be used to determine the class of the iris plants later on in the module.

6. Drop the class field using the code below:

```
new_iris_df = iris_df.drop(['class'], axis=1)
new_iris_df.head()
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

SKILL DRILL

Try reordering the columns so the sepal and petal lengths are the first two columns and the widths are the last two columns.

Cleaning this dataset appears complete with all the data in numerical form and the same type, so no data processing is needed. However, you'll encounter data transformations on datasets that contain categorical data or non-numeric features (e.g., transforming male and female categorical values to 0 and 1, respectively).

Finally, the preprocessed DataFrame is saved on a new CSV file for future use. This is done by storing the file path in a variable, then using the Pandas `to_csv()` method to export the DataFrame to a CSV by supplying the file path and file name as arguments, as shown below:

```
output_file_path = "<path to folder>/new_iris_data.csv"
new_iris_df.to_csv(output_file_path, index=False)
```

18.2.3 Preprocessing Data With Pandas

Martha is super grateful for the Pandas refresher—it's always fun to work with such a classic dataset! Now that you are both on the same page, you start to think critically about your cryptocurrency dataset.

As mentioned, we don't know the output of the data, but that doesn't mean we shouldn't think about our data or that we should carelessly plug it into a model.

Let's take a look at how we should start our data processing by loading in the **shopping_data.csv (https://courses.bootcampspot.com/courses/138/files/28066/download?wrap=1)**  **(https://courses.bootcampspot.com/courses/138/files/28066/download?wrap=1)** :

```
# Data loading
file_path = "Resources/shopping_data.csv"
df_shopping = pd.read_csv(file_path, encoding="ISO-8859-1")
df_shopping.head(5)
```

	CustomerID	Card Member	Age	Annual Income	Spending Score (1–100)
0	1	Yes	19.0	15000	39.0
1	2	Yes	21.0	15000	81.0
2	3	No	20.0	16000	6.0
3	4	No	23.0	16000	77.0
4	5	No	31.0	17000	40.0

Questions for Data Preparation

Unsupervised learning doesn't have a clear outcome or target variable like supervised learning, but it is used to find patterns. By properly preparing the data, we can select features that help us find patterns or groups.

Before we begin, consider these questions:

- What knowledge do we hope to glean from running an unsupervised learning model on this dataset?
- What data is available? What type? What is missing? What can be removed?
- Is the data in a format that can be passed into an unsupervised learning model?
- Can I quickly hand off this data for others to use?

Let's address the first question on our list:

What knowledge do we hope to glean from running an unsupervised learning model on this dataset?

It’s a shopping dataset, so we can group together shoppers based on spending habits.

18.2.4 Data Selection

It's not every day that you and Martha have a chance to convince an accounting firm to invest in cryptocurrency! So, you want to make sure you know how to select the data that will best help the model determine patterns or grouping.

To help us select the data, let's return to some of the questions on our list.

What data is available?

First, account for the data you have. After all, you can't extract knowledge without data. We can use the `columns` method and output the columns, as shown below:

```
# Columns
df_shopping.columns
```

Looking at the columns, we see there is data for CustomerID, Gender, Age, Annual Income, and Spending Score:

```
Index(['Customer ID', 'Card Member', 'Age', 'Annual Income',
       'Spending Score (1-100)'],
      dtype='object')
```

Now that we know what data we have, we can start thinking about possible analysis. For example, data points for features like Age and Annual Income might appear in our end result as groupings or clusters. However, there are no data points for items purchased, so our algorithms cannot discover related patterns.

What type of data is available?

Using the `dtypes` method, confirm the data type, which also will alert us if anything should be changed in the next step (e.g., converting text to numerical data). All the columns we plan to use in our model must contain a numerical data type:

```
# List dataframe data types
df_shopping.dtypes
```

```
CustomerID                 int64
Card Member               object
Age                      float64
Annual Income              Int64
Spending Score (1-100)   float64
dtype: object
```

Which column doesn't contain a data type we can use for our unsupervised learning model?

- CustomerID
- Card Member
- Annual Income
- All columns contain correct data types.

Check Answer

Finish ▸

What data is missing?

Next, let's see if any data is missing. Unsupervised learning models can't handle missing data. If you try to run a model on a dataset with missing data, you'll get an error such as the one below:

```
ValueError: Input contains NaN, infinity or a value too large for dtype('flo
```

If you initially had hoped to produce an outcome using a type of data, but it turned out more than 80% of those rows are empty, then the results won't be very accurate!

For example, return to our Age and Income groups: If it turns out there are 1,200 rows without any Age data points, then we clearly can't use that column in our model. There is no set cutoff for missing data—that decision is left up to you, the analyst, and must be made based on your understanding of the business needs.

NOTE

> Handling missing data is a complex topic that is out of scope for this unit. However, if you're interested, read this **article (https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4)** on the possible approaches to handling missing data.

Pandas has the `isnull()` method to check for missing values. We'll loop through each column, check if there are null values, sum them up, and print out a readable total:

```
# Find null values
for column in df_shopping.columns:
    print(f"Column {column} has {df_shopping[column].isnull().sum()}
null values")
```

```
Column CustomerID had 0 null values
Column Card Member has 2 null values
Column Age has 2 null values
Column Annual Income has 0 null values
Column Spending Score (1-100) has 1 null values
```

There will be a few rows with missing values that we'll need to handle. The judgement call will be to either remove these rows or decide that the dataset is not suitable for our model. In this case, we'll proceed with handling these values because they are a small percentage of the overall data.

IMPORTANT

> When deciding to proceed, the percentage of data missing isn't always the only determining factor. See the Note callout above for a resource on handling missing data.

What data can be removed?

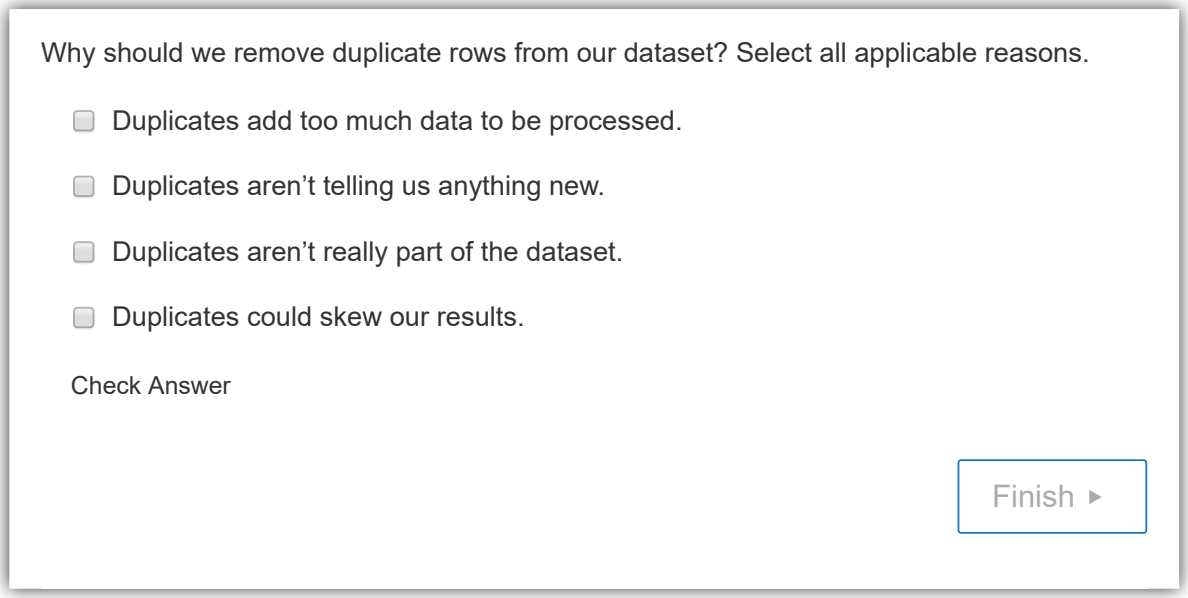
You have begun to explore the data and have taken a look at null values. Next, determine if the data can be removed. Consider: Are there string columns that we can't use? Are there columns with excessive null data points? Was our decision to handle missing values to just remove them?

In our example, there are no string type columns, and we made the decision that only a few rows have null data points, but not enough to remove a whole column's worth.

Rows of data with null values can be removed with the `dropna()` method, as shown below:

```
# Drop null rows
df_shopping = df_shopping.dropna()
```

Duplicates can also be removed.



Use the `duplicated().sum()` method to check for duplicates, as shown below:

```
# Find duplicate entries
print(f"Duplicate entries: {df_shopping.duplicated().sum()}")
```

```
Duplicate entries: 0
```

Looks good with no duplicates!

We also can remove data that doesn't tell us anything interesting. Knowing this, is there anything we can remove from this DataFrame?

- Card Member
- CustomerID
- Annual Income
- None of the above

Check Answer

Finish ▸

To remove the column, just enter the code below:

```
# Remove the CustomerID Column
df_shopping.drop(columns=["CustomerID"], inplace=True)
df_shopping.head()
```

	Card Member	Age	Annual Income	Spending Score (1-100)
0	Yes	19.0	15000	39.0
1	Yes	21.0	15000	81.0
2	No	20.0	16000	6.0
3	No	23.0	16000	77.0
4	No	31.0	17000	40.0

18.2.5 Data Processing

Now that you know what kind of data you want to work, it's time to meet the needs for your unsupervised model.

The next step is to move on from what you (the user) want to get out of your data and on to what the unsupervised model needs out of the data.

Recall that in the data selection step, you, as the user, are exploring the data to see what kind of insights and analysis you might glean. You reviewed the columns available and the data types stored, and determined if there were missing values.

For data processing, the focus is on making sure the data is set up for the unsupervised learning model, which requires the following:

- Null values are handled.
- Only numerical data is used.
- Values are scaled. In other words, data has been manipulated to ensure that the variance between the numbers won't skew results.

REWIND

Recall that when features have different scales, they can have a disproportionate impact on the model. The unscaled value could lead to messy graphs. Therefore, it is important to understand when to scale and normalize data. For example, if four columns of data are single digits, and the fifth column is in the millions, we would need to scale the fifth column to align the other four.

Let's return again to our list of questions.

Is the data in a format that can be passed into an unsupervised learning model?

We saw before that all our data had the correct type for each column; however, we know that our model can't have strings passed into it.

To make sure we can use our string data, we'll transform our strings of `Yes` and `No` from the Card Member column to `1` and `0`, respectively, by creating a function that will convert `Yes` to a `1` and anything else to `0`.

The function will then be run on the whole column with the `.apply` method, as shown below:

```
# Transform String column
def change_string(member):
    if member == "Yes":
        return 1
    else:
        return 0


df_shopping["Card Member"] = df_shopping["Card Member"].apply(change_string)
df_shopping.head()
```

	Card Member	Age	Annual Income	Spending Score (1-100)
0	1	19.0	15000	39.0
1	1	21.0	15000	81.0
2	0	20.0	16000	6.0
3	0	23.0	16000	77.0
4	0	31.0	17000	40.0

Also, there is one more thing you may notice about the data. The scale for Annual Income is much larger than all the other values in the dataset. We can adjust this format by dividing by 1,000 to rescale those data points, as shown below:

```
# Transform annual income
df_shopping["Annual Income"] = df_shopping["Annual Income"] / 1000
df_shopping.head()
```

	Genre	Age	Annual Income	Spending Score (1-100)
0	1	19	15.0	39
1	1	21	15.0	81
2	0	20	16.0	6
3	0	23	16.0	77
4	0	31	17.0	40

SKILL DRILL

Reformat the names of the columns so they contain no spaces or numbers.

18.2.6 Data Transformation

You have done all this work to get your data ready to be passed into an unsupervised learning model, but what about when other teams need to use this data? The next step is transforming your data into a convenient way for others to use in the future.

Data transformation involves thinking about the future. More times than not, there will be new data coming into your data storage (a place where raw data is stored before being touched), with many people working on different types of data analysis. We want to make sure that whoever wants to use the data in the future can do so.

Let's return once more to our list of questions.

Can I quickly hand off this data for others to use?

The data now needs to be transformed back into a more user-friendly format. It would be nice if everyone was as great with DataFrames as you two; unfortunately, that is not the case. You'll want to convert the final product into a common data type like CSV or Excel files.

Now that our data has been cleaned and processed, it is ready to be converted to a readable format for future use:

```
# Saving cleaned data
file_path = "<path to your folder>/shopping_data_cleaned.csv"
df_shopping.to_csv(file_path, index=False)
```

SKILL DRILL

Try to export the data to a different format.

Now you know the questions to ask about your data and understand the Pandas processes used to help answer those questions. Different datasets have different issues. With practice, you'll get better at identifying these.

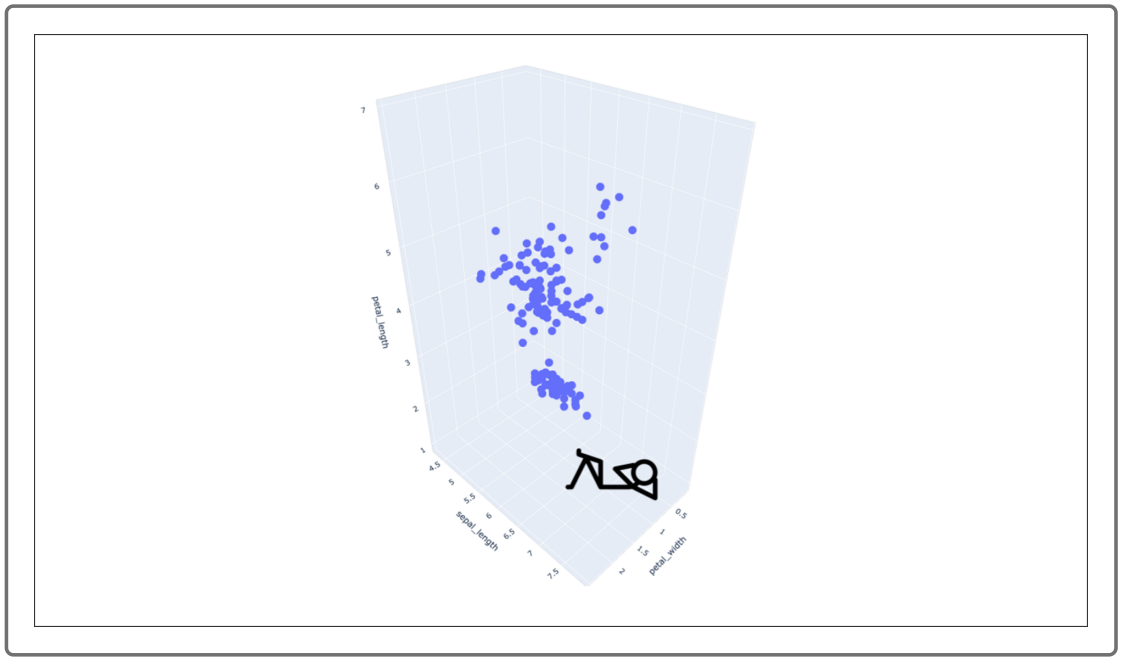
18.3.1 Clustering Data

Now that you and Martha are asking the right questions about your data and processing it to a state that you feel comfortable with, it's time to put it into action and start clustering!

Clustering is a type of unsupervised learning that groups data points together. This group of data points is called a **cluster**.

Imagine you are in a roomful of spheres (data points). You want to learn more about them, so you start to observe them.

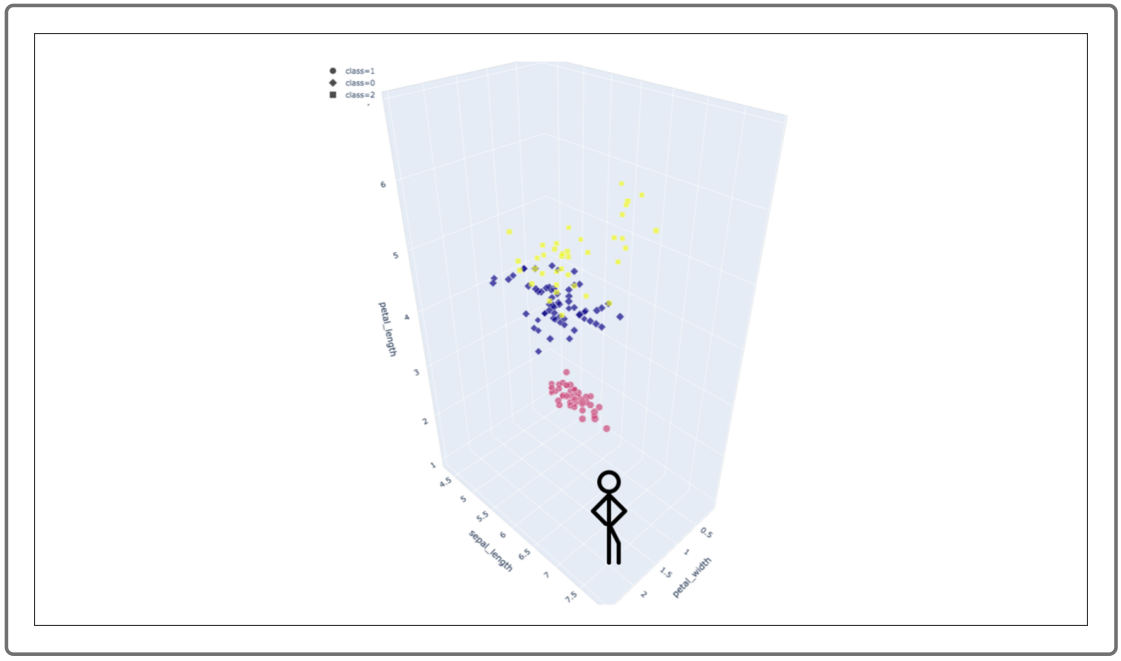
Every sphere represents a flower, and three axes represent features of flowers. After observing the flowers, you discover patterns when you combine the three features:



We can see that spheres (data points) with similar features seem to be closer together than data points with dissimilar features. We can use this spatial information to group similar data points together.

If you look at the flower features in the graph below, and start to plot them, they'll start to form groups on the graph.

After we plot the data points, they start to form three different groups, or clusters:



18.3.2 K-means Algorithm

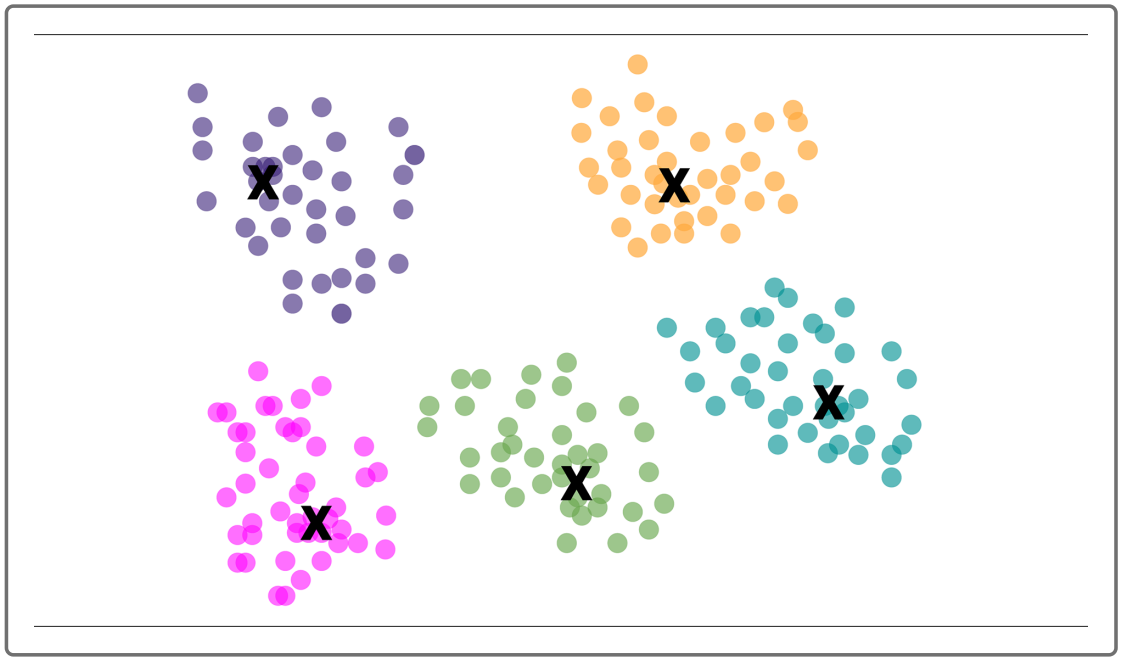
Clustering is exactly what we want from unsupervised learning, but exactly how can we determine the clusters? Martha knows she needs to group the cryptocurrency data, but she isn't sure how to determine the number of groups to create. One of the most popular ways to cluster is by using the K-means algorithm.

K-means is an unsupervised learning algorithm used to identify and solve clustering issues.

K represents how many clusters there will be. These clusters are then determined by the **means** of all the points that will belong to the cluster.

The K-means algorithm groups the data into K clusters, where belonging to a cluster is based on some similarity or distance measure to a centroid.

A **centroid** is a data point that is the arithmetic mean position of all the points on a cluster:



The centroid is found by taking the mean of all the x values in a cluster, and the mean of all the y values in a cluster.

What would be the centroid to the points (8, 1), (5, 9), (7, 3), and (4,7)?

- (6, 0)
- (6, 5)
- (0, 5)
- (0, 0)

Check Answer

Finish ▸

The following examples use the cleaned iris data from the previous section.

Why is the cleaned version of the iris data being used, not the original data?

- The original version of the data contains a column that contains text data.
- The original dataset contained too many columns.
- The original data needs multiple adjustments.
- The original dataset didn't contain all of the data.

Check Answer

Finish ▸

Code along to see how we can use K-means on the iris dataset. To get started, we'll import our libraries as well as the library for the `KMeans` algorithm from the `sklearn` library, as shown below:

```
import pandas as pd
import plotly.express as px
import hvplot.pandas
from sklearn.cluster import KMeans
```

After we have imported our library, we'll store the cleaned iris data into a DataFrame:

```
# Loading data
file_path = "Resources/new_iris_data.csv"
df_iris = pd.read_csv(file_path)
df_iris.head(10)
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1

Initialize the K Starting Centroids

After data has been loaded, create an instance of the K-means algorithm and initialize it with the desired number of clusters (*K*).

IMPORTANT

> We're working with data that has a set number of clusters. Often, you won't know the number that you should work with, so you'll have to use the trial-and-error method to determine it. In the next section, we'll learn an approach that can help with the trial-and-error method.

For this example, we know that there are three different classes of iris plants, so we'll use `K = 3`:

```
# Initializing model with K = 3 (since we already know there are three classes of iris plants)
model = KMeans(n_clusters=3, random_state=5)
model
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=5, tol=0.0001, verbose=0)
```

```
# Initializing model with K = 3 (since we already know there are three class
model = KMeans(n_clusters=3, random_state=5)
model
```

Data Points Assigned to Nearest Centroid

Once the model instance is created, our next step is to fit the model with the unlabeled data. This step should be familiar with fitting data from supervised learning; however, you'll notice that data is not being split into training and test data. When the model is being trained (fit the data), the K-means algorithm will iteratively look for the best centroid for each of the *K* clusters:

```
# Fitting model
model.fit(df_iris)
```

Group Data Points

After the model is fit, the corresponding cluster for every iris plant in the dataset can be found using the `predict()` method:

```
# Get predictions
predictions = model.predict(df_iris)
print(predictions)
```

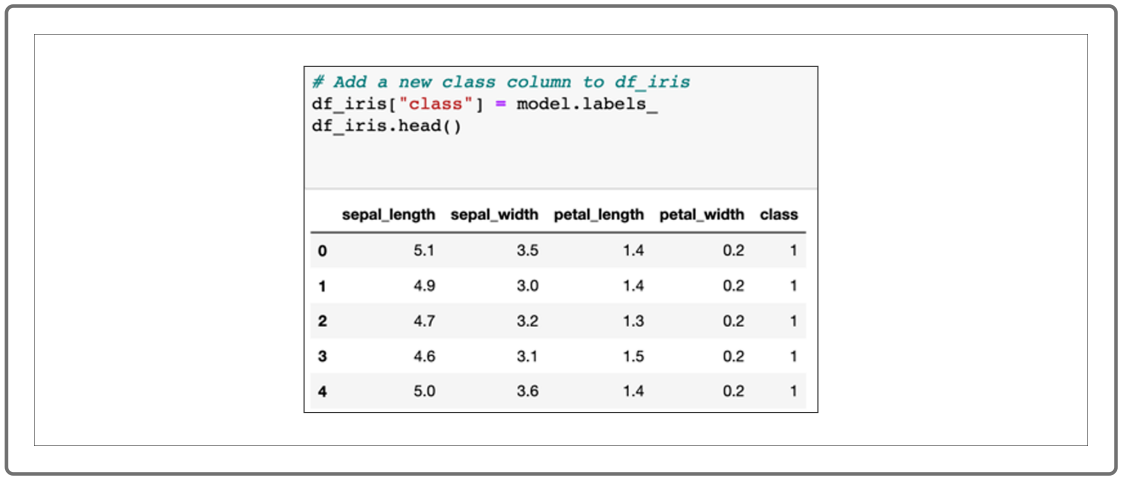
```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 2 0 2 0 2 0 2 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 2 0 2 2 2
 2 0]
```

```
# Get the predictions
predictions = model.predict(df_iris)
print(predictions)
```

IMPORTANT

> As you can see, there were three subclasses that were labeled 0, 1, and 2. These are **not** the means for the centroids, but rather just the label names. The actual naming of the classes is part of the job by a subject matter expert, or whoever performs the analysis, such as yourself. The K-means algorithm is able to identify how many clusters are in the data and label them with numbers.

After we have the class for each data point, we can add a new column to the DataFrame with the predicted classes:



	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1

```
# Add a new class column to the df_iris
df_iris["class"] = model.labels_
df_iris.head()
```

Visualize the Results

Visualizing the clusters helps to graphically understand how they are arranged. In this case, we actually have too many features to represent visually, but we can select a few of them and plot the clusters.

For our visualizations, we'll use hvPlot, a graphing library that allows deeper exploration of the data.

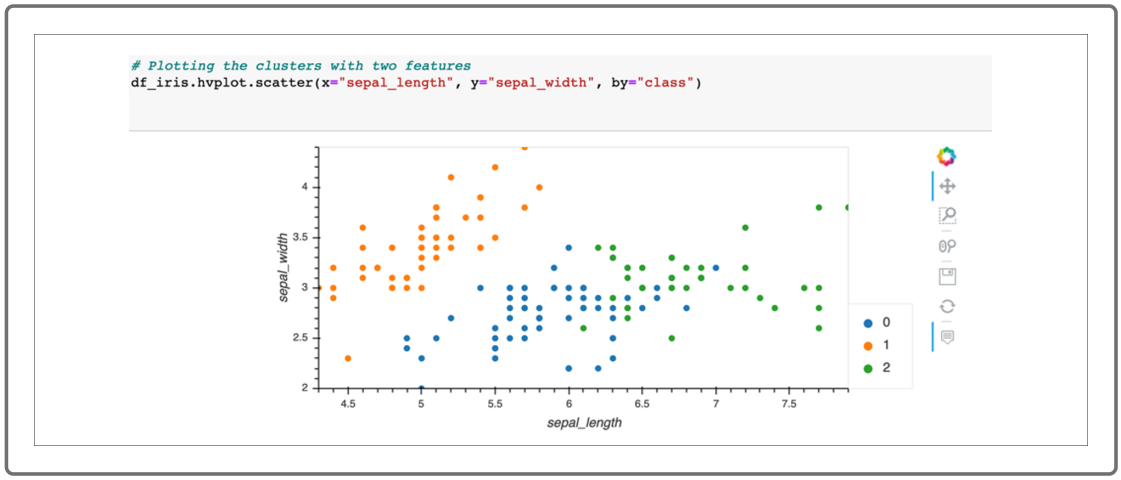
NOTE

> We won't dive too much into the library, aside from scatter plots. Still, if you'd like to learn more about hvPlot, visit the official **website (https://hvplot.holoviz.org/)** for the documentation.

We'll import the following dependencies for `hvplot`, as shown below:

```
import plotly.express as px
import hvplot.pandas
```

First, look at the data with two features. The hvPlot library makes it easy to create scatter plots directly from a Pandas DataFrame. After our DataFrame has been loaded in from the CSV, we can create a scatter plot with one line of code. We pass in the arguments for the x- and y-axis and color them by class:

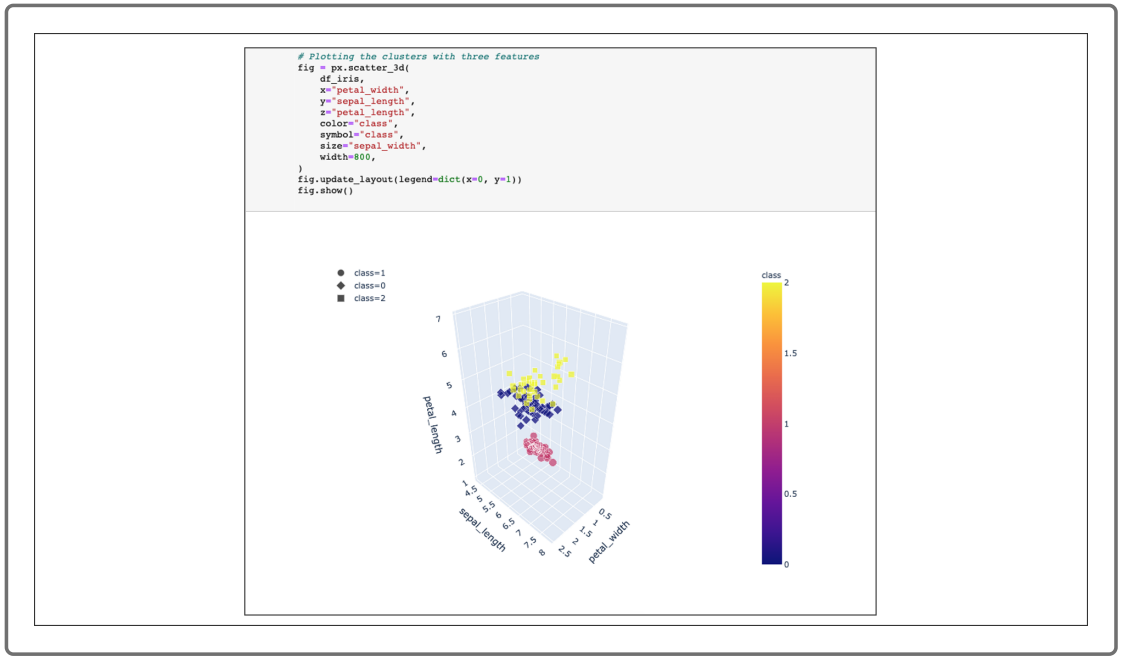



```
# Add a new class column to the df_iris
df_iris.hvplot.scatter(x="sepal_length", y="sepal_width", by="class")
```

In the results, it appears some of the clusters are overlapping and not quite forming three distincts groups as we had hoped. Before jumping to the conclusion that our model didn't do what we wanted, remember that we are taking multiple data points (petal_width, sepal_length, and petal_length). Since this plot is on a 2D graph, all three features can't be properly displayed.

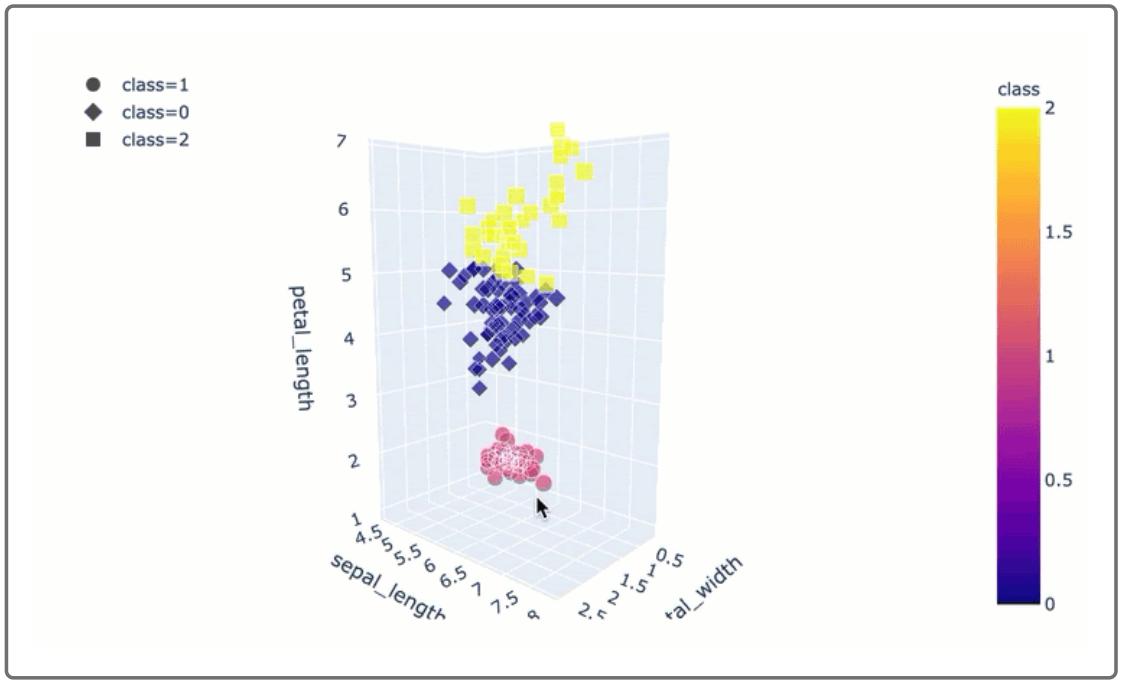
Plotting in 3D takes a few more arguments and will allow us to visualize more data points. We now have an x-, y-, and z-axis that will take all three of our features as coordinates. We pass in the class data points to determine color and symbol of the points. Size of the points will be determined by sepal_width.

Finally, we'll update the figure by passing a dictionary with x, y, and z:




```
# Plotting the clusters with three features
fig = px.scatter_3d(df_iris, x="petal_width", y="sepal_length", z="petal_len
fig.update_layout(legend=dict(x=0,y=1))
fig.show()
```

The 3D scatter plot can be rotated using the mouse to click and drag and panned using the scroll wheel.



Here, you can see that our model did do what we wanted! There are now three distinct groups that correspond to the three clusters that we expect the model to break the data into.

18.3.3 Trial and Error of Finding Centroids

Working with the iris dataset, you knew there would be three clusters to match with three types of irises. However, you and Martha realize that you have no idea where to begin with the cryptocurrency dataset. The first approach you decide to take is testing out the cluster amounts through trial and error.

So far, the value of K we have used was known ahead of time. We knew the amount of classes that were contained in the dataset, and so we set the value appropriately. This will usually not be the case, and the decision will need to be made by looking at the data with a bit of trial and error.

To test by trial and error with clusters, we'll use a sample of the shopping dataset that contains customer data.

Start by opening a new notebook and enter the code to import the libraries we'll need to use:

```
# Initial imports
import pandas as pd
from sklearn.cluster import KMeans
```

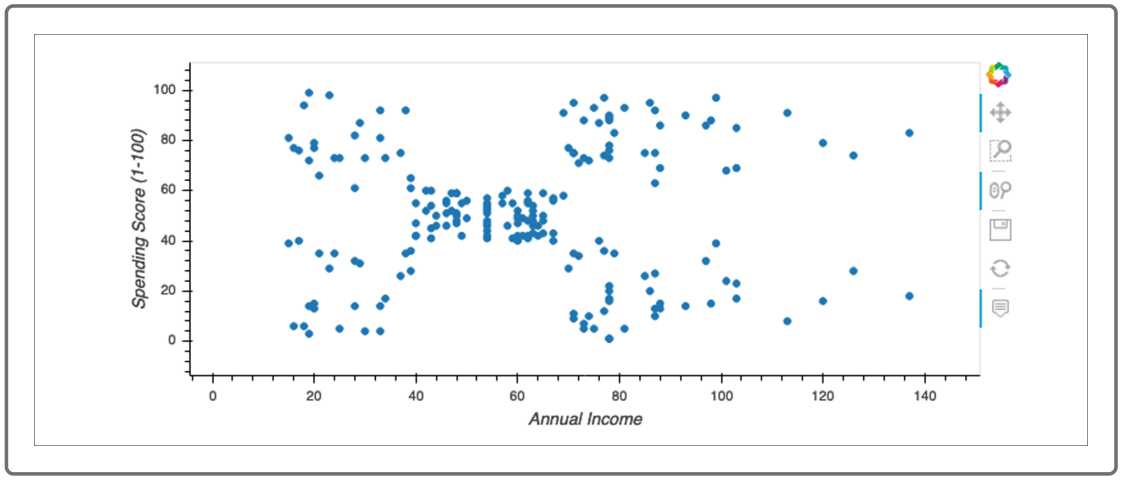
```
import plotly.express as px
import hvplot.pandas
```

Then enter the code to load in the dataset into a DataFrame:

```
# Load data
file_path = "Resources/shopping_data_cleaned.csv"
df_shopping = pd.read_csv(file_path)
df_shopping.head(10)
```

See what the points look like at the start by entering the code:

```
df_shopping.hvplot.scatter(x="Annual Income", y="Spending Score (1-100)")
```



On first look, it may seem obvious the amount of clusters that would work, but let's see what happens when we start to cluster.

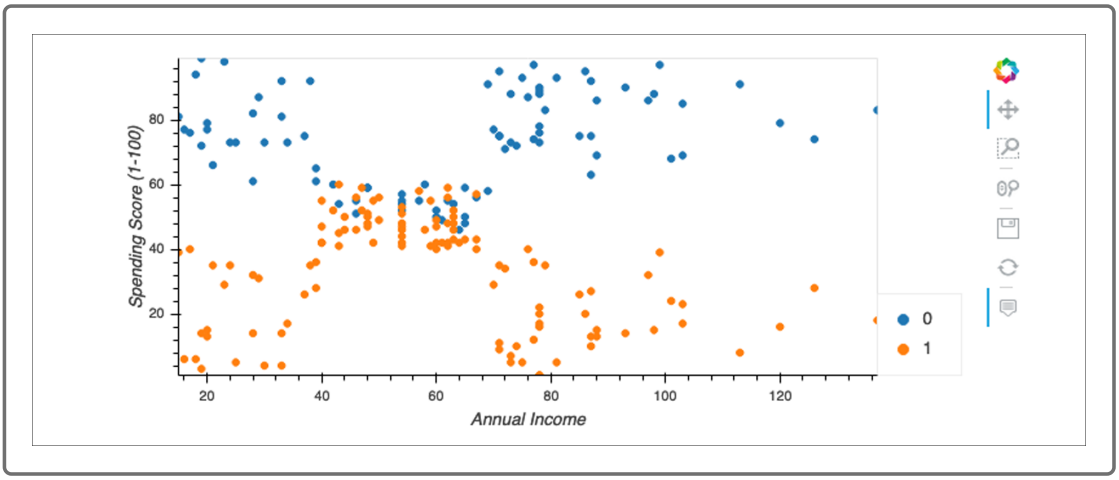
First, let's create a function so we can quickly run K-means on the DataFrame with a different amount of clusters by entering the following code:

```
# Function to cluster and plot dataset
def test_cluster_amount(df, clusters):
    model = KMeans(n_clusters=clusters, random_state=5)    model
    # Fitting model
    model.fit(df)

# Add a new class column to df_iris
df["class"] = model.labels_
```

This function will take a DataFrame and the number of clusters to make as arguments. Start by running the function to create two clusters and then plot the results:

```
test_cluster_amount(df_shopping, 2)
df_shopping.hvplot.scatter(x="Annual Income", y="Spending Score (1-100)", by
```

The graph will appear as follows:



At first glance, two clusters look okay with some data points mixed in the middle.

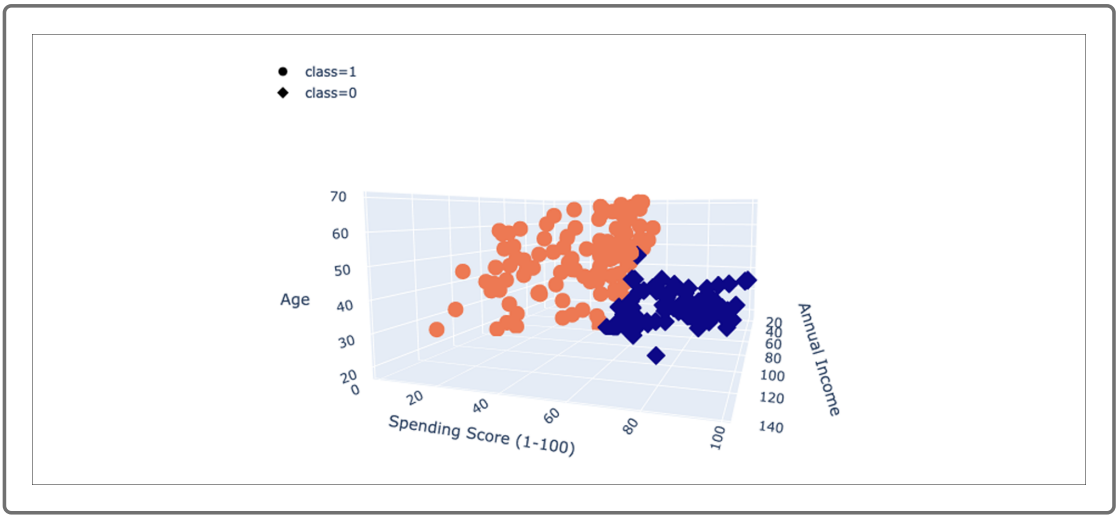
NOTE

> Recall that sometimes plotting data with more than two data points in a 2D plot might show the true clustering.

Since there are some data points in the middle, let's plot the DataFrame with a third axis. Enter the code to create a 3D plot:

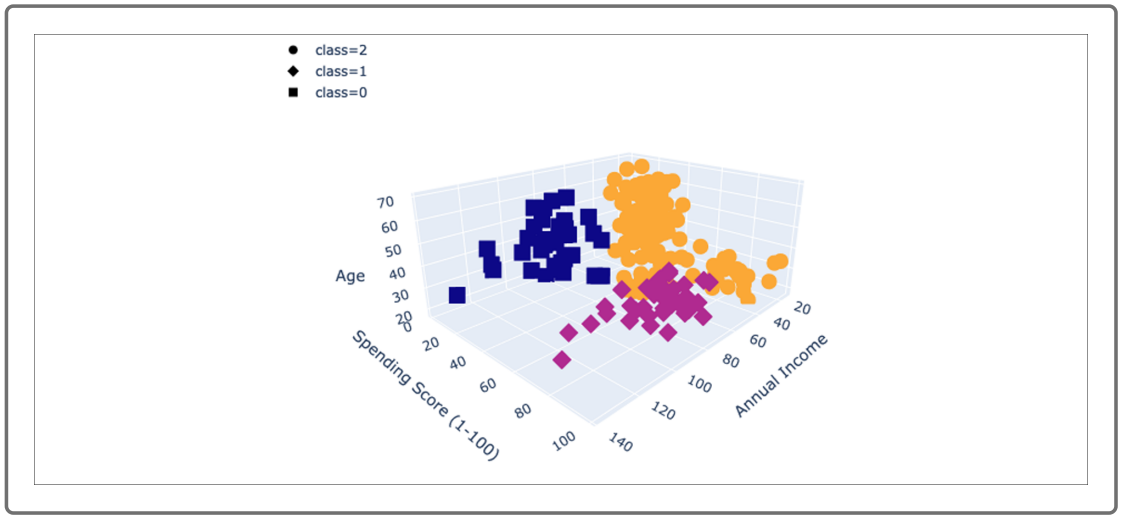
```
fig = px.scatter_3d(
    df_shopping,
x="Annual Income",
    y="Spending Score (1-100)",
    z="Age",
color="class",
    symbol="class",
    width=800,
)
fig.update_layout(legend=dict(x=0, y=1))
fig.show()
```

The results plot now looks as follows:

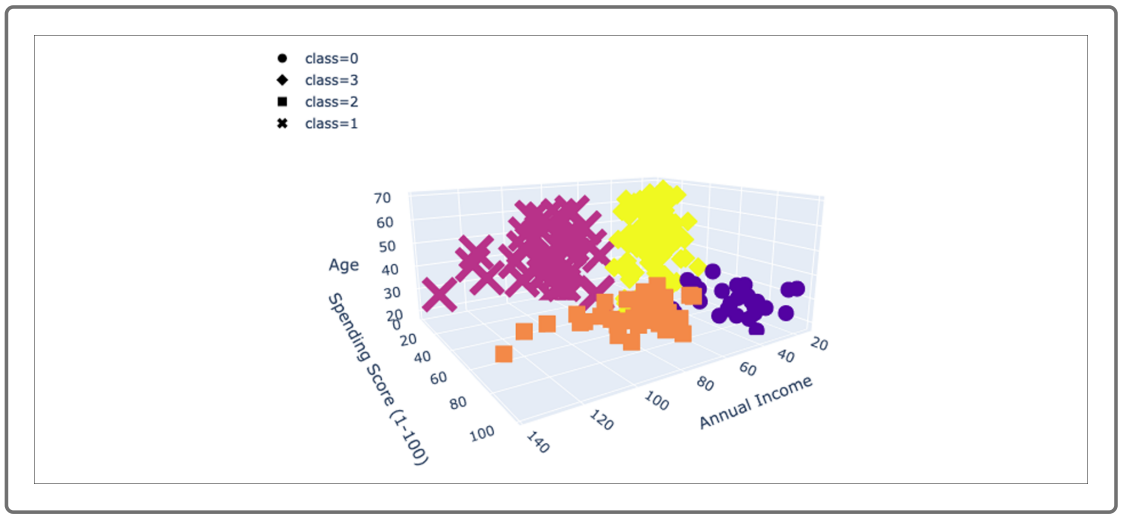


With the 3D plot, the cluster looks much better. Let's repeat the process a few more times and see what the different clusters look like.

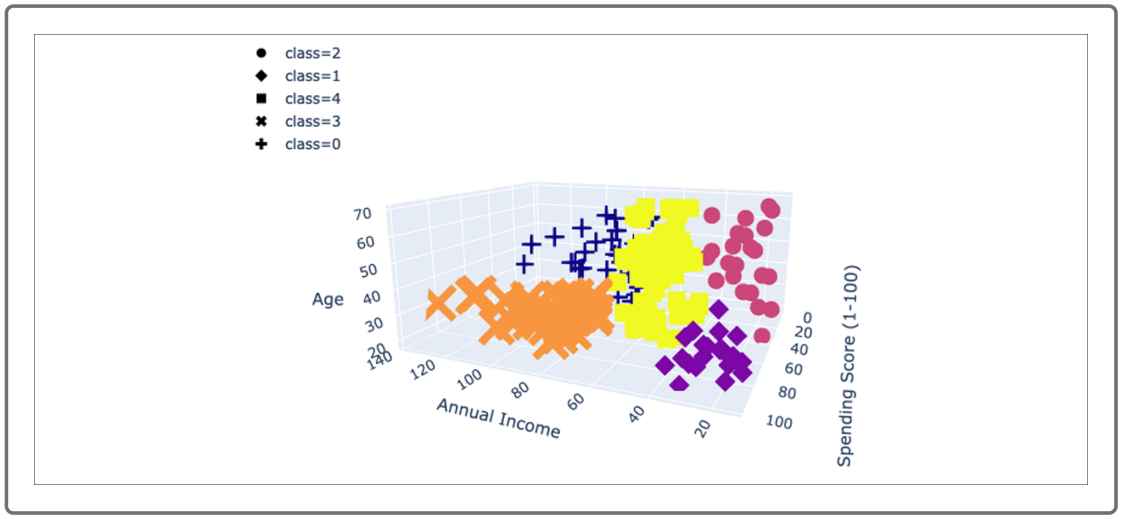
Three clusters appear like so:



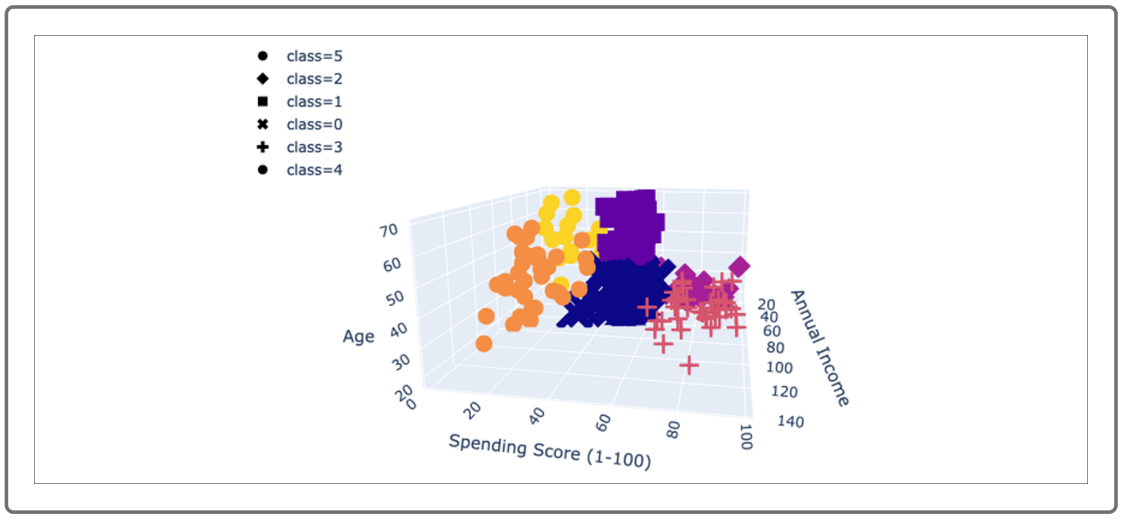
Four clusters appear like so:



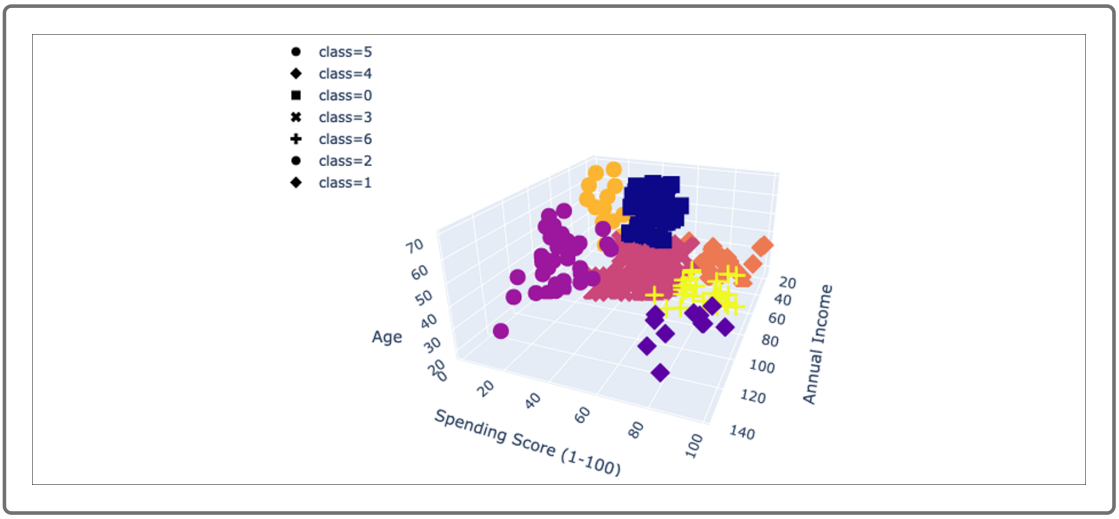
Five clusters appear like so:



Six clusters appear like so:



You might be thinking that each time we added more clusters, the graph looked better, so let's really go for it and split the data into seven clusters, like so:



This also looks great! We're really starting to see some solid clusters break out. However, before we get trigger-happy and increase the clusters further, we should consider when there might be too many clusters.

If we have too many, will it even tell us something about the data? If we increase to 100 clusters, that would really fine-tune each group, but with so many clusters, can we even do anything with that?

Recall that unsupervised learning doesn't have a concrete outcome like supervised learning does. We use unsupervised learning to parse data to help us make decisions. So, at what point do we lose the helpfulness of unsupervised learning?

With trial and error, this can become unclear and can only get us so far with more complex datasets. In the next section, we'll learn a method that will help us determine the best value for *K* when clustering data.

18.4.1 Elbow Curve

The trial-and-error method seemed to work to an extent, but the two of you wonder what happens when data gets more complex, such as in the cryptocurrency dataset.

There is a method that will help you make a more educated guess on the amount of clusters called the elbow curve.

An easy method for determining the best number for K is the elbow curve. Elbow curves get their names from their shape: they turn on a specific value, which looks a bit like an elbow!

To create an elbow curve, we'll plot the clusters on the x-axis and the values of a selected objective function on the y-axis.

Inertia is one of the most common objective functions to use when creating an elbow curve. While what it's actually doing can get into some pretty complicated math, basically the inertia objective function is measuring the amount of variation in the dataset.

So, for our elbow curve, we'll plot the number of clusters (also known as the values of K) on the x-axis and the inertia values on the y-axis.

Let's see what happens when we plot our K values versus inertia for the preprocessed iris dataset created earlier.

We will first take a look at the elbow curve using this dataset, since we know that there should be three clusters.

Let's first take a look at the elbow curve using the iris dataset, since we know that there should be three clusters:

```
# Initial imports
import pandas as pd
from sklearn.cluster import KMeans
import plotly.express as px
import hvplot.pandas
```

Then enter the code to load in the dataset into a DataFrame:

```
# Loading data
file_path = "Resources/new_iris_data.csv.csv"
df_iris = pd.read_csv(file_path)
df_iris.head(10)
```

Store Values of *K* to Plot

We'll start with creating an empty list to hold inertia values. We'll also store a range of *K* values we want to test. Enter the code in a new cell:

```
inertia = []
k = list(range(1, 11))
```

Loop Through *K* Values and Find Inertia

Next, we'll loop through each *K* value, find the inertia, and store it into our list. Enter the code in the next cell:

```
# Looking for the best K
for i in k:
km = KMeans(n_clusters=i, random_state=0)
    km.fit(df_iris)
inertia.append(km.inertia_)
```

Create a DataFrame and Plot the Elbow Curve

We'll create a DataFrame that stores our *K* values and their appropriate inertia values. This will allow for an easy plot of the results with `hvplot`. In another new cell, enter the code:

```
# Define a DataFrame to plot the Elbow Curve using hvPlot
elbow_data = {"k": k, "inertia": inertia}
df_elbow = pd.DataFrame(elbow_data)
df_elbow.hvplot.line(x="k", y="inertia", title="Elbow Curve", xticks=k)
```

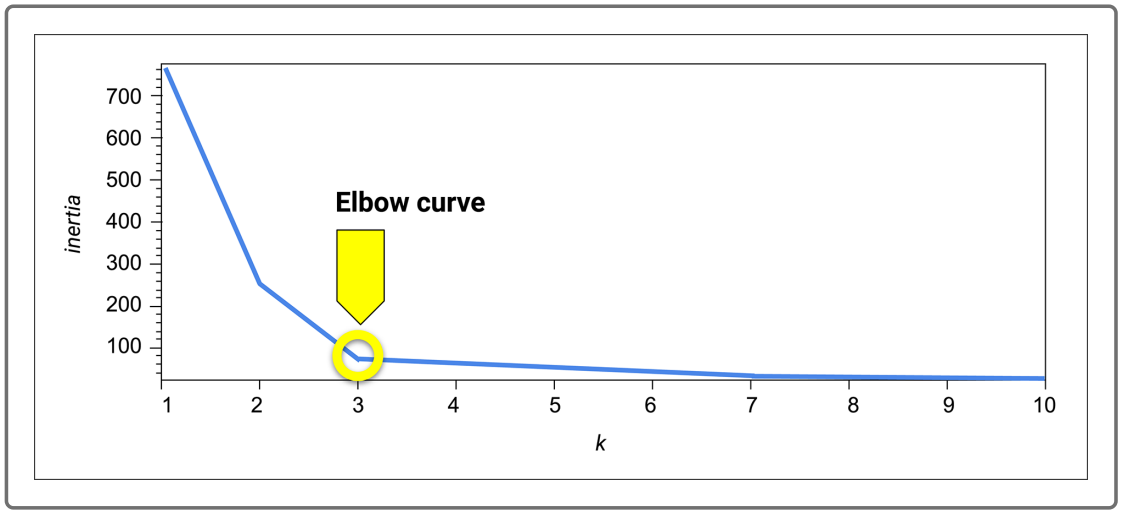
This will create a graph.

Use the Elbow Curve to Determine the Best *K* Value

Let's take a look at the graph.

Note the shape of the curve on the following graph. At point 0 (top left), the line starts as a steep vertical slope that breaks at point 2, shifts to a

slightly horizontal slope, breaks again at point 3, then shifts to a strong horizontal line that reaches to point 10. The angle at point 3 looks like an elbow, which gives this type of curve its name:



18.4.2 Use the Elbow Curve

Now that you and Martha know how to create an elbow curve, use one on a dataset to help determine the number of clusters you should use.

Let's walk through an example of how to use the elbow curve. This time, we'll answer the question from the previous section on customer data and how many clusters would be ideal.

Open a new notebook and import our dependencies:

```
# Initial imports
import pandas as pd
from sklearn.cluster import KMeans
import plotly.express as px
import hvplot.pandas
```

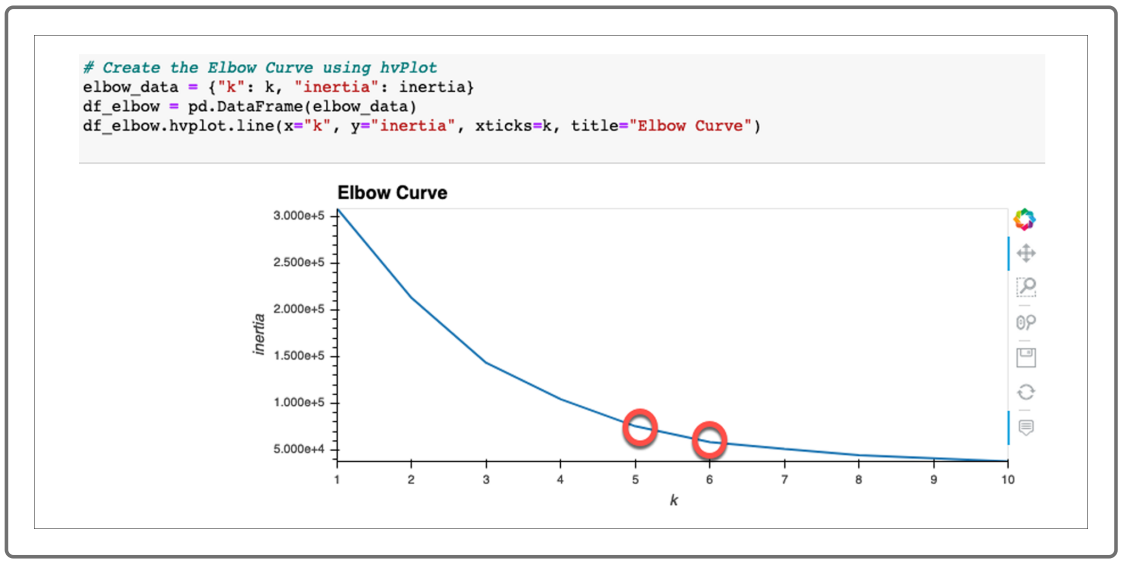
Then enter the code to load in the dataset into a DataFrame:

```
# Load data
file_path = "Resources/shopping_data_cleaned.csv"
df_shopping = pd.read_csv(file_path)
df_shopping.head(10)
```

To create the elbow curve, remember there are two values we need: a list of *K* values and a list of inertia values. Recall that inertia is the objective function to plot *K* values against. We will loop through 10 values for *K* and determine the inertia:

```
inertia = []
k = list(range(1, 11))
# Calculate the inertia for the range of K values
for i in k:
    km = KMeans(n_clusters=i, random_state=0)
    km.fit(df_shopping)
    inertia.append(km.inertia_)
```

Next, let's create a plot for the elbow curve:



This elbow curve doesn't have as obvious of an elbow as previously seen. Either K values for points 5 or 6 could be considered the elbow. There is no

surefire way to pick between the two, but we did knock down the potential number of K values between points from 10 to 2. You might also wonder why point 3 wasn't considered. Remember, we're looking for the break where the vertical direction shifts to a strong horizontal direction. Compared to points 5 and 6, the shift at point 3 isn't as dramatic.

Before plotting the two K values, let's create a K-means function again to reuse the K-means cluster. As you may recall, functions allow us to save time because we don't need to write the code contained in the function more than once:

```
def get_clusters(k, data):   # Create a copy of the DataFrame   data = data.
```

NOTE

> Creating a function is not required for K-means. The `get_clusters` function helps us save time since we'll run the algorithm twice: once with point 5 and again with point 6. If you're still struggling with functions, feel free to run the code twice, but do revisit using `get_clusters` after strengthening your Python function skills.
>
> Consider looking into the principle of **"Don't repeat yourself" (https://en.wikipedia.org/wiki/Don%27t_repeat_yourself)** to learn why it's important to use functions.

We can now run the function for K = 5:

```
five_clusters = get_clusters(5, df_shopping)
five_clusters.head()
```

	Card Member	Age	Annual Income	Spending Score (1-100)	class
0	1	19	15	39	0
1	1	21	15	81	4
2	0	20	16	6	0
3	0	23	16	77	4
4	0	31	17	40	0

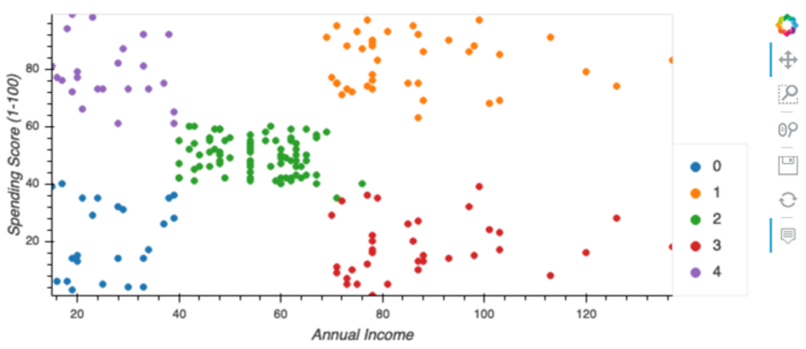
Run the function again for K = 6:

```
six_clusters = get_clusters(6, df_shopping)
six_clusters.head()
```

	Genre	Age	Annual Income	Spending Score (1-100)	class
0	1	19	15.0	39	5
1	1	21	15.0	81	3
2	0	20	16.0	6	5
3	0	23	16.0	77	3
4	0	31	17.0	40	5

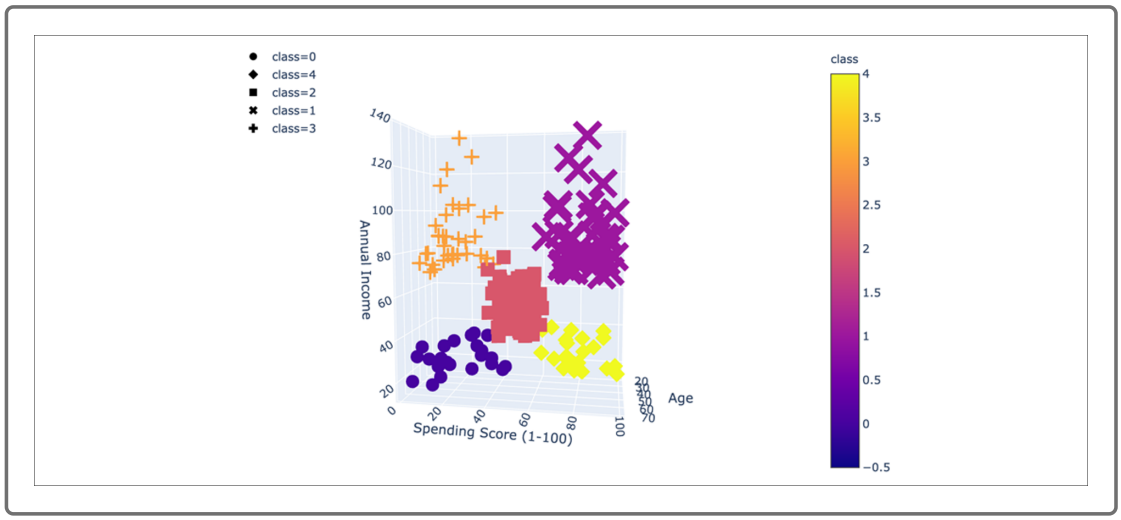
Plot a 2D graph for K = 5:

```
# Plotting the 2D-Scatter with x="Annual Income" and y="Spending Score (1-100)"
five_clusters.hvplot.scatter(x="Annual Income", y="Spending Score (1-100)", by="class")
```



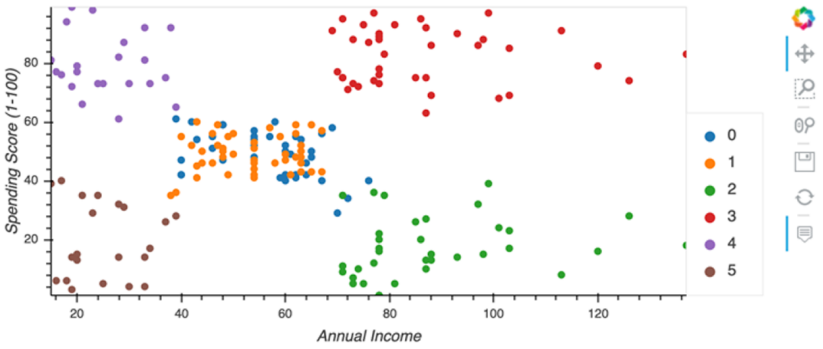
Plot a 3D graph for K = 5:

```
# Plot the 3D-scatter with x="Annual Income", y="Spending Score (1-100)" and
fig = px.scatter_3d(
    five_clusters,
    x="Age",
    y="Spending Score (1-100)",
    z="Annual Income",
    color="class",
    symbol="class",
    width=800,
)
fig.update_layout(legend=dict(x=0, y=1))
fig.show()
```



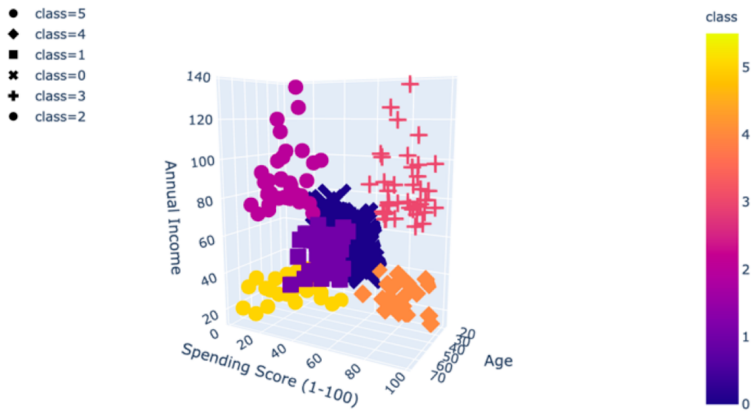
Plot a 2D graph for K = 6:

```
# Plotting the 2D-Scatter with x="Annual Income" and y="Spending Score (1-100)"
six_clusters.hvplot.scatter(x="Annual Income", y="Spending Score (1-100)", by="class")
```



Plot a 3D graph for K = 6:

```
# Plotting the 3D-Scatter with x="Annual Income", y="Spending Score (1-100)"
fig = px.scatter_3d(
    six_clusters,
    x="Age",
    y="Spending Score (1-100)",
    z="Annual Income",
    color="class",
    symbol="class",
    width=800,
)
fig.update_layout(legend=dict(x=0, y=1))
fig.show()
```



Recall, in the trial-and-error method, both graphs displayed multiple clusters. We're still applying some trial and error here, but the elbow curve helps narrow down the number of clusters.

Now, the important question: So do we use five or six groups? This depends on what insights you can take away from the data. One might conclude that six groups would be most useful because they could be broken down like so:

- Cluster 0: medium income, low annual spend
- Cluster 1: low income, low annual spend
- Cluster 2: high income, low annual spend
- Cluster 3: low income, high annual spend
- Cluster 4: medium income, high annual spend
- Cluster 5: very high income, high annual spend

If we choose five groups, they would need to be different and would not fit into what you're looking for, which is grouping types of customers based on spending habits. Remember, unsupervised learning can help us make decisions about the data, up to a point, then it is up to you, the expert, to make the final call.

So far, you've learned that when dealing with multiple features, the clusters were best viewed in 3D graphs, which can get messy. In the next section, we'll learn how to limit or combine features.

18.5.1 Dimensionality Reduction

Martha has noticed that so far we have been working with pretty good datasets in terms of data used. Even after some data cleanup, there haven't been too many features to work with. However, she is beginning to worry that her cryptocurrency data has too many features and is not sure how this will affect our model. The way to handle this is with dimensionality reduction.

Think back to our example with the store owner who is trying to sell school supplies. His customer data could contain endless features, or columns. The data could include name, age, address, items bought, amount spent, time spent shopping, zip code, and so forth. Some features just aren't necessary and could throw off our algorithm. For instance, would converting names to an integer value be worth the time or even inform our analysis?

Also, throwing all of these features into the model might overfit the data.

Why is overfitting a model bad?

- It makes the model too specific and unable to work with additional datasets.
- The model won't take into consideration all of the data points.
- Overfitting only works for supervised learning.
- Make the model too broad and it won't weigh data points evenly.

Check Answer

Finish ▸

Since overfitting is bad, it is best to find a way to limit features. The process of reducing features is called dimensionality reduction. There are two options for coping with too many features: elimination and extraction.

Feature Elimination

Your first idea is to remove a good amount of features so the model won't be run using every column. This is called **feature elimination**.

Feature elimination means what you think: You remove, or eliminate, a feature from the dataset. In our school supply example, you remove features that aren't relevant to what we're looking for, such as name, address, and zip code. This simple method increases and maintains interpretability.

The downside is, once you remove that feature, you can no longer glean information from it. If we want to know the likelihood of people buying school supplies, but we removed the zip code feature, then we'd miss a detail that could help us understand when certain residents tend to purchase school supplies.

Feature Extraction

Feature extraction combines all features into a new set that is ordered by how well they predict our original variable.

In other words, feature extraction reduces the number of dimensions by transforming a large set of variables into a smaller one. This smaller set of variables contains most of the important information from the original large set.

NOTE

> Sometimes, you need to use both feature elimination and extraction. For instance, the customer name feature doesn't inform us about whether or not customers will purchase school supplies. So, we would eliminate that feature during the preprocessing stage, then apply extraction on the remaining features.

18.5.2 Principal Component Analysis

Your client assured you that all the data they have collected is important and needs to be used. Being worried about overfitting your data, you decided to use Principal Component Analysis (PCA).

PCA is a statistical technique to speed up machine learning algorithms when the number of input features (or dimensions) is too high. PCA reduces the number of dimensions by transforming a large set of variables into a smaller one that contains most of the information in the original large set.

PCA is a complicated process to understand, but it is easy to code. Let's start out by coding some PCA into our K-means, so that you can see it in action, then revisit the code's underlying theory.

Using the **new_iris_data.csv (https://courses.bootcampspot.com/courses/138/files/26357/download?wrap=1)**  **(https://courses.bootcampspot.com/courses/138/files/26357/download?wrap=1)** first, import the libraries we'll use and load the data into a Pandas DataFrame:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import hvplot.pandas
```

```
# Loading the preprocessed iris dataset
file_path = "Resources/new_iris_data.csv"
df_iris = pd.read_csv(file_path)
df_iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

There are four features in this dataset with values on different scales. The first step in PCA is to standardize these features by using the StandardScaler library:

```
# Standardize data with StandardScaler
iris_scaled = StandardScaler().fit_transform(df_iris)
print(iris_scaled[0:5])
```

```
[[-0.90068117  1.03205722 -1.3412724  -1.31297673]
 [-1.14301691 -0.1249576  -1.3412724  -1.31297673]
 [-1.38535265  0.33784833 -1.39813811 -1.31297673]
 [-1.50652052  0.10644536 -1.2844067  -1.31297673]
 [-1.02184904  1.26346019 -1.3412724  -1.31297673]]
```

Now that the data has been standardized, we can use PCA to reduce the number of features. The PCA method takes an argument of `n_components`, which will pass in the value of 2, thus reducing the features from 4 to 2:

```
# Initialize PCA model
pca = PCA(n_components=2)
```

After creating the PCA model, we apply dimensionality reduction on the scaled dataset:

```
# Get two principal components for the iris data.
iris_pca = pca.fit_transform(iris_scaled)
```

After this dimensionality reduction, we get a smaller set of dimensions called principal components. These new components are just the two main dimensions of variation that contain most of the information in the original dataset.

The resulting principal components are transformed into a DataFrame to fit K-means:

```
# Transform PCA data to a DataFrame
df_iris_pca = pd.DataFrame(
    data=iris_pca, columns=["principal component 1", "principal component 2"]
)
df_iris_pca.head()
```

	principal component 1	principal component 2
0	-2.264542	0.505704
1	-2.086426	-0.655405
2	-2.367950	-0.318477
3	-2.304197	-0.575368
4	-2.388777	0.674767

Use `explained_variance_ratio` to learn how much information can be attributed to each principal component:

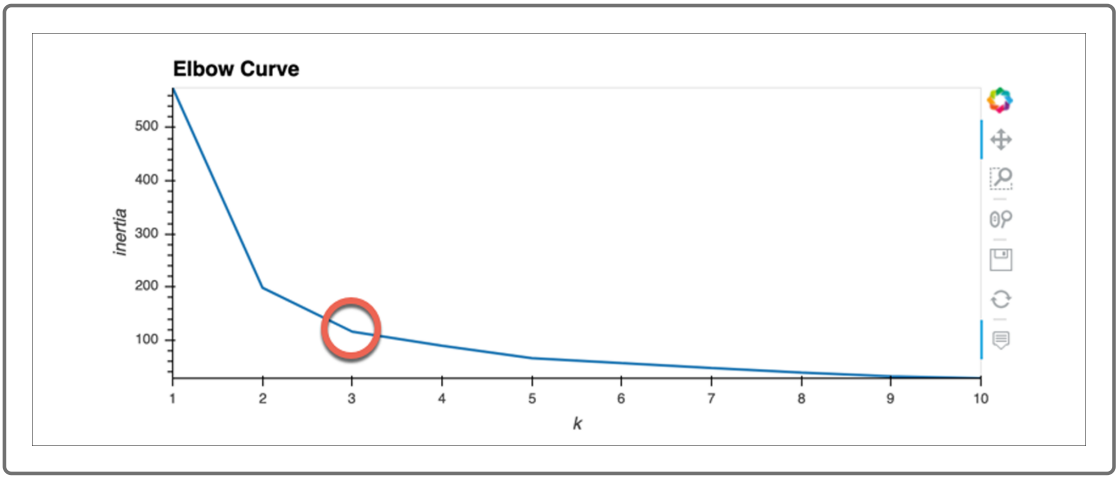
```
# Fetch the explained variance
pca.explained_variance_ratio_
```

```
array([0.72770452, 0.23030523])
```

What this tells us, is that the first principal component contains 72.77% of the variance and the second contains 23.03%. Together, they contain 95.80% of the information.

Next, we'll use the elbow curve with the generated principal components and see the *K* value is 3:

```
# Find the best value for K
inertia = []
k = list(range(1, 11))

# Calculate the inertia for the range of K values
for i in k:
    km = KMeans(n_clusters=i, random_state=0)
    km.fit(df_iris_pca)
    inertia.append(km.inertia_)

# Create the elbow curve
elbow_data = {"k": k, "inertia": inertia}
df_elbow = pd.DataFrame(elbow_data)
df_elbow.hvplot.line(x="k", y="inertia", xticks=k, title="Elbow Curve")
```

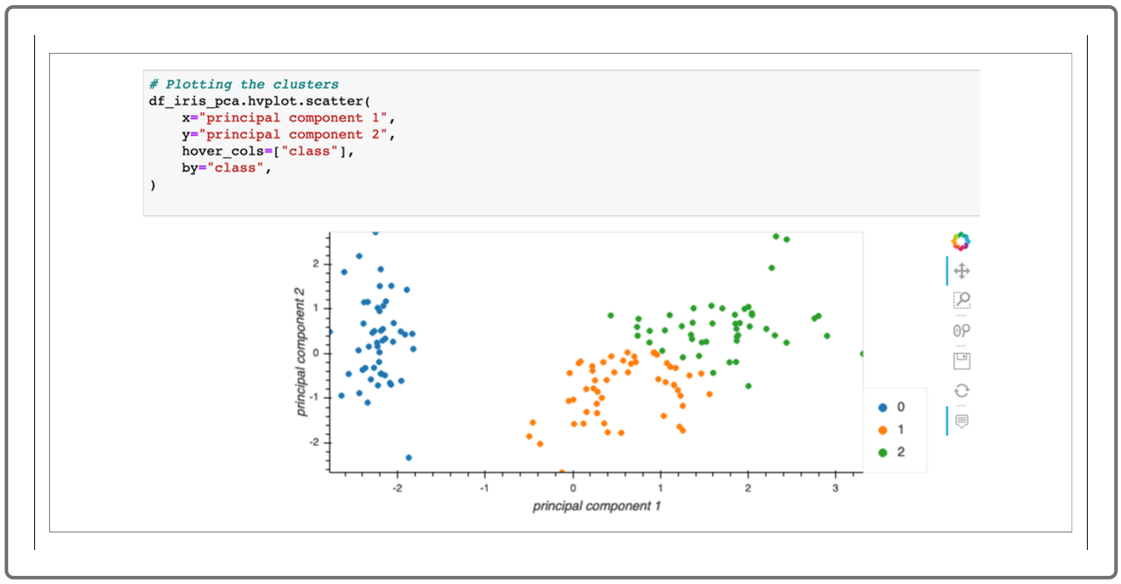


Use the principal components data with the K-means algorithm with a K value of 3. We could consider 2, but the direction shifts more after 3:

```
# Initialize the K-means model
model = KMeans(n_clusters=3, random_state=0)

# Fit the model
model.fit(df_iris_pca)

# Predict clusters
predictions = model.predict(df_iris_pca)

# Add the predicted class columns
df_iris_pca["class"] = model.labels_
df_iris_pca.head()
```

Finally, we can plot the clusters. Instead of a 3D plot, the data is easier to analyze with only two features:

```
df_iris_pca.hvplot.scatter(
    x="principal component 1",
    y="principal component 2",
    hover_cols=["class"],
    by="class",
)
```



NOTE

> The next few sections will go over exactly how PCA works and can be a bit daunting. Remember, you can already code it!

18.5.3 Mean, Variance, and Covariance

Now that you convinced Martha that feature extraction is the way to go, she needs some background on why this works in case questions come up during her presentation on how she can "magically" combine these features in a meaningful way. To start, you dust off your stats knowledge and refresh your memory on mean, variance, and covariance. These will be the building blocks used for PCA.

There is a mathematical way to use feature extraction, but first let's review some stats concepts.

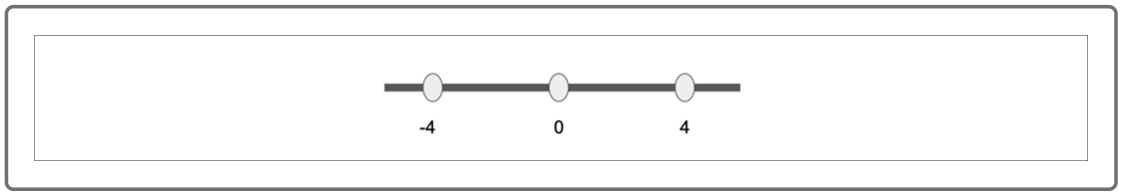
Mean

Recall that the **mean** is the sum of a group of numbers divided by the total amount of numbers. For example, we start with points 2, 3, and 7. First, we add up all the numbers: 2 + 3 + 7 = 12. Then we divide the result by the total amount of points, which is 3. So, 12 / 3 = 4, so the mean of those three points is 4.

Variance

Variance is the square distance from each point from the center, added together, and divided by the total number of points. The variance measures the spread of a set of numbers. The center of the points may look familiar, and it should, because it is the mean of all the points. Variance, in other words, is a measure of how far apart the data points are from the mean.

Look at the following points on a line:



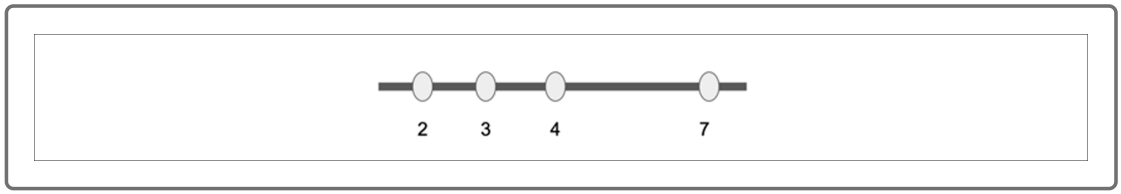
Using 0 as the center point, the distances are -4 from the center, 0 from the center (the center point is still a point), and 4 from the center.

The sum of squared distances would be (-4)^2 + (0)^2 + (4)^2 = 16 + 0 + 16 = 32. We use squared distance so they are all positive.

Divide by the total number of points, which is 3. The variance of this dataset would be 32/3, or 10⅔.

Normally, there won't be an even distribution of points around the center. The points 2, 3, and 7 from the previous example don't have a clear center.

This is where the mean comes into play. The center of the line is set to the mean, which we found to be 4. Here is what the points look like on a line:



The distance from 4 to 2 is -2, the distance from 4 to 3 is -1, the distance from 4 to 4 is 0, and the distance from 4 to 7 is 3.

Add up the squares of each distance: (-2)^2 + (-1)^2 + (0)^2 + (3)^2 = 4 + 1 + 0 + 9 = 14.

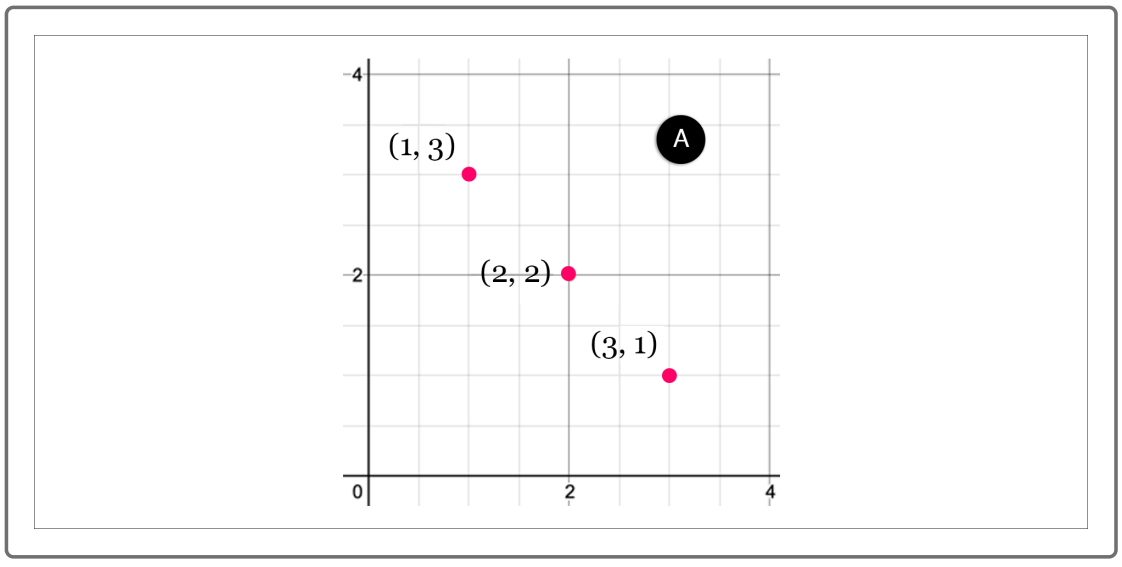
Finally, divide the distances by the total number of points: 14 / 3. The variance equals 14/3, or 4⅔.

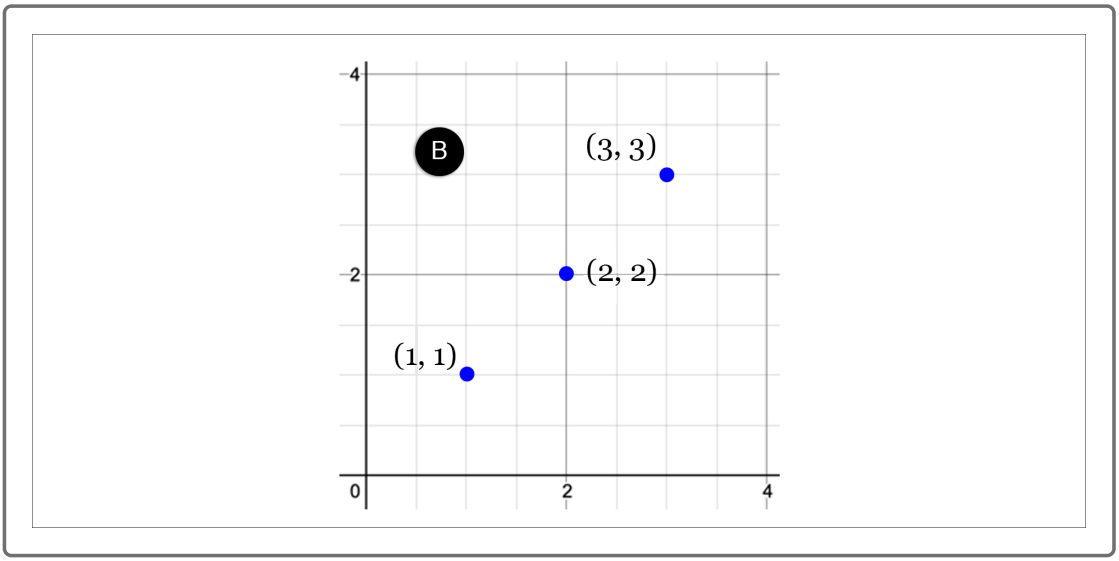
NOTE

> These examples showed points on the x-axis, and thus, form the x variance. The same process applies to elements on the y-axis, forming the y variance.

Covariance

Before defining what covariance is, look at the following two plots:





These two plots clearly are very different. Each has the same center, with different points on the left and the right, one sloping negatively and the other sloping positively.

Let's find the x and y variance for each line.

For graph A:

- The center point is (2, 2).
- The distances for the points are the distance from (2, 2).
- Point (1, 3) is a distance of -1 away on the x-axis and 1 on the y-axis.
- Point (3, 1) is a distance of 1 away on the x-axis and -1 on the y-axis.
- x variance = (-1)^2 + 0^2 + (1)^2 = 2 / 3
- y variance = (1)^2 + 0^2 + (-1)^2 = 2 / 3

For graph B:

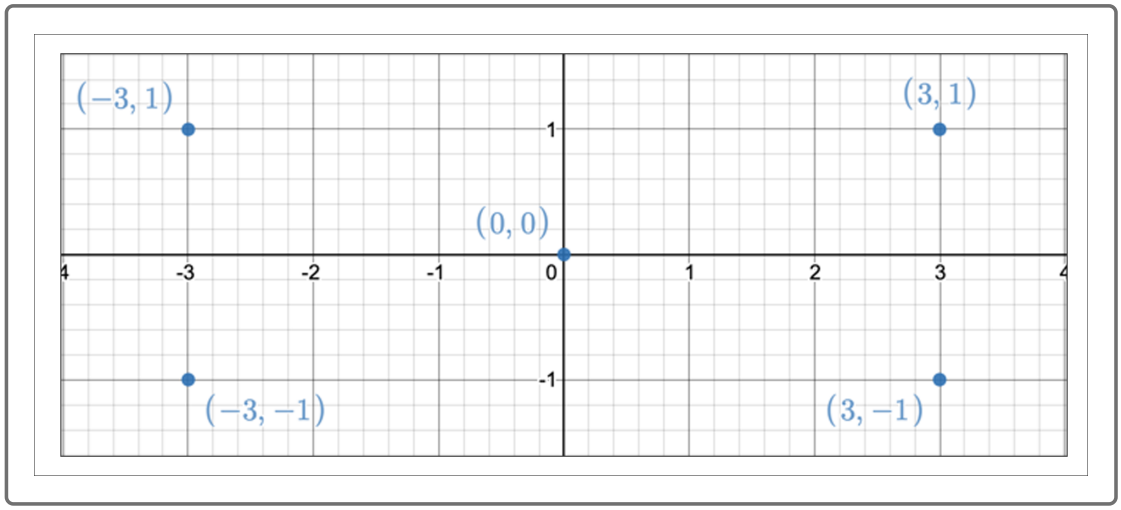
- The center point is (2, 2).
- The distances for the points are the distance from (2, 2).
- Point (1, 1) is a distance of -1 away on the x-axis and -1 on the y-axis.
- Point (3, 3) is a distance of 1 away on the x-axis and 1 on the y-axis.
- x variance = (-1)^2 + 0^2 + (1)^2 = 2 / 3

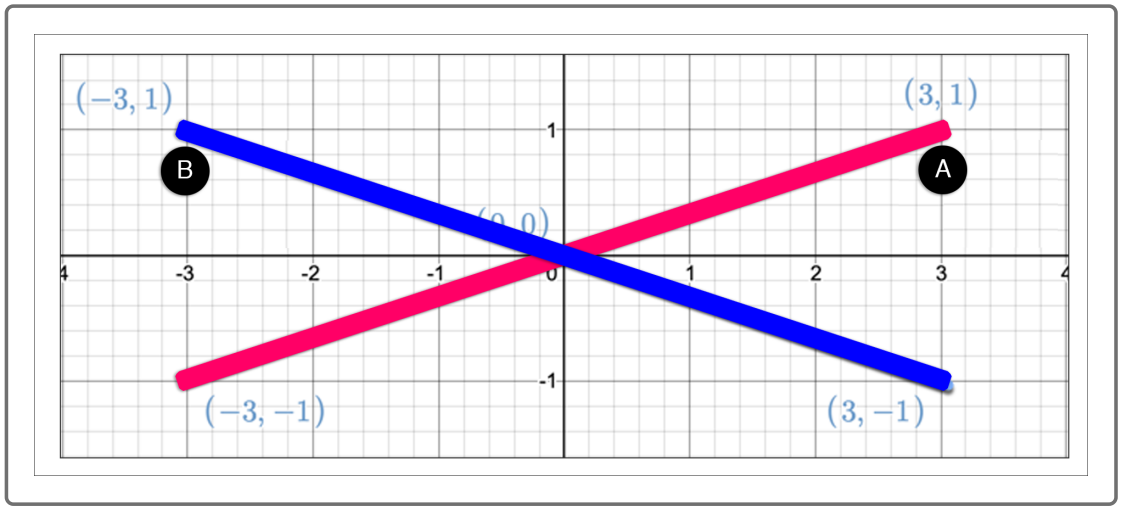
- y variance = (-1)^2 + 0^2 + (1)^2 = 2 / 3

Wait. Both of these variances are exactly the same; however, it is very obvious that these two graphs are totally different! How can we tell the difference?

This is where covariance comes into play. **Covariance** is a metric that allows us to tell these two different sets of points apart.

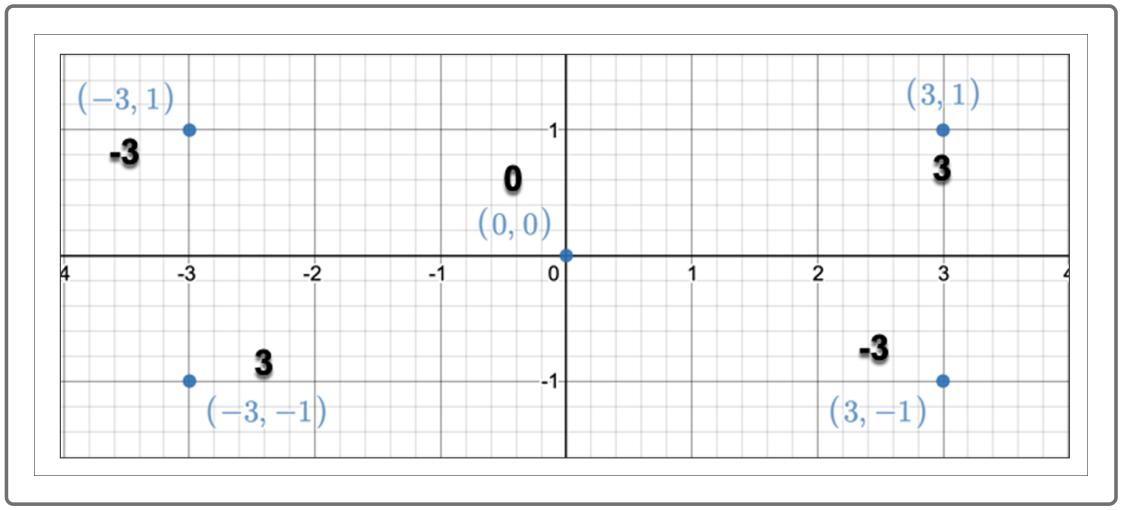
Let's look at the following examples:





How can we tell the difference between the points that lie along Line A versus the points that lie along Line B?

We can do this with the product of coordinates, which is the multiple of each of the two points:



Covariance is the sum of the product of coordinates divided by the number of points.

Covariance is used to determine the relationship between points.

The formula for covariance is as follows:

$$Cov\,(x,y) = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\mathbb{N}}$$

What this equation is saying is that the covariance takes the sum of the product between each pair of coordinates and their difference from the mean divided by the total number of points. This may sound complicated but will make more sense once we look at an example.

Let's solve for the covariance of line A first which contains the points (-3, -1), (0, 0) and (3, 1).

First take the mean of the x coordinates in line A, -3 + 0 + 3 = 0 divided by 3 is zero. Then repeat for the y coordinates, -1 + 0 + 1 = 0 divided by 3 is also zero.

Then for each pair of coordinates find the difference between the point and their respective means.

X	Y	$(x_i - \bar{x})$	$(y_i - \bar{y})$
-3	-1	-3 - 0 = -3	-1 - 0 = -1
0	0	0 - 0 = 0	0 - 0 = 0
3	1	3 - 0 = 3	1 - 0 = 1

Now multiply the results of the coordinate pairs.

X	Y	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
-3	-1	-3 - 0 = -3	1 - 0 = -1	3
0	0	0 - 0 = 0	0 - 0 = 0	0
3	1	3 - 0 = 3	1 - 0 = 1	3

Finally add the product of all the coordinated paris and divide by the number of points to find the covariance.

$$3 + 0 + 3 = 6$$

Plug the results into the top part of the equation, and since we know there or 3 points, we plug that in for N to get.

$$Cov(x, y) = \frac{6}{3}$$

Reduce the equation.

$$Cov(x, y) = 2$$

The covariance for line A is 2.

Repeat the same process for line B would produce the following:

X	Y	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$

-3	1	-3 - 0 = -3	1 - 0 = 1	-3
0	0	0 - 0 = 0	0 - 0 = 0	0
3	-1	3 - 0 = 3	1 - 0 = -1	-3

Add the product of all the coordinated pairs.

$$-3 + 0 + -3 = -6$$

Plug the results into the top part of the equation, and again we know there are 3 points, we plug that in for N to get.

$$Cov(x, y) = -\frac{6}{3}$$

Reduce the equation.

$$Cov(x, y) = -2$$

The covariance for line A is -2.

The covariance for Line A is 2 while the covariance for Line B is -2.

We can then say that Line A has a positive covariance (at 2) while Line B has a negative covariance (at -2). There is also a third type of covariance called **covariance zero**. This is when the points tend to form a horizontal line.

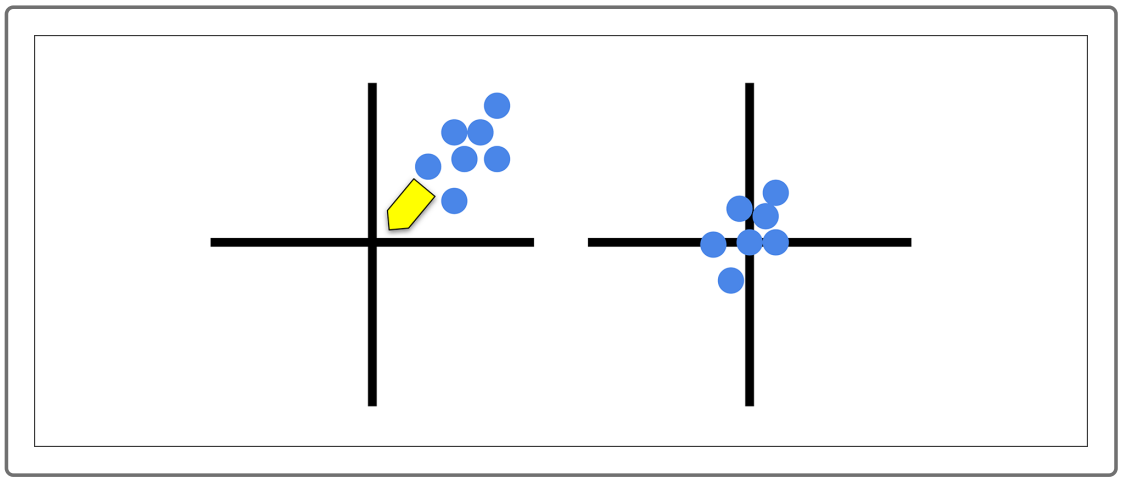
NOTE

> Covariance is used to only describe the relationship between points, such as positive and negative as we just saw. You may recall another method for determining relationships is correlation. However, correlation is used to determine the strength of the relationship.

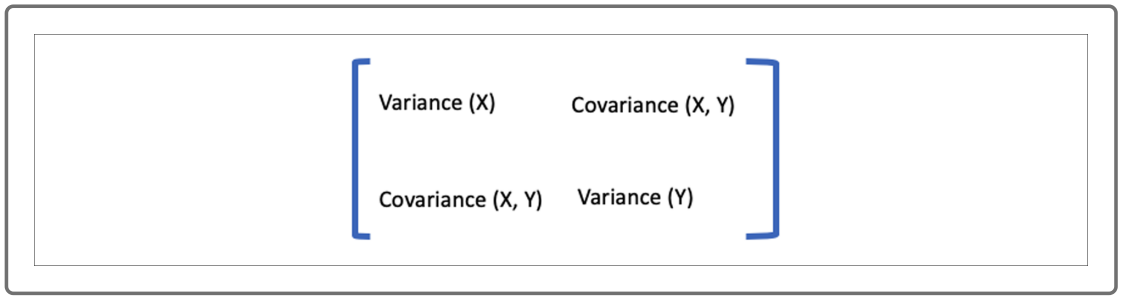
18.5.4 Linear Transformations

Martha appreciates the refreshers on stats but is wondering where this is going. Well, patience is a virtue, and trust that all of this forms the building blocks to really start to understand how PCA works. Next up is **linear transformations**.

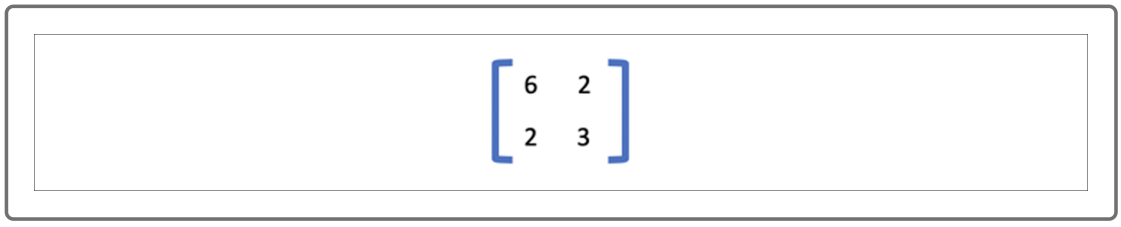
Say we have a set of points on a graph. We want to center these points by taking the average of the coordinate, both X and Y. Find the balance point and move that to zero:



Once the points are centered, we're going to create a 2x2 matrix that consists of the variance and covariances that we found in the previous step:



So, let's say the matrix above contains the following:



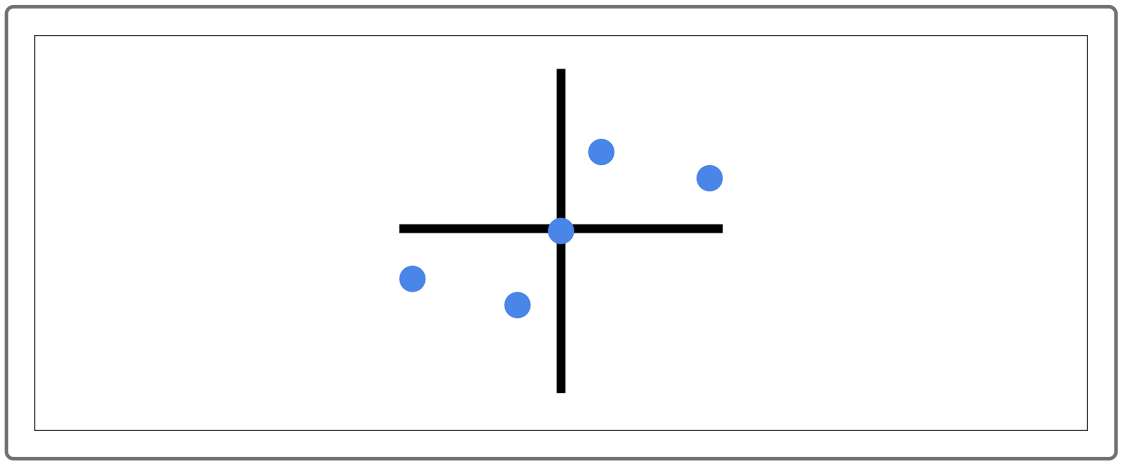
This matrix will be used to transform the points from one graph to another by using the numbers to create a formula for our transformation. The top two values of the matrix will correspond to one point and the bottom two values to another.

In our example, the formula for the points becomes (6x + 2y, 2x, + 3y). Let's plug some coordinates into the formula:

(x, y)	**(6x + 2y, 2x + 3y)**
(0,0)	(0,0)
(1,0)	(6,2)
(0,1)	(2,3)
(-1,0)	(-6,-2)

(0,-1)	(-2,-3)

Now, let's plot the new points from the right side of the matrix to create a linear transformation:

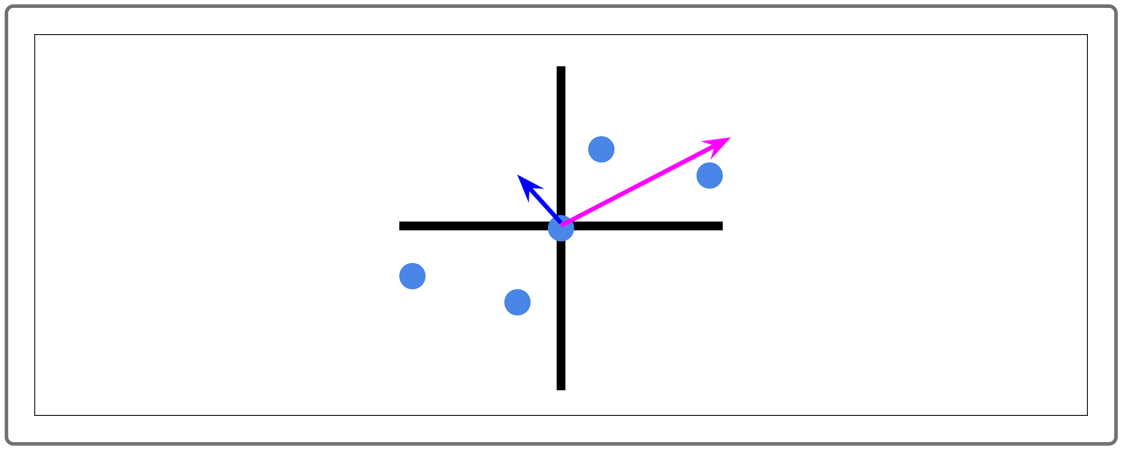


Eigenvectors and Eigenvalues

NOTE

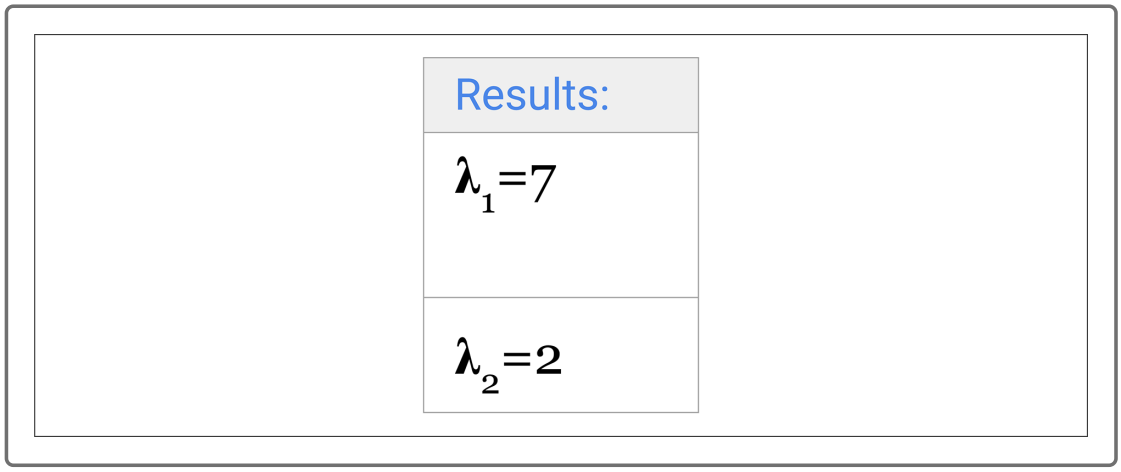
> Eigenvectors and eigenvalues can be complicated subjects rooted in linear algebra. We cover these at a very high level, but if you wish to explore more on your own, you can read more about **Eigenvalues and eigenvectors** (https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors) and watch this **video** (https://www.youtube.com/watch?v=PFDu9oVAE-g)

As you can see, the points stretch out in our graph in two directions. One direction moves from southwest to northeast direction while another direction moves from northwest to southeast. These are called **eigenvectors**, as indicated by the arrows in the graph below:



There is a way to figure out the vectors and values with algebra, but we use the calculator on **WolframAlpha (https://www.wolframalpha.com/input/?i=eigenvalues)** to simplify the process. Plug in our matrix of {{6,2},{2,3}}, then click calculate.

From the results website, you can see in one direction the shape stretched to a value of 7 and another to a value of 2. The magnitude that each of these stretches is called the **eigenvalue**:



We also see the direction that stretched with the eigenvectors of (2, 1) and (-1, 2):

Corresponding eigenvectors:
$v_1 = (2, 1)$
$v_2 = (-1, 2)$

The big takeaway from eigenvectors and eigenvalues is that they show us the spread of the dataset and by how much.

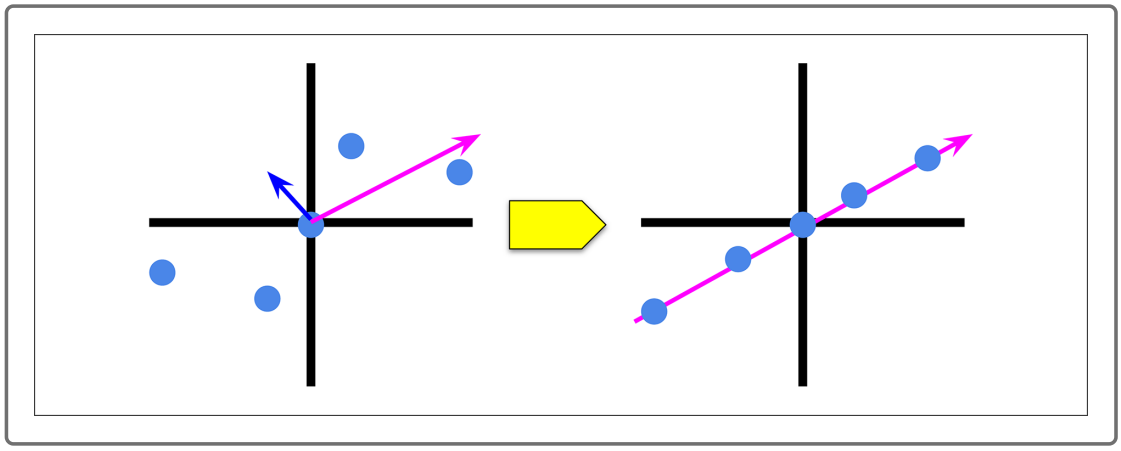
18.5.5 PCA's Underlying Theory

With all the math and fancy words to describe, let's finally see how all this comes together with PCA.

Now it's time to put everything together and show how PCA works. Given our two eigenvalues from before, 7 and 2, take the greater eigenvalue, 7, and eliminate the other since it's less important. The higher eigenvalue is the axis that carries the most amount of information.

We'll also take the corresponding eigenvector, which is (2,1).

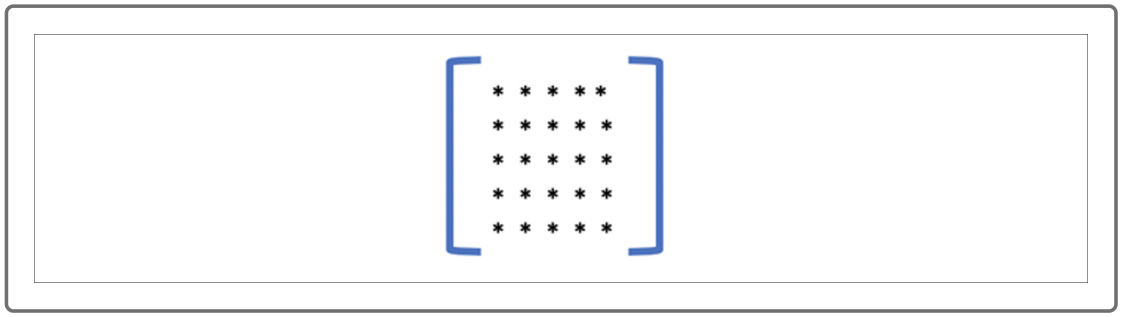
Next, extend that eigenvector with the higher value to a line and project all our points onto that line:



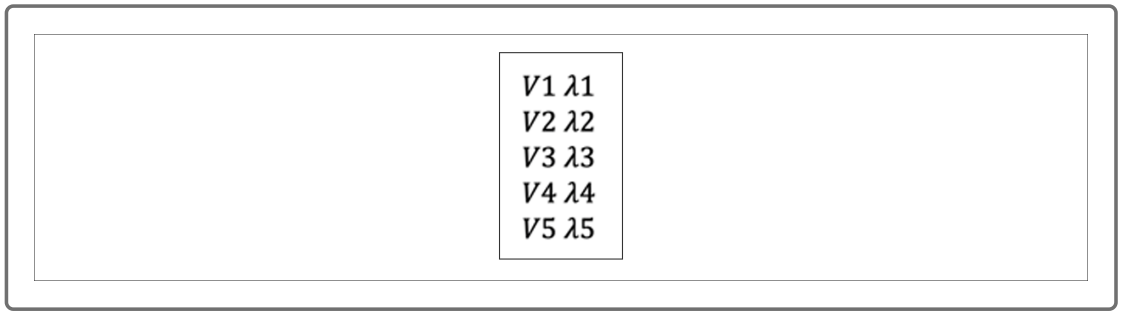
Now let's put everything together and show what PCA is doing. We'll up the ante a little bit and expand from two to five columns of data. First, take our data that consists of five columns, or features. Note, the asterisk (*) will represent a number as we'll avoid using numbers to simplify the exercise:

x1	x2	x3	x4	x5
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*

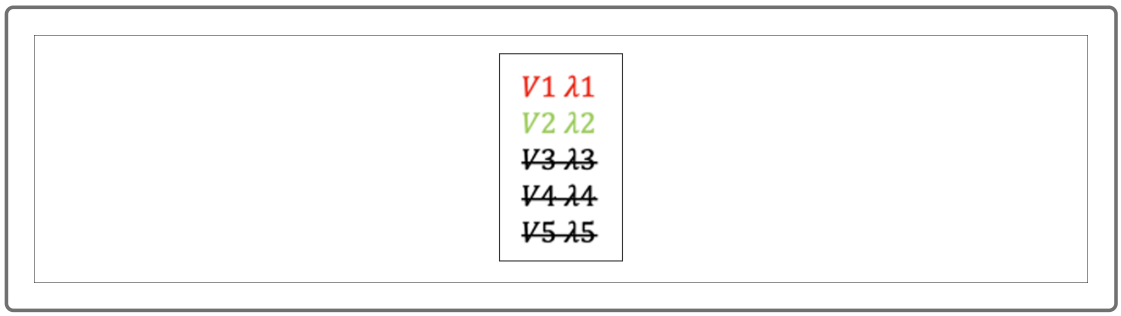
Put all the data points into a 5x5 covariance matrix. The eigenvectors and eigenvalues are calculated for each of those five columns in the matrix. Again, the asterisk (*) represents a number:



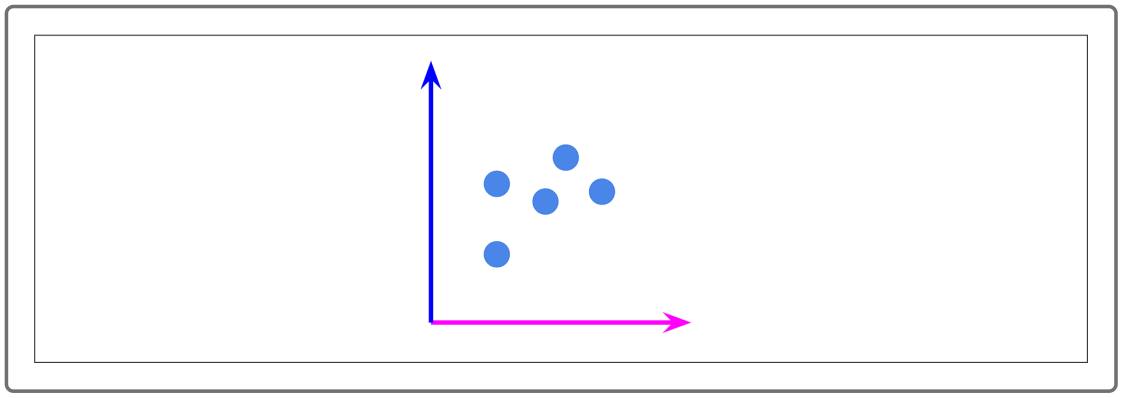
From the matrix we can produce a list of eigenvectors and corresponding eigenvalues:



We pick how many eigenvalues we want to keep and which to drop. For this example, we'll keep the top two eigenvalues and drop three:



Taking two will allow us to plot on a 2D plane. The two eigenvalues and eigenvectors will create a plane on which all the points can be plotted:



This now narrows down our five features to two and gives us a good snapshot of what the data should look like because we chose the

directions the data spread the most.

Finally, these data points will give us a table of two columns, where the asterisk (*) is a number. Remember, when we coded PCA, the end result was two columns of principal components:

Principal Component 1	Principal Component 2
*	*
*	*
*	*
*	*

IMPORTANT

> The statistics, linear transformations, and eigenvalues and eigenvectors all illustrate how PCA works. As you saw earlier, it is much easier to code than do all of this math. So, don't worry if this is confusing—remember, you've already coded it! It is important to understand, on some level, what PCA is doing in case you're ever asked in an interview.

That wraps up how PCA works. (Wow! That was a lot of fancy jargon!) Thankfully, code has made our work easier and now you have a better understanding of how to reduce dimensions yet still keep the values.

Next, look at another form of clustering.

18.6.1 Understanding Hierarchical Clustering

You both covered a lot of ground today with unsupervised learning, and Martha now feels ready to complete her project for her client. She does have one last question: Is K-means the only sort of clustering algorithm? There is another type, hierarchical clustering.

Similar to K-means clustering, hierarchical clustering, also known as agglomerative clustering, works with groups (clusters) of data points. The algorithm starts by declaring each point with its own cluster, then merges the two most similar clusters until a declared stopping point has been reached. This stopping point is implemented within your code.

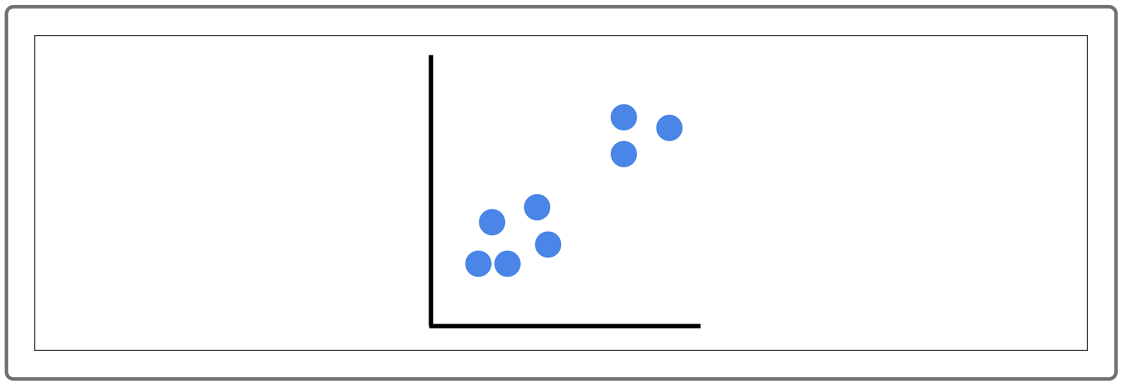
Hierarchical clustering has three methods for determining how points are linked: ward, average, and complete.

Ward is the algorithm's default setting. Simply put, this function selects the two clusters that, when merged, will mean the least amount of variance between all remaining clusters. This often leads to clusters that are relatively equal in size.

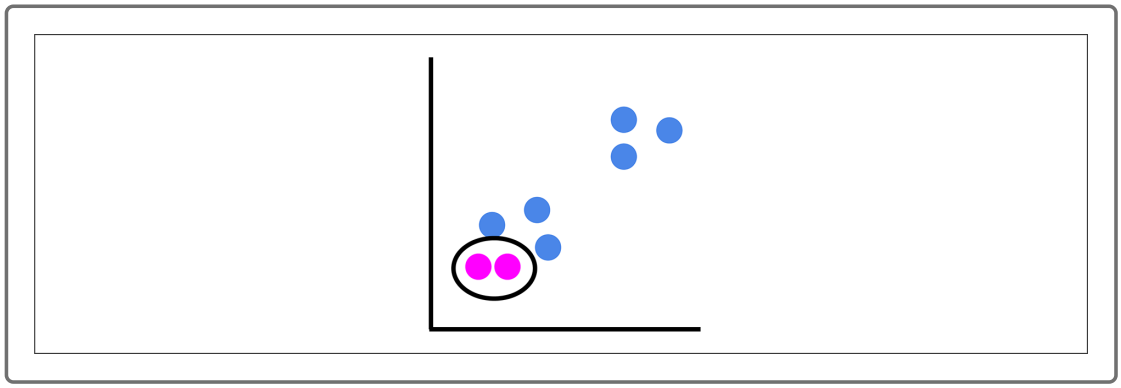
You can also link points using the **average**. This method connects clusters that have the smallest average distance between all of their points, then connects clusters on the smallest average distance between all their points.

The third method is **complete**, which links by merging clusters that have the smallest maximum distance.

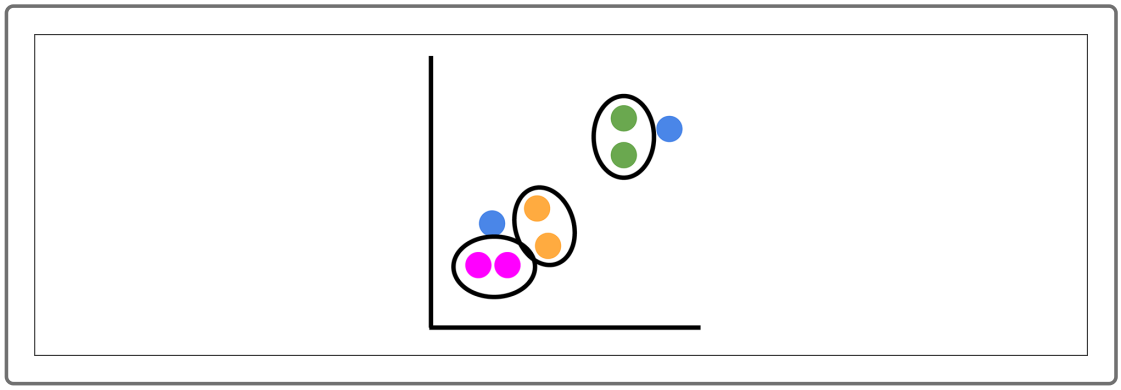
For example, say we're given the following points and our stopping point is set at two clusters:



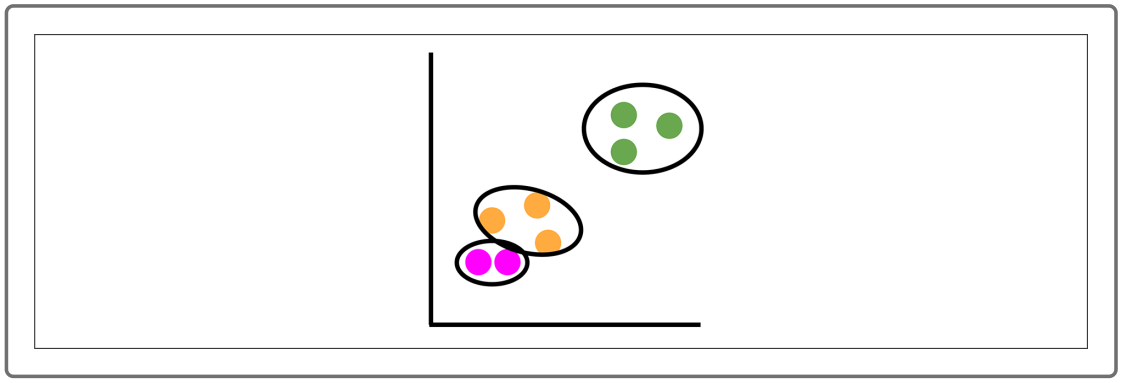
Each point, at the start, is considered a part of its own cluster, so we can say our starting point is eight clusters. We'll start by grouping together the two closest points:



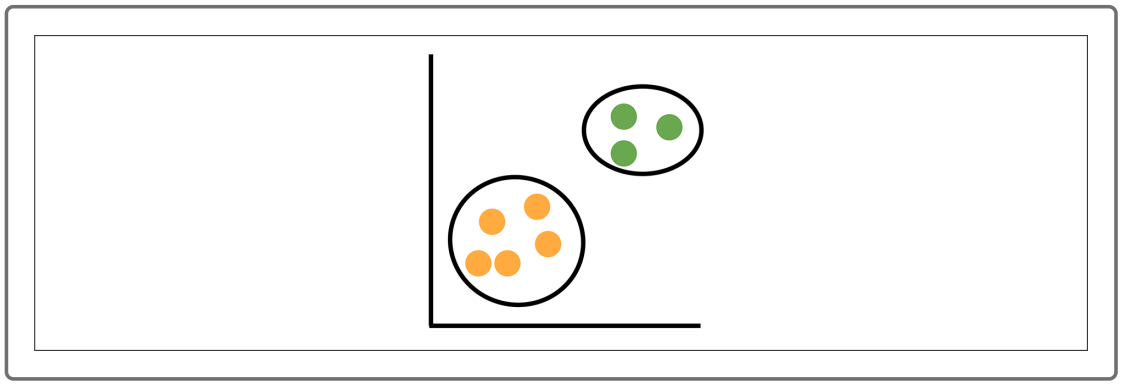
We now have seven clusters, and our stopping point of two clusters has not been met, so we keep going. The next two closest points are then grouped together, and we continue with this process:



Now, the next closest points contain a point in a cluster. When this happens, we join that point to the closest clusters. Remember, each point is considered a cluster:



The next two closest points to each other are now both contained in a cluster, since the stopping point of two clusters has still not been met, these clusters are merged:



We have now reached the stopping point of two clusters, and we just completed hierarchical clustering.

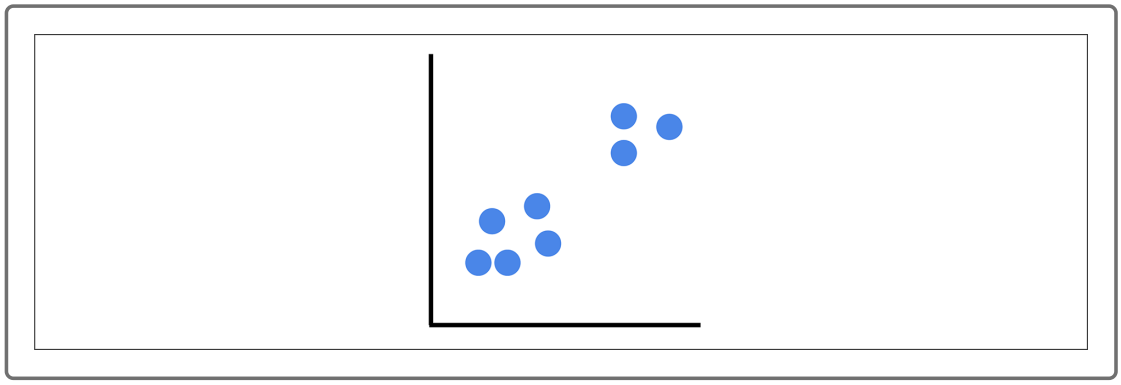
Now, you might wonder how we determine the ideal amount of clusters, as we did for the elbow curve. We'll cover that in the next section with dendrograms.

18.6.2 Dendrograms

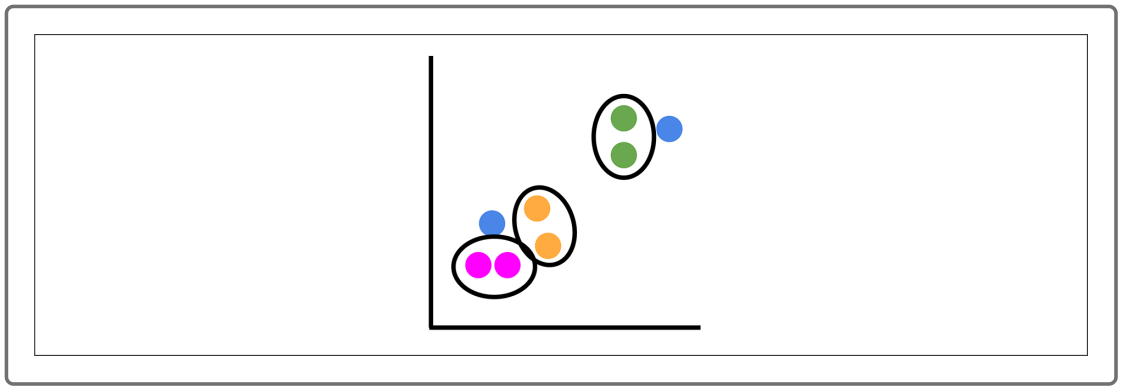
Just like with K-means, trial and error for the amount of clusters is not ideal, even in hierarchical clustering. Luckily, there is a similar method with the use of dendrograms.

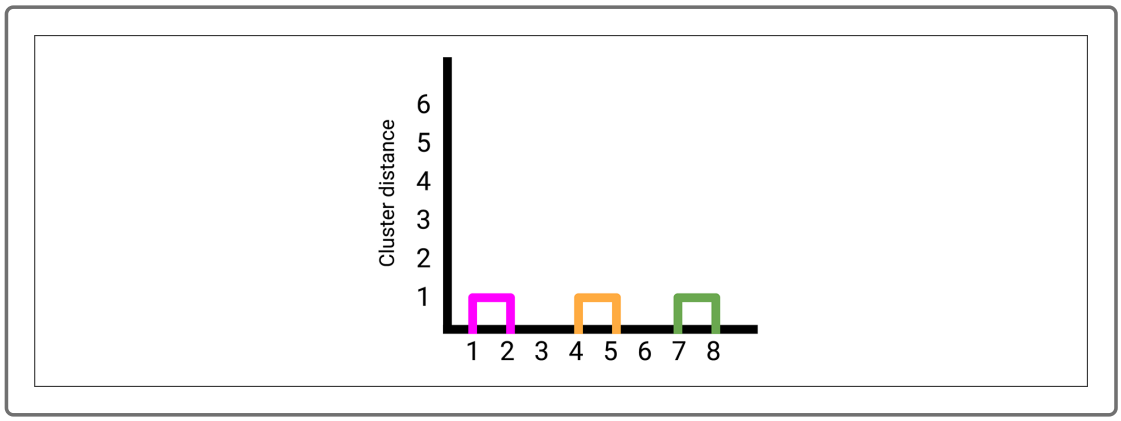
A **dendrogram** is a graph that keeps the values of the points on the x-axis, then connects all the points as they are clustered. This is similar to the elbow curve, as it gives us a better idea of the ideal amount of clusters we want to use. After making a dendrogram, you'll know how many clusters to make based on how refined you want them to be.

Let's build on the clusters from the prior exercise to create our first dendrogram:

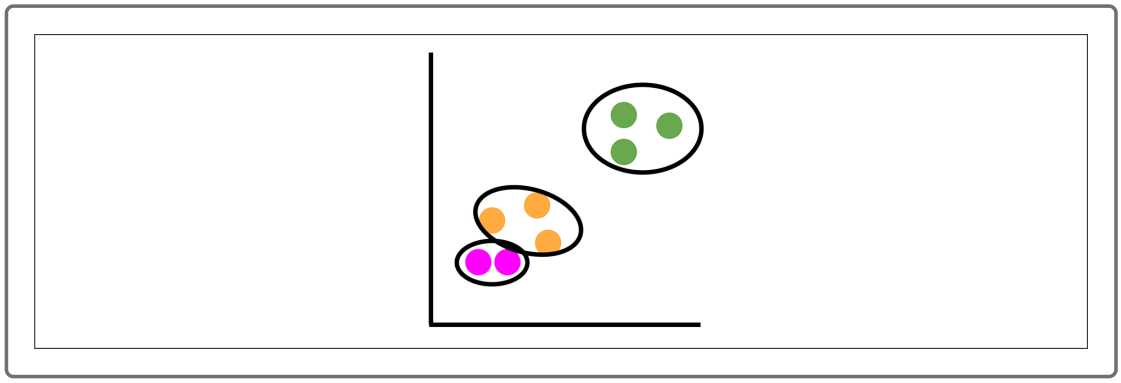


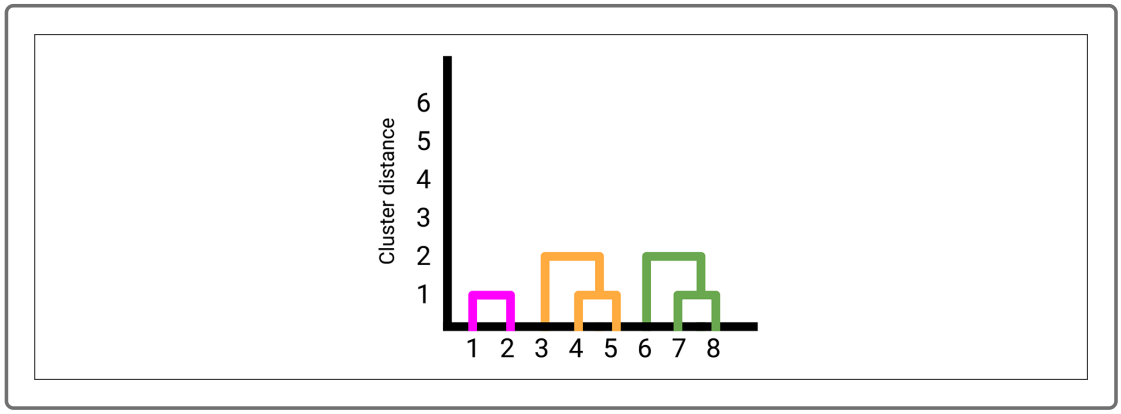
Each point on the axis represents our initial point. Next, we'll start connecting the points with a tree for each cluster that is joined:



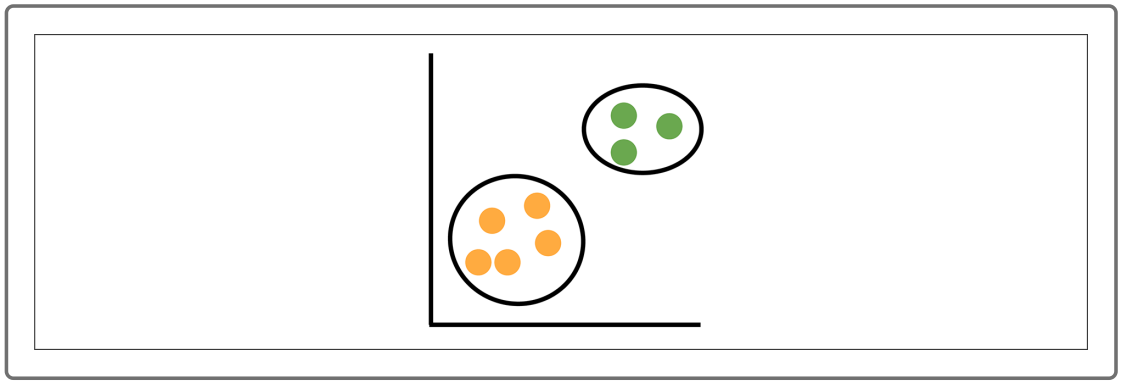


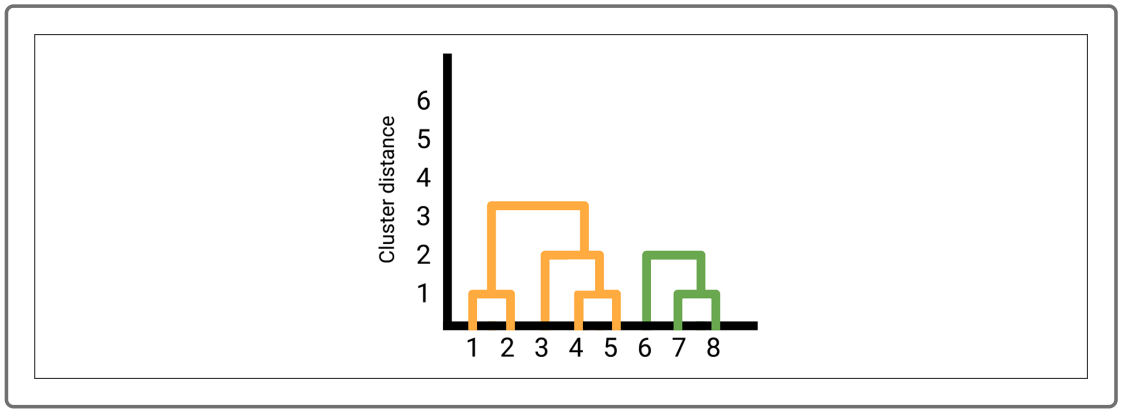
We now have three clusters, with two points unaccounted for, and the points that were merged into a cluster will be connected in the dendrogram:



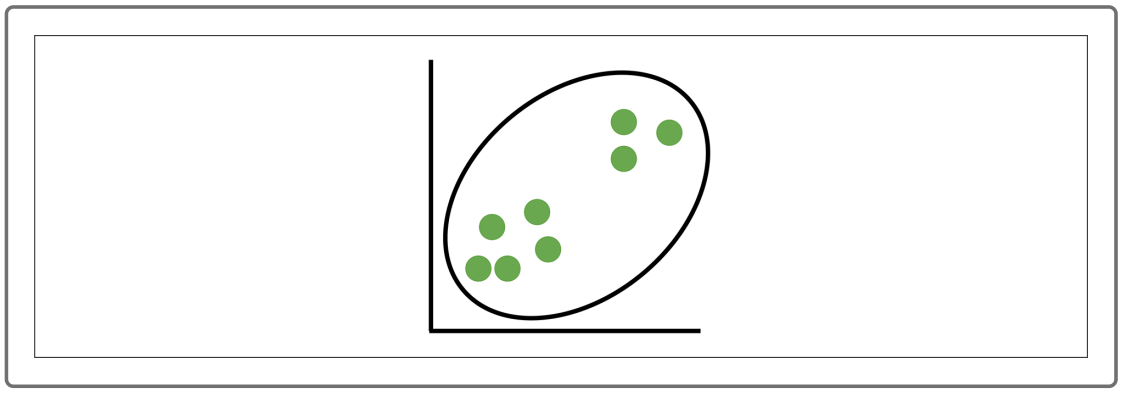


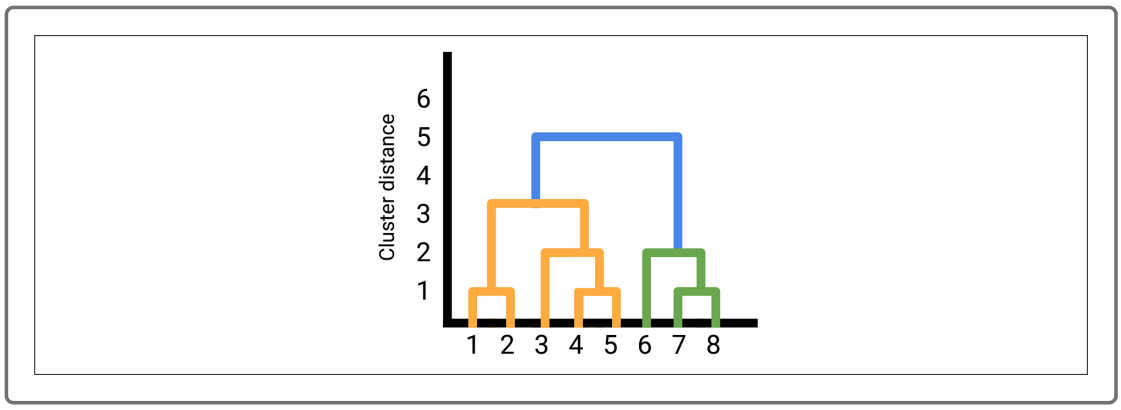
Next, combine the next merger of two clusters:



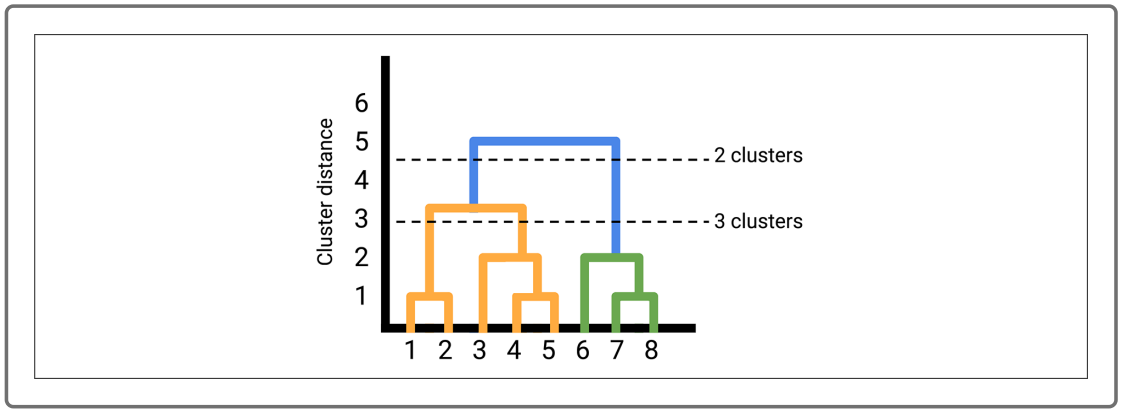


In the previous example, we didn't need to merge all points into one cluster. As you'll see, we'll join the remaining two clusters into one and plot it on the dendrogram:





Now we can set the height to determine how many clusters we want:



Choose the tolerance of cluster distance and draw a line at that point. Note how many clusters are beneath that line.

The solid horizontal lines indicate how far apart the clusters are from each other before they're merged. The dashed line for three clusters indicates that the longest horizontal lines below it must be that far away from each other to form three clusters. The dashed line for two clusters indicates that the top horizontal lines are at a greater distance from three clusters to two clusters. Finally, the horizontal line above the dashed line for two clusters would be an even greater distance if we were to form one cluster.

Therefore, the greater the distance, the less similarity exists. There is no correct choice between two and three here. Again, that decision is left up to you, the analyst, to decide how refined you want the clusters to be.

In the next section, we'll return to the iris dataset and see how running hierarchical clustering works.

18.6.3 Running Hierarchical Clustering

The two of you are curious to see what hierarchical clustering has to offer over K-means and decide to test it out on the iris dataset.

Open a notebook and enter the following code to import our libraries. (Most of these should look familiar by this point, with the only new one being the AgglomerativeClustering library, the hierarchical clustering algorithm that will replace K-means.):

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
import hvplot.pandas
```

Enter the code to load in the iris dataset:

```
# Load data
file = "Resources/new_iris_data.csv"
df_iris = pd.read_csv(file)
df_iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

SKILL DRILL

Apply PCA to reduce the dataset from four features to two.

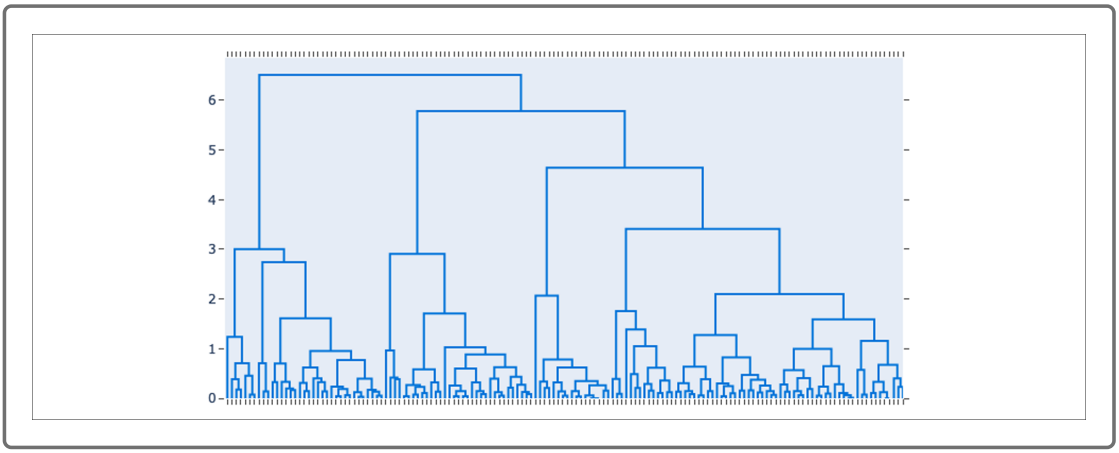
After PCA has been applied, it is time to run the hierarchical clustering algorithm. We start by creating a dendrogram. Enter the code to import the libraries:

```
import plotly.figure_factory as ff
```

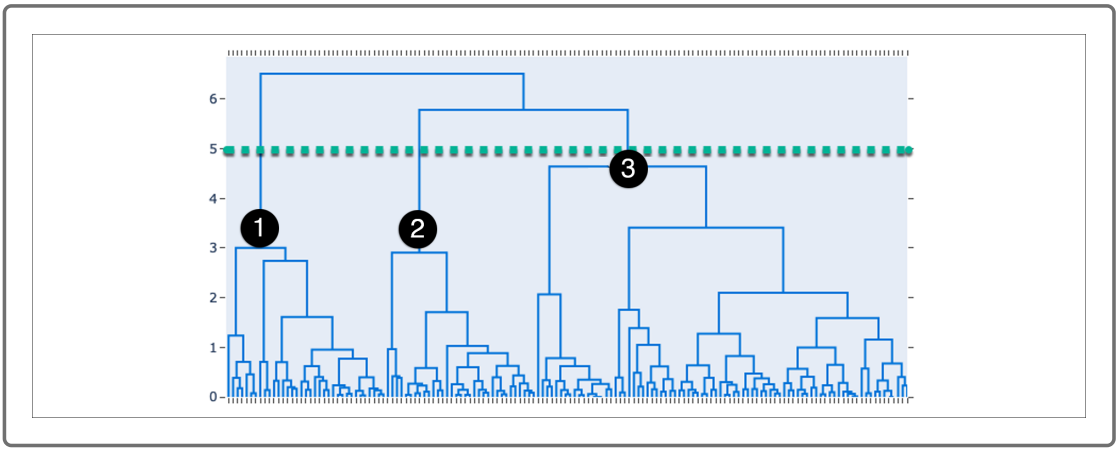
Enter the code to create a dendrogram. We'll pass a `color_threshold` of 0 to make all the branches the same color:

```
# Create the dendrogram
fig = ff.create_dendrogram(df_iris_pca, color_threshold=0)
fig.update_layout(width=800, height=500)
fig.show()
```

The resulting dendrogram will look as follows:



Now it is up to us to determine how many clusters we want to make. Remember, the higher the horizontal lines, the less similarity there is between the clusters. We know the iris dataset contains three clusters. The cutoff will be set at five to obtain three clusters:



IMPORTANT

> We knew ahead of time the number of clusters to make; however, the cutoff line on the dendrogram seems high in terms of distances. This is one of the difficulties when using a dendrogram.

Now it's time to run the hierarchical algorithm. Agglomerative clustering is another name for hierarchical clustering. Enter the following code:

```
agg = AgglomerativeClustering(n_clusters=3)
model = agg.fit(df_iris_pca)
```

This will set up our model, and since you're working with a dataset that you're already familiar with, there should be three clustered groups we decided previously, so three will be passed into the `n_clusters` parameter. Then the model is fit against your `df_iris_pca` DataFrame.

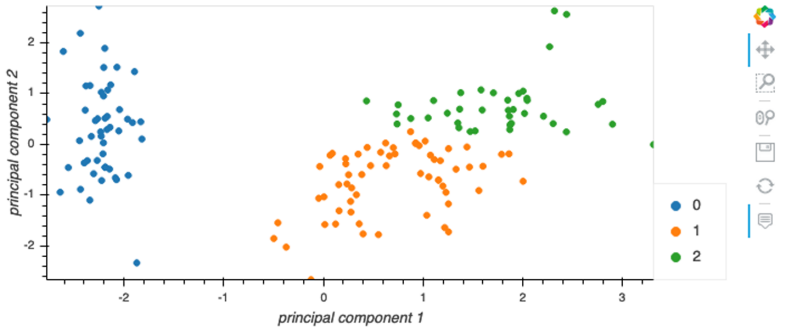
Next, add a class column that will be used to identify the clusters:

```
# Add a new class column to df_iris
df_iris_pca["class"] = model.labels_
df_iris_pca.head()
```

	principal component 1	principal component 2	class
0	-2.264542	0.505704	0
1	-2.086426	-0.655405	0
2	-2.367950	-0.318477	0
3	-2.304197	-0.575368	0
4	-2.388777	0.674767	0

Finally, create a plot to show the results of the hierarchical clustering algorithm:

```
df_iris_pca.hvplot.scatter(
    x="principal component 1",
    y="principal component 2",
    hover_cols=["class"],
    by="class",
)
```



You'll see that the process is similar for both types of clustering algorithms, and so are the results. You decide whether to apply the K-means or hierarchical clustering algorithm. In the next section, we'll review the pros and cons of each clustering algorithm.

SKILL DRILL

Re-run the algorithm using different cutoffs from the dendrogram.

18.6.4 K-means vs. Hierarchical Clustering

Hierarchical clustering seems like a fairly interesting idea, but you wonder what the differences are between K-means and hierarchical.

The K-means algorithm is the main algorithm we used in this module. It is easy, runs relatively quickly, and can scale to large datasets. This is not to say there aren't drawbacks to the K-means algorithm.

Behind the scenes, K-means is dependent on random initialization, so the outcome depends on a random seed. With K-means, you need to have an idea of how many clusters you're looking for ahead of time, which might not always be known. This can be an issue when the points of data are not so clearly grouped into clusters, as K-means works best for spherical-looking data with similar density points closely grouped together.

With hierarchical clustering and the use of dendrograms, it's easier to pick how many clusters we want without making any assumptions since a *K* value does not need to be known ahead of time.

The dendrogram might not always create as clear of a choice as we would like, and it leaves the final decision up to the analyst. With the iris dataset,

we knew the *K* value ahead of time, so using K-means in that situation would make more sense. Hierarchical clustering might not work as well on larger datasets because it is slower at run time, and there are a lot of decisions to be made about when to merge groups of clusters.

NOTE

> Both clustering algorithms have their pros and cons. Read the **No Free Lunch (NFL) theorem (https://en.wikipedia.org/wiki/No_free_lunch_theorem)** , which states that there will always be times when one algorithm outperforms the other, and vice versa.

Match the following clustering algorithm with the appropriate description.

		K-means	Hierarchical	Both
A	Requires the clusters known ahead of time.	○	○	○
B	Still leaves it up to the user to analyze.	○	○	○
C	Uses a dendrogram.	○	○	○
D	Uses the elbow curve.	○	○	○
E	Groups based on a distance metric.	○	○	○
F	Is good for large data.	○	○	○
G	Is good for grouping data.	○	○	○

Check Answer

Finish ▸

Module 18 Challenge

Submit Assignment

Due May 17 by 11:59pm **Points** 100 **Submitting** a text entry box or a website url

> You and Martha have done your research. You understand what unsupervised learning is used for, how to process data, how to cluster, how to reduce your dimensions and what goes on when you use PCA. It's time to put all these skills to use and create an analysis for your clients looking to get into the cryptocurrency market.

In this challenge, you'll use unsupervised learning to analyze data on the cryptocurrencies traded on the market.

Background

Martha is a senior manager for the Advisory Services Team at Accountability Accounting, one of your most important clients. Accountability Accounting, a prominent investment bank, is interested in offering a new cryptocurrencies investment portfolio for its customers. The company, however, is lost in the immense universe of cryptocurrencies and asks you to present a report of what cryptocurrencies are on the trading market and how cryptocurrencies could be grouped toward creating a classification for developing this new investment product.

The data Martha will be working with is not ideal, so it will be processed to fit the machine learning models. Since there is no known output for what Martha is looking for, she decided to use unsupervised learning. To group the cryptocurrencies, Martha decided on a clustering algorithm to help determine about investing in this product. She'll use data visualizations to share her findings with the board.

Objectives

The goals for this challenge are for you to:

- Prepare the data for dimensions reduction with PCA and clustering using K-means.
- Reduce data dimensions using PCA algorithms from sklearn.
- Predict clusters using cryptocurrencies data using the K-means algorithm form sklearn.
- Create some plots and data tables to present your results.

Instructions

Begin by downloading the CSV you need to complete the challenge.

Download cryptocurrency data (crypto_data.csv)

Data Preprocessing

In this section, you have to load the information about cryptocurrencies from the provided CSV file and perform some data preprocessing tasks. The data was retrieved from **CryptoCompare (https://min-api.cryptocompare.com/data/all/coinlist)** .

Start by loading the data in a Pandas DataFrame named “crypto_df.” Continue with the following data preprocessing tasks:

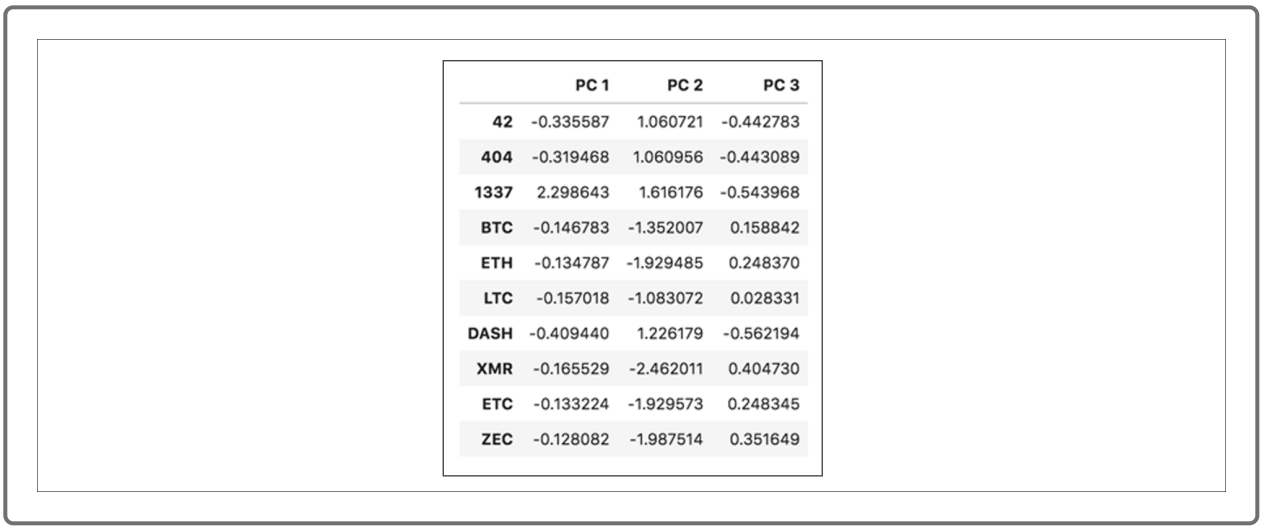
1. Remove all cryptocurrencies that aren’t trading.
2. Remove all cryptocurrencies that don’t have an algorithm defined.
3. Remove the IsTrading column.
4. Remove all cryptocurrencies with at least one null value.
5. Remove all cryptocurrencies without coins mined.
6. Store the names of all cryptocurrencies on a DataFramed named coins_name, and use the crypto_df.index as the index for this new DataFrame.
7. Remove the CoinName column.
8. Create dummies variables for all of the text features, and store the resulting data on a DataFrame named X.
9. Use the **StandardScaler from sklearn (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)** to standardize all of the data from the X DataFrame. Remember, this is important prior to using PCA and K-means algorithms.

Reducing Data Dimensions Using PCA

Use the **PCA algorithm from sklearn (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)** to reduce the dimensions of the X DataFrame down to three principal components.

Once you have reduced the data dimensions, create a DataFrame named “pcs_df” that includes the following columns: PC 1, PC 2, and PC 3. Use the crypto_df.index as the index for this new DataFrame.

You should have a DataFrame like the following:



	PC 1	PC 2	PC 3
42	-0.335587	1.060721	-0.442783
404	-0.319468	1.060956	-0.443089
1337	2.298643	1.616176	-0.543968
BTC	-0.146783	-1.352007	0.158842
ETH	-0.134787	-1.929485	0.248370
LTC	-0.157018	-1.083072	0.028331
DASH	-0.409440	1.226179	-0.562194
XMR	-0.165529	-2.462011	0.404730
ETC	-0.133224	-1.929573	0.248345
ZEC	-0.128082	-1.987514	0.351649

Clustering Cryptocurrencies Using K-means

You'll use the **KMeans algorithm from sklearn (https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html)** to cluster the cryptocurrencies using the PCA data.

Complete the following tasks:

1. Create an elbow curve to find the best value for *K*, and use the pcs_df DataFrame.
2. Once you define the best value for *K*, run the K-means algorithm to predict the *K* clusters for the cryptocurrencies' data. Use the pcs_df to run the K-means algorithm.
3. Create a new DataFrame named "clustered_df," that includes the following columns: Algorithm, ProofType, TotalCoinsMined, TotalCoinSupply, PC 1, PC 2, PC 3, CoinName, and Class. You should maintain the index of the crypto_df DataFrames as is shown below:

	Algorithm	ProofType	TotalCoinsMined	TotalCoinSupply	PC 1	PC 2	PC 3	CoinName	Class
42	Scrypt	PoW/PoS	42	42	-0.335587	1.060721	-0.442783	42 Coin	0
404	Scrypt	PoW/PoS	1.00862e+09	532000000	-0.319468	1.060956	-0.443089	404Coin	0
1337	X13	PoW/PoS	2.92794e+10	314159265359	2.298643	1.616176	-0.543968	EliteCoin	0
BTC	SHA-256	PoW	17918662	21000000	-0.146783	-1.352007	0.158842	Bitcoin	1
ETH	Ethash	PoW	1.07626e+08	0	-0.134787	-1.929485	0.248370	Ethereum	1
LTC	Scrypt	PoW	6.30392e+07	84000000	-0.157018	-1.083072	0.028331	Litecoin	1
DASH	X11	PoW/PoS	9.02401e+06	22000000	-0.409440	1.226179	-0.562194	Dash	0
XMR	CryptoNight-V7	PoW	1.71937e+07	0	-0.165529	-2.462011	0.404730	Monero	1
ETC	Ethash	PoW	113255332	210000000	-0.133224	-1.929573	0.248345	Ethereum Classic	1
ZEC	Equihash	PoW	7.35252e+06	21000000	-0.128082	-1.987514	0.351649	ZCash	1

Visualizing Results

You'll create data visualizations to present the final results.

Complete the following tasks:

1. Create a 3D scatter plot using Plotly Express to plot the clusters using the clustered_df DataFrame. You should include the following parameters on the plot: `hover_name="CoinName" and hover_data=["Algorithm"]` to show this additional info on each data point.
2. Use `hvplot.table` to create a data table with all the current tradable cryptocurrencies. The table should have the following columns: CoinName, Algorithm, ProofType, TotalCoinSupply, TotalCoinsMined, and Class.
3. Create a scatter plot using `hvplot.scatter` to present the clustered data about cryptocurrencies having x="TotalCoinsMined" and y="TotalCoinSupply" to contrast the number of available coins versus the total number of mined coins. Use the hover_cols=["CoinName"] parameter to include the cryptocurrency name on each data point.

Submission

Make sure your repo is up to date and includes the following:

- A `README.md` file containing a short description of your project.
- A Jupyter Notebook with all of your code.

Submit the link to your repository through Canvas.

Rubric

Please **download the detailed rubric** to access the assessment criteria.

Note: You are allowed to miss up to two Challenge assignments and still earn your certificate. If you complete all Challenge assignments, your lowest two grades will be dropped. If you wish to skip this assignment, click Submit then indicate you are skipping by typing “I choose to skip this assignment” in the text box.

Module 18 Rubric

Criteria	Ratings					Pts
Data Pre-Processing Please see detailed rubric linked in Challenge description.	**30.0 pts** **Mastery**	**23.0 pts** **Approaching Mastery**	**16.0 pts** **Progressing**	**7.0 pts** **Emerging**	**0.0 pts** **Incomplete**	30.0 pts
Reducing Data Dimensions Using PCA Please see detailed rubric linked in Challenge description.	**20.0 pts** **Mastery**	**15.0 pts** **Approaching Mastery**	**10.0 pts** **Progressing**	**5.0 pts** **Emerging**	**0.0 pts** **Incomplete**	20.0 pts
Clustering Cryptocurrencies Using K-Means Please see detailed rubric linked in Challenge description.	**20.0 pts** **Mastery**	**15.0 pts** **Approaching Mastery**	**10.0 pts** **Progressing**	**5.0 pts** **Emerging**	**0.0 pts** **Incomplete**	20.0 pts
Visualizing Results Please see detailed rubric linked in Challenge description.	**30.0 pts** **Mastery**	**23.0 pts** **Approaching Mastery**	**16.0 pts** **Progressing**	**7.0 pts** **Emerging**	**0.0 pts** **Incomplete**	30.0 pts
						Total Points: 100.0

	Mastery	Approaching Mastery	Progressing	Emerging	Incomplete
Data Pre-Processing (30 points)	Data is appropriately pre-processed, including all of the following steps. ✓ Remove all cryptocurrencies that are not on trading. ✓ Remove all cryptocurrencies that have not an algorithm defined. ✓ Remove the IsTrading column. ✓ Remove all cryptocurrencies with at least one null value. ✓ Remove all cryptocurrencies without coins mined. ✓ Store the names of all cryptocurrencies on a DataFramed named coins_name, ✓ Use the crypto_df.index as the index for this new DataFrame. ✓ Remove the CoinName column. ✓ Create dummies variables for all the text features, store the resulting data on a DataFrame named X. ✓ Use the StandardScaler from sklearn to standardize all the data of the X DataFrame.	Data is appropriately pre-processed, including 7-9 of the following steps. ✓ Remove all cryptocurrencies that are not on trading. ✓ Remove all cryptocurrencies that have not an algorithm defined. ✓ Remove the IsTrading column. ✓ Remove all cryptocurrencies with at least one null value. ✓ Remove all cryptocurrencies without coins mined. ✓ Store the names of all cryptocurrencies on a DataFramed named coins_name, ✓ Use the crypto_df.index as the index for this new DataFrame. ✓ Remove the CoinName column. ✓ Create dummies variables for all the text features, store the resulting data on a DataFrame named X. ✓ Use the StandardScaler from sklearn to standardize all the data of the X DataFrame.	Data is pre-processed, including 4-6 of the following steps. ✓ Remove all cryptocurrencies that are not on trading. ✓ Remove all cryptocurrencies that have not an algorithm defined. ✓ Remove the IsTrading column. ✓ Remove all cryptocurrencies with at least one null value. ✓ Remove all cryptocurrencies without coins mined. ✓ Store the names of all cryptocurrencies on a DataFramed named coins_name, ✓ Use the crypto_df.index as the index for this new DataFrame. ✓ Remove the CoinName column. ✓ Create dummies variables for all the text features, store the resulting data on a DataFrame named X. ✓ Use the StandardScaler from sklearn to standardize all the data of the X DataFrame.	Data is incompletely pre-processed, including 1-3 of the following steps. ✓ Remove all cryptocurrencies that are not on trading. ✓ Remove all cryptocurrencies that have not an algorithm defined. ✓ Remove the IsTrading column. ✓ Remove all cryptocurrencies with at least one null value. ✓ Remove all cryptocurrencies without coins mined. ✓ Store the names of all cryptocurrencies on a DataFramed named coins_name, ✓ Use the crypto_df.index as the index for this new DataFrame. ✓ Remove the CoinName column. ✓ Create dummies variables for all the text features, store the resulting data on a DataFrame named X. ✓ Use the StandardScaler from sklearn to standardize all the data of the X DataFrame.	No submission was received -OR- Submission was empty or blank -OR- Submission contains evidence of academic dishonesty
Reducing Data Dimensions	✓ Use the PCA algorithm from sklearn to reduce the dimensions of the X DataFrame down to	✓ Use the PCA algorithm from sklearn to reduce the dimensions of the X DataFrame down to three	✓ Use the PCA algorithm from sklearn to reduce the dimensions of the X DataFrame down to three	✓ Use the PCA algorithm from sklearn to reduce the dimensions of the X DataFrame down to three	

Using PCA (20 points)	three principal components. ✓ Create a DataFrame named pcs_df ✓ Create three columns, called “PC 1,” “PC 2,” and “PC 3” ✓ Use the crypto_df.index as the index for this new DataFrame.	principal com And two of the three below steps: ✓ Create a DataFrame named pcs_df ✓ Create three columns, called “PC 1,” “PC 2,” and “PC 3”ponents. ✓ Use the crypto_df.index as the index for this new DataFrame.	principal components. And one of the three below steps: ✓ Create a DataFrame named pcs_df ✓ Create three columns, called “PC 1,” “PC 2,” and “PC 3”ponents. ✓ Use the crypto_df.index as the index for this new DataFrame.	principal components.	
Clustering Cryptocurrencies Using K-Means (20 points)	KMeans algorithm from sklearn is used to cluster the cryptocurrencies using the PCA data, including the below steps. ✓Create an Elbow Curve to find the best value for k, use the pcs_df DataFrame. ✓Predict the k clusters for the cryptocurrencies data. Use the pcs_df to run the KMeans algorithm. ✓Create a new DataFrame named clustered_df, that includes the following columns: Algorithm, ProofType, TotalCoinsMined, TotalCoinSupply, PC 1, PC 2, PC 3, CoinName, and Class. ✓ Maintains the index of the crypto_df DataFrames	KMeans algorithm from sklearn is used to cluster the cryptocurrencies using the PCA data, including three of the below steps. ✓Create an Elbow Curve to find the best value for k, use the pcs_df DataFrame. ✓Predict the k clusters for the cryptocurrencies data. Use the pcs_df to run the KMeans algorithm. ✓Create a new DataFrame named clustered_df, that includes the following columns: Algorithm, ProofType, TotalCoinsMined, TotalCoinSupply, PC 1, PC 2, PC 3, CoinName, and Class. ✓ Maintains the index of the crypto_df DataFrames	KMeans algorithm from sklearn is used to cluster the cryptocurrencies using the PCA data, including two of the below steps. ✓Create an Elbow Curve to find the best value for k, use the pcs_df DataFrame. ✓Predict the k clusters for the cryptocurrencies data. Use the pcs_df to run the KMeans algorithm. ✓Create a new DataFrame named clustered_df, that includes the following columns: Algorithm, ProofType, TotalCoinsMined, TotalCoinSupply, PC 1, PC 2, PC 3, CoinName, and Class. ✓ Maintains the index of the crypto_df DataFrames	KMeans algorithm from sklearn is used to cluster the cryptocurrencies using the PCA data, including one of the below steps. ✓Create an Elbow Curve to find the best value for k, use the pcs_df DataFrame. ✓Predict the k clusters for the cryptocurrencies data. Use the pcs_df to run the KMeans algorithm. ✓Create a new DataFrame named clustered_df, that includes the following columns: Algorithm, ProofType, TotalCoinsMined, TotalCoinSupply, PC 1, PC 2, PC 3, CoinName, and Class. ✓ Maintains the index of the crypto_df DataFrames	
Visualizing Results (30 points)	Visualizations include two scatter plots and a data table, described below, with no errors. **Scatter Plot 1** ✓Create a 3D-Scatter using Plotly Express to plot the clusters using the clustered_df DataFrame.	Visualizations include two scatter plots and a data table, described below, with some minor errors. **Scatter Plot 1** ✓Create a 3D-Scatter using Plotly Express to plot the clusters using the clustered_df DataFrame. ✓3D-Scatter plot includes the	Visualizations include two of the three visualizations described below. **Scatter Plot 1** ✓Create a 3D-Scatter using Plotly Express to plot the clusters using the clustered_df DataFrame. ✓3D-Scatter plot includes the	Visualizations include one of the three visualizations described below. **Scatter Plot 1** ✓Create a 3D-Scatter using Plotly Express to plot the clusters using the clustered_df DataFrame. ✓3D-Scatter plot includes the following parameters:	

	✓3D-Scatter plot includes the following parameters: hover_name="CoinName" and hover_data=["Algorithm"] to show this additional info on each data point. **Data Table** ✓Use hvplot.table to create a data table with all the current tradable cryptocurrencies. ✓The table should have the following columns: CoinName, Algorithm, ProofType, TotalCoinSupply, TotalCoinsMined, and Class. **Scatter Plot 2** ✓Create a scatter plot using hvplot.scatter, to present the clustered data about cryptocurrencies having x="TotalCoinsMined" and y="TotalCoinSupply" to contrast the number of available coins versus the total number of mined coins. ✓Use the hover_cols=["CoinName"] parameter to include the cryptocurrency name on each data point.	following parameters: hover_name="CoinName" and hover_data=["Algorithm"] to show this additional info on each data point. **Data Table** ✓Use hvplot.table to create a data table with all the current tradable cryptocurrencies. ✓The table should have the following columns: CoinName, Algorithm, ProofType, TotalCoinSupply, TotalCoinsMined, and Class. **Scatter Plot 2** ✓Create a scatter plot using hvplot.scatter, to present the clustered data about cryptocurrencies having x="TotalCoinsMined" and y="TotalCoinSupply" to contrast the number of available coins versus the total number of mined coins. ✓Use the hover_cols=["CoinName"] parameter to include the cryptocurrency name on each data point.	following parameters: hover_name="CoinName" and hover_data=["Algorithm"] to show this additional info on each data point. **Data Table** ✓Use hvplot.table to create a data table with all the current tradable cryptocurrencies. ✓The table should have the following columns: CoinName, Algorithm, ProofType, TotalCoinSupply, TotalCoinsMined, and Class. **Scatter Plot 2** ✓Create a scatter plot using hvplot.scatter, to present the clustered data about cryptocurrencies having x="TotalCoinsMined" and y="TotalCoinSupply" to contrast the number of available coins versus the total number of mined coins. ✓Use the hover_cols=["CoinName"] parameter to include the cryptocurrency name on each data point.	hover_name="CoinName" and hover_data=["Algorithm"] to show this additional info on each data point. **Data Table** ✓Use hvplot.table to create a data table with all the current tradable cryptocurrencies. ✓The table should have the following columns: CoinName, Algorithm, ProofType, TotalCoinSupply, TotalCoinsMined, and Class. **Scatter Plot 2** ✓Create a scatter plot using hvplot.scatter, to present the clustered data about cryptocurrencies having x="TotalCoinsMined" and y="TotalCoinSupply" to contrast the number of available coins versus the total number of mined coins. ✓Use the hover_cols=["CoinName"] parameter to include the cryptocurrency name on each data point.	

Module 18
Career Connection

Introduction

Welcome back to another Career Connection! This week you continued to explore machine learning and how algorithms are used in data analytics. You also spent time digging into the difference between supervised and unsupervised learning, as well as how to cluster data using the K-means algorithm.

It was a packed week! This material can be tricky, but it's essential to your future career as a data engineer.

NOTE

> The artificial intelligence talent market is hot! It's estimated that the AI business value will reach **$3.9 trillion by 2022 (https://enterprisersproject.com/article/2019/8/ai-artificial-intelligence-careers-salaries-7-statistics)** .

Machine learning is an extremely popular career choice. According to **Indeed (http://blog.indeed.com/2019/03/14/best-jobs-2019/)** , in 2019, machine learning engineers earned an average base salary of $146,085, and job postings for this position grew by 344%. There are a variety of career paths you might take in machine learning, including:

Machine Learning Engineer

Machine learning engineers work in programming languages such as Python, JavaScript, and Scala, with their respective and appropriate machine learning libraries. They also analyze data to create different algorithms that run autonomously with little need for human interaction.

Data Scientist

Data scientists use data analytics technologies in order to analyze and interpret large amounts of data that can produce actionable insights. In short, they look at data and determine what it tells us. Using languages such as Python and SQL, data scientists also use machine learning technologies to improve their analyses.

Natural Language Processing Scientist

Natural language processing (NLP) scientists create machine algorithms that learn patterns of speech and translate spoken words in other languages. This job combines skills from machine learning as well as communication, grammar, syntax, and spelling.

Here are some **statistics about machine learning careers (https://enterprisersproject.com/article/2019/8/ai-artificial-intelligence-careers-salaries-7-statistics)** :

- As a result of AI adoption, 40% of Global 2000 organizations are adding jobs.
- By 2022, an estimated 133 million new jobs will be created by AI.
- There are approximately 7,000 AI job openings in the United States.
- The average salary for machine learning engineers is $142,858.

Hypothetical Case Study

This week, there is no narrative for the hypothetical case study. Instead, we want you to consider something that could have practical application in your life.

Consider a real-life situation in which the skills you learned this week would apply. Pseudocode a solution for tackling it. You can use the template below or write your own. Complete this assignment in any text editor, or write it in a markdown file that can be uploaded to GitHub.

NOTE

> Pseudocode means to write out in plain language (English or other!) the steps you would take in order to solve a specific problem. You do not need to include actual code here; the goal is just to create a kind of roadmap for solving the problem.

Template

Title

A short, succinct title that identifies the topic of the narrative.

Business Problem

Write two or three sentences outlining the problem and why it should be solved. In other words, why is your idea a good product proposition?

Solution

Write two or three sentences outlining the solution and/or steps that you could take to solve the problem. Your solution should utilize machine learning. Feel free to search the internet for ideas.

Consider the following example:

> **NARRATIVE**
>
> **Title:** Winematcher, learning your tastes.
>
> **Business Problem:** You organize a dinner party with friends. You put a lot of time and effort into the food being served, but you're not sure which wine goes best with each type of food. How can you make wine suggestions for your guests if you don't have expertise in wine pairing?
>
> **Solution**
> Build a wine-matching application that recommends a wine based on the dish or ingredients that you enter. This app would be a self-learning program that acquires knowledge from a large number of examples. A user enters their taste profiles and ingredients of a dish, and the application returns a matching wine.
>
> Data could be entered using database entries as well as crowd-sourcing opinions and flavor profiles from the app's users.
>
> Include libraries and algorithms that will help build the machine learning aspect of this application.

Continue to Hone Your Skills

If you're interested in hearing more about the technical interviewing process and practicing algorithms in a mock interview setting, check out our **upcoming workshops! (https://careerservicesonlineevents.splashthat.com/)**