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://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_18/new_iris_data.csv)** 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:

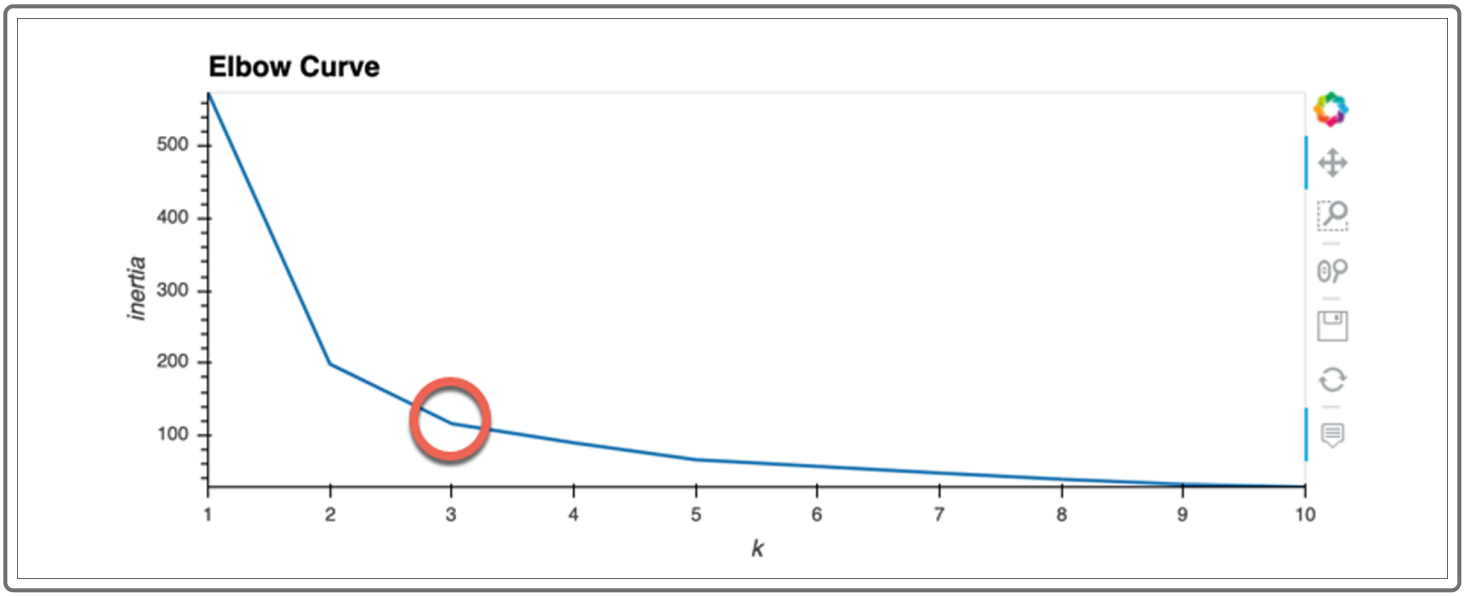
```
# Fetch the explained variance
pca.explained_variance_ratio_
```

```
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:

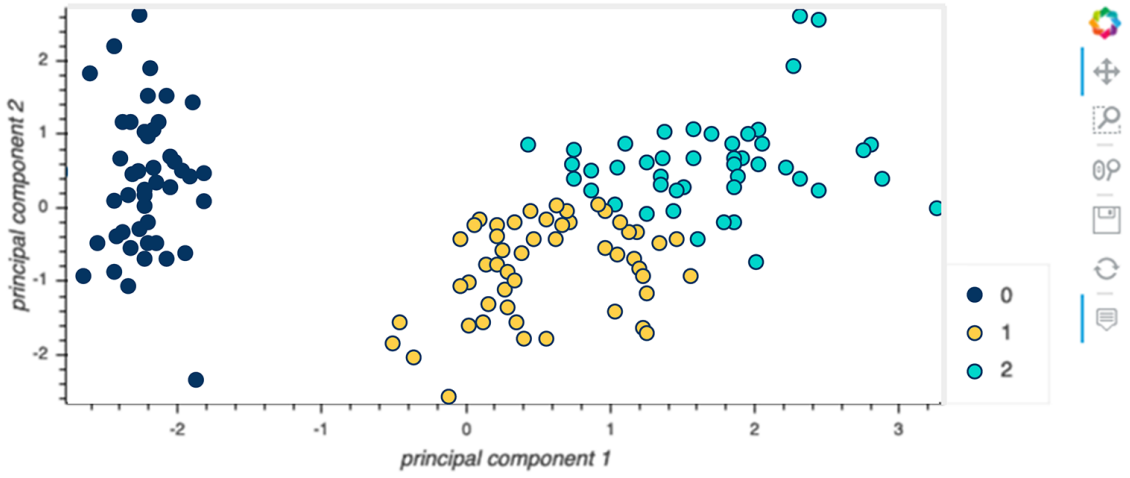
```
# Initialize the K-means model
model = KMeans(n_clusters=3, random_state=0)

# 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",
)
```

```
# Plotting the clusters
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!