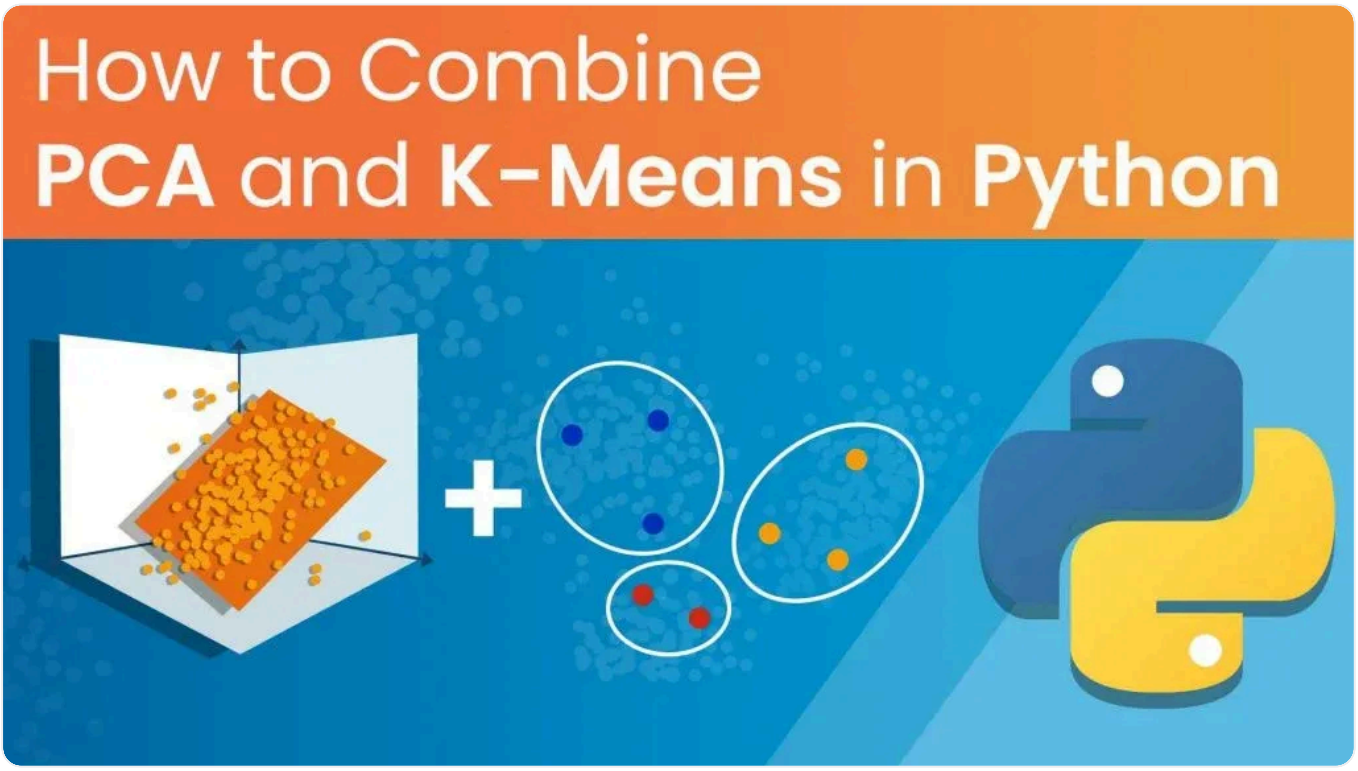
How to Combine PCA and K-means Clustering in Python?

Elitsa Kaloyanova

15 Apr 2024 • 7 min read



In this tutorial, we'll see a practical example of a mixture of PCA and K-means for clustering data using Python.

Why Combine PCA and K-means Clustering?

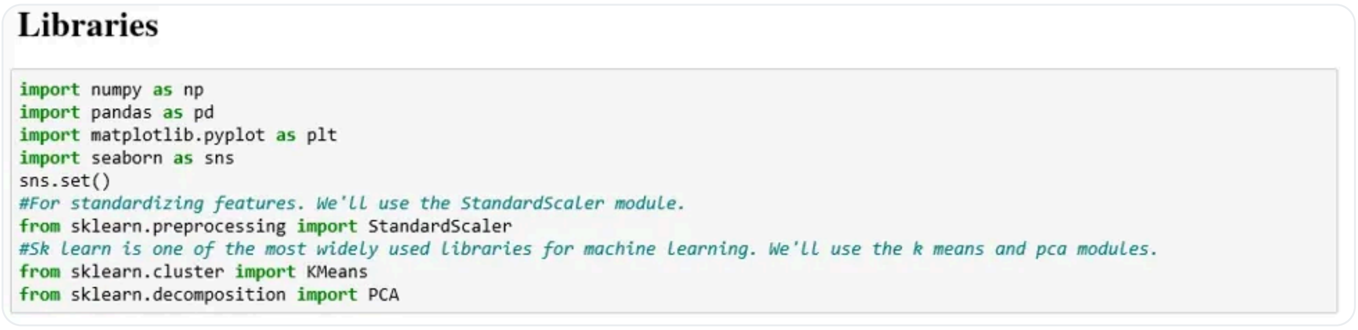
There are varying reasons for using a dimensionality reduction step such as PCA prior to data segmentation. Chief among them? By reducing the number of features, we're improving the performance of our algorithm. On top of that, by decreasing the number of features the noise is also reduced.

In the case of PCA and K-means in particular, there appears to be an even closer relationship between the two.

This paper discusses the exact relationship between the techniques and why a combination of both techniques could be beneficial. In case you're not a fan of the heavy theory, keep reading. In the next part of this tutorial, we'll begin working on our PCA and K-means methods using Python.

1. Importing and Exploring the Data Set

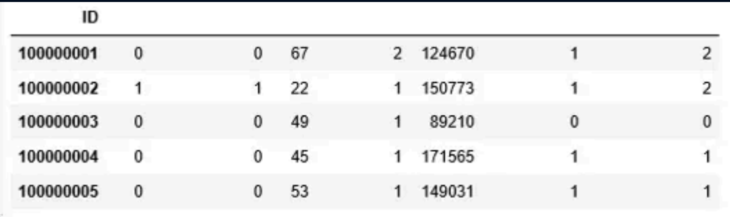
We start as we do with any programming task: by importing the relevant Python libraries. In our case they are:



Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
#For standardizing features. We'll use the StandardScaler module.
from sklearn.preprocessing import StandardScaler
#Sk learn is one of the most widely used libraries for machine learning. We'll use the k means and pca modules.
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

The second step is to acquire the data which we'll later be segmenting. We'll use customer data, which we load in the form of a pandas' data frame.



ID							
100000001	0	0	67	2	124670	1	2
100000002	1	1	22	1	150773	1	2
100000003	0	0	49	1	89210	0	0
100000004	0	0	45	1	171565	1	1
100000005	0	0	53	1	149031	1	1

The data set we've chosen for this tutorial comprises 2,000 observations and 7 features.

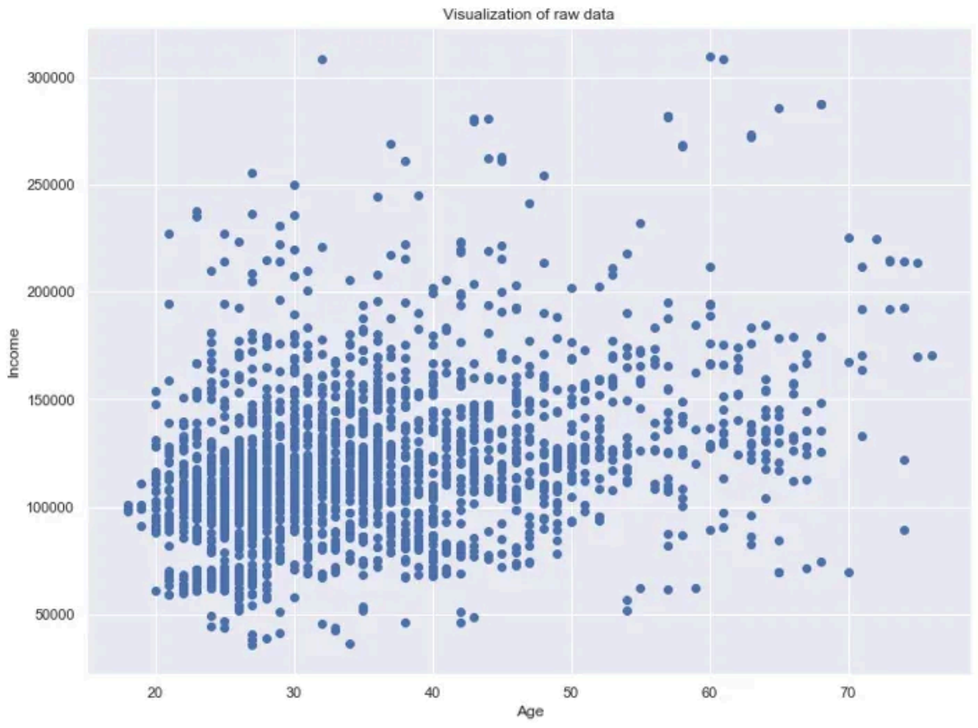
More specifically, it contains information about 2,000 individuals and has their IDs, as well as geodemographic features, such as Age, Occupation, etc.

Lastly, we take a moment to visualize the raw data set on the two numerical features; Age and Income.

Visualize Raw Data

```
# We'll plot the data. We create a 12 by 9 inches figure.
# We have 2000 data points, which we'll scatter acrros Age and Income, located on positions 2 and 4 in our data set.
plt.figure(figsize = (12, 9))
plt.scatter(df_segmentation.iloc[:, 2], df_segmentation.iloc[:, 4])
plt.xlabel('Age')
plt.ylabel('Income')
plt.title('Visualization of raw data')
```

```
Text(0.5, 1.0, 'Visualization of raw data')
```



Another observation from the graph concerns the domains of the two variables Age and Income. We understand that the domain for Age is from around 20 to 70, whereas for Income it is from around 40,000 to over 300,000. Which points to a vast difference between the range of these values. Therefore, we must incorporate an important step in our analysis, and we must first standardize our data.

Standardization is an important part of data preprocessing, which is why we've devoted the entire next paragraph precisely to this topic.

2. Data Preprocessing

Our segmentation model will be based on similarities and differences between individuals on the features that characterize them.

We'll quantify these similarities and differences.

Well, you can imagine that two persons may differ in terms of 'Age'. One may be a 20-year-old, while another – 70 years old. The difference in age is 50 years. However, it spans almost the entire range of possible ages in our dataset.

At the same time, the first individual may have an annual income of $100,000; while the second may have an annual income of $150,000. Therefore, the difference between their incomes will be $50,000.

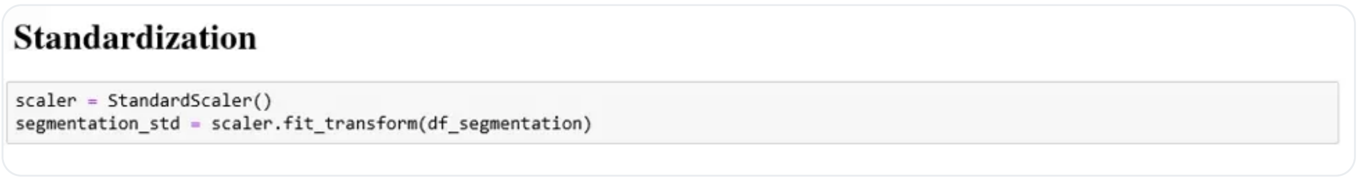
If these numbers were to go into any of our segmentation models as they are, the algorithm would believe that the two differ in terms of one variable by 50; while in terms of another by 50,000. Then, because of the mathematical nature of modeling, it would completely disregard 'Age' as a feature. The reason is that the numbers from 20 to 70 are insignificant when compared with the income values around 100K.

Why?

Therefore, it will place a much bigger **weight** on the income variable.

It is obvious that we must protect ourselves from such an outcome. What's more, in general, we want to treat all the features equally. And we can achieve that by transforming the features in a way that makes their values fall within the same numerical range. Thus, the differences between their values will be comparable. This process is commonly referred to as standardization.

For this tutorial, we'll use a Standard Scaler to standardize our data, which is currently in the df segmentation data frame:



Standardization

```
scaler = StandardScaler()
segmentation_std = scaler.fit_transform(df_segmentation)
```

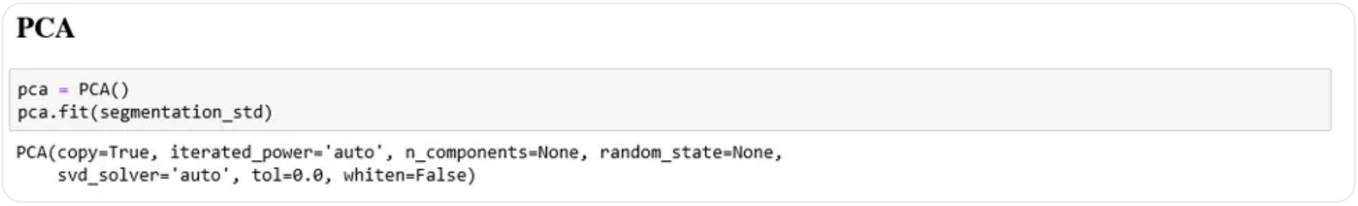
After data standardization, we may proceed with the next step, namely Dimensionality Reduction.

3. How to Perform Dimensionality Reduction with PCA?

We'll employ PCA to reduce the number of features in our data set. Before that, make sure you refresh your knowledge on what is Principal Components Analysis.

In any case, here are the steps to performing dimensionality reduction using PCA.

First, we must fit our standardized data using PCA.



PCA

```
pca = PCA()
pca.fit(segmentation_std)
```

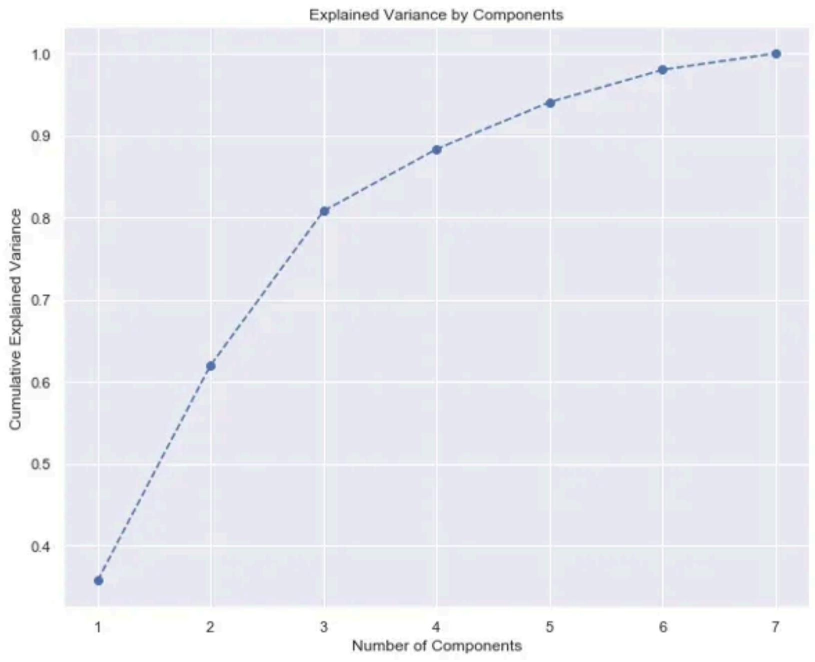
```
PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

```
# The attribute shows how much variance is explained by each of the seven individual components.
pca.explained_variance_ratio_
```

```
array([0.35696328, 0.26250923, 0.18821114, 0.0755775 , 0.05716512,
       0.03954794, 0.02002579])
```

```
plt.figure(figsize = (10,8))
plt.plot(range(1,8), pca.explained_variance_ratio_.cumsum(), marker = 'o', linestyle = '--')
plt.title('Explained Variance by Components')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
```

```
Text(0, 0.5, 'Cumulative Explained Variance')
```



The graph shows the amount of variance captured (on the y-axis) depending on the number of components we include (the x-axis). A rule of thumb is to preserve around 80 % of the variance. So, in this instance, we decide to keep 3 components.

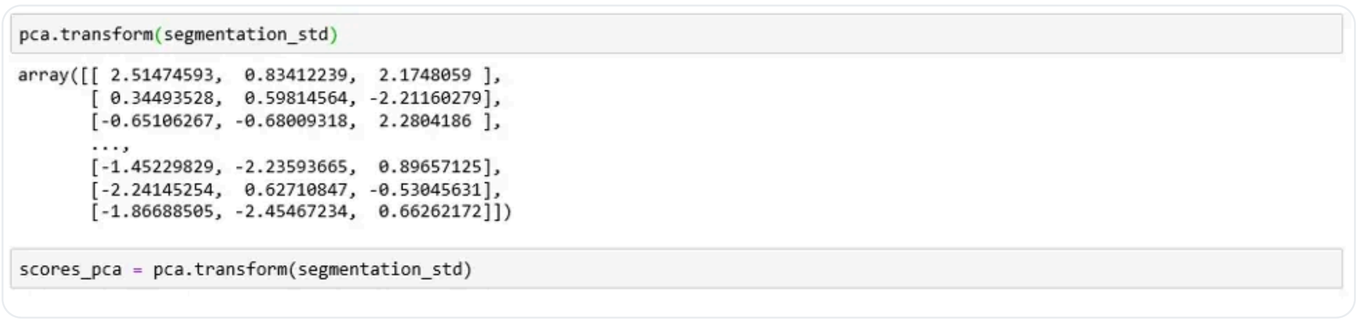
As a third step, we perform PCA with the chosen number of components.

For our data set, that means 3 principal components:

```
# We choose three components. 3 or 4 seems the right choice according to the previous graph.
pca = PCA(n_components = 3)
```

```
#Fit the model the our data with the selected number of components. In our case three.
pca.fit(segmentation_std)
```

```
PCA(copy=True, iterated_power='auto', n_components=3, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```



```
pca.transform(segmentation_std)
```

```
array([[ 2.51474593,  0.83412239,  2.1748059 ],
       [ 0.34493528,  0.59814564, -2.21160279],
       [-0.65106267, -0.68009318,  2.2804186 ],
       ...,
       [-1.45229829, -2.23593665,  0.89657125],
       [-2.24145254,  0.62710847, -0.53045631],
       [-1.86688505, -2.45467234,  0.66262172]])
```

```
scores_pca = pca.transform(segmentation_std)
```

We'll incorporate the newly obtained PCA scores in the K-means algorithm. That's how we can perform segmentation based on principal components scores instead of the original features.

4. How to Combine PCA and K-means Clustering?

As promised, it is time to combine PCA and K-means to segment our data, where we use the scores obtained by the PCA for the fit.

Based on how familiar you are with K-means, you might already know that K-means doesn't determine the number of clusters in your solution. If you need a refresher on all things K-means, you can read our dedicated blog post.

In any case, it turns out that we **ourselves** need to determine the number of clusters in a K-means algorithm.

In order to do so, we run the algorithm with a different number of clusters. Then, we determine the Within Cluster Sum of Squares or WCSS for each solution. Based on the values of the WCSS and an approach known as the Elbow method, we make a decision about how many clusters we'd like to keep.

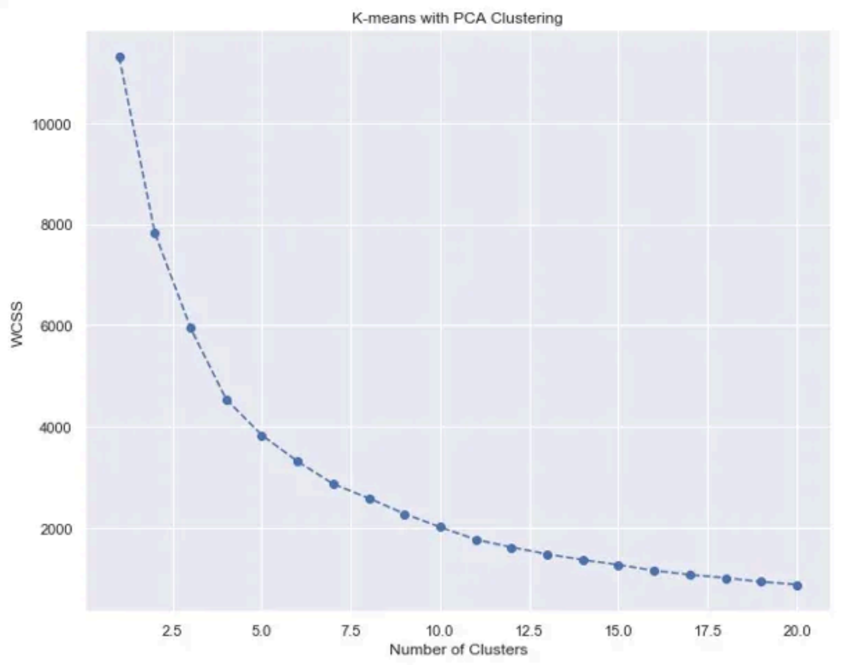
First, however, we must decide how many clustering solutions we'd test.

There is no general ruling on this issue. It really depends on the data. In our case, we test an algorithm with up to 20 clusters.

```
    kmeans_pca = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans_pca.fit(scores_pca)
    wcss.append(kmeans_pca.inertia_)
```

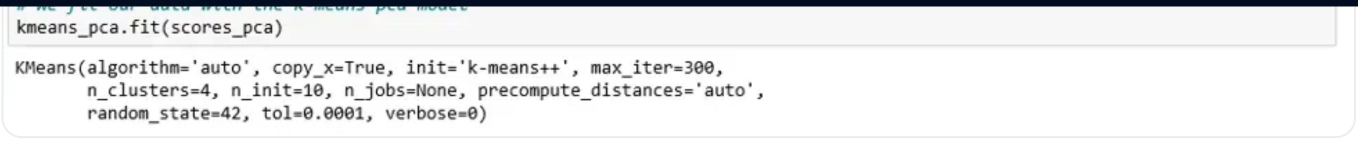
The next step involves plotting the WCSS against the number of components on a graph.

```
plt.figure(figsize = (10,8))
plt.plot(range(1, 21), wcss, marker = 'o', linestyle = '--')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.title('K-means with PCA Clustering')
plt.show()
```



And from this graph, we determine the number of clusters we'd like to keep. To that effect, we use the Elbow-method. The approach consists of looking for a kink or elbow in the WCSS graph. Usually, the part of the graph before the elbow would be steeply declining, while the part after it – much smoother. In this instance, the kink comes at the 4 clusters mark. So, we'll be keeping a four-cluster solution.

All left to do is to implement it.



```
kmeans_pca.fit(scores_pca)
```

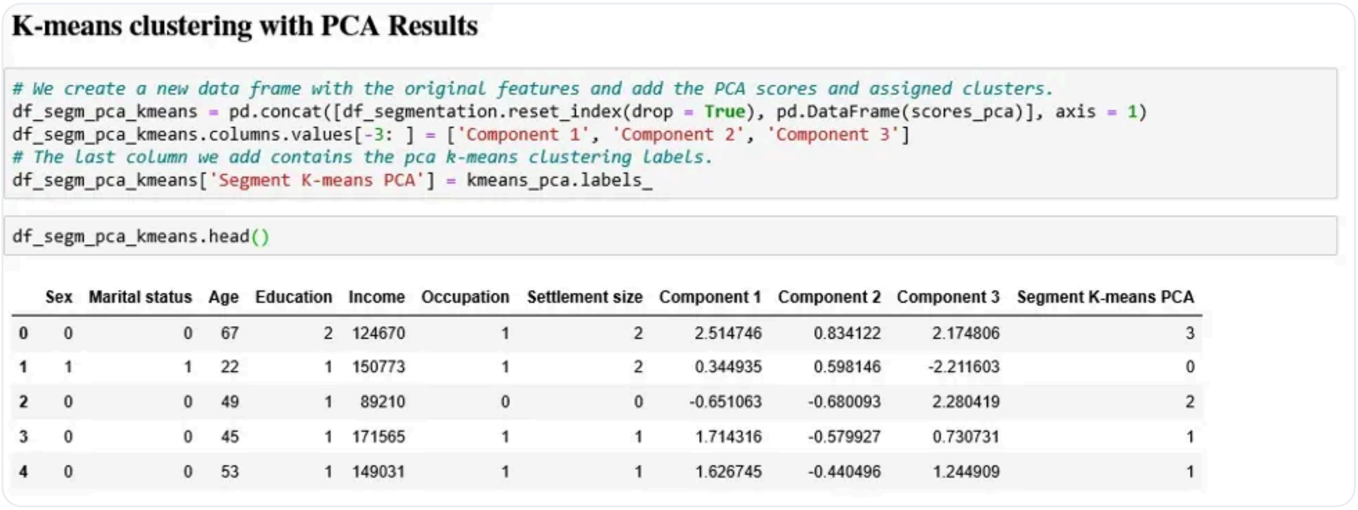
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
```

Here, we use the same initializer and random state as before. Subsequently, we fit the model with the principal component scores.

And now we've come to the most interesting part: analyzing the results of our algorithm.

5. How to Analyze the Results of PCA and K-Means Clustering

Before all else, we'll create a new data frame. It allows us to add in the values of the separate components to our segmentation data set. The components' scores are stored in the 'scores P C A' variable. Let's label them Component 1, 2 and 3. In addition, we also append the 'K means P C A' labels to the new data frame.



K-means clustering with PCA Results

```
# We create a new data frame with the original features and add the PCA scores and assigned clusters.
df_segm_pca_kmeans = pd.concat([df_segmentation.reset_index(drop = True), pd.DataFrame(scores_pca)], axis = 1)
df_segm_pca_kmeans.columns.values[-3: ] = ['Component 1', 'Component 2', 'Component 3']
# The last column we add contains the pca k-means clustering labels.
df_segm_pca_kmeans['Segment K-means PCA'] = kmeans_pca.labels_
```

```
df_segm_pca_kmeans.head()
```

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size	Component 1	Component 2	Component 3	Segment K-means PCA
0	0	0	67	2	124670	1	2	2.514746	0.834122	2.174806	3
1	1	1	22	1	150773	1	2	0.344935	0.598146	-2.211603	0
2	0	0	49	1	89210	0	0	-0.651063	-0.680093	2.280419	2
3	0	0	45	1	171565	1	1	1.714316	-0.579927	0.730731	1
4	0	0	53	1	149031	1	1	1.626745	-0.440496	1.244909	1

We're all but ready to see the results of our labor.

One small step remains: we should add the names of the segments to the labels.

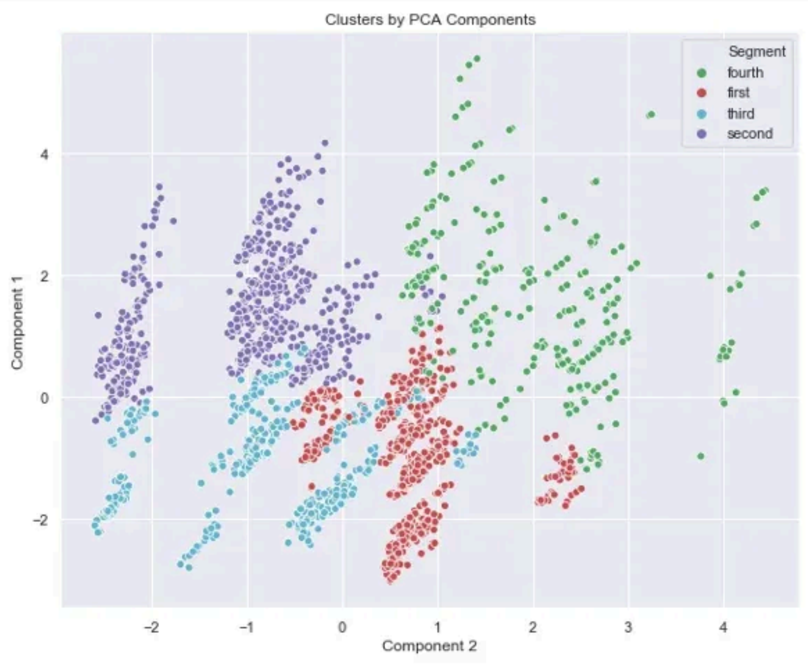
We create a new column named 'Segment' and map the four clusters directly inside it.

6. How to Visualize Clusters by Components?

Let’s finish off by visualizing our clusters on a 2D plane. It's a 2D visualization, so we need to choose two components and use them as axes. The point of PCA was to determine the most important components. This way, we can be absolutely sure that the first two components explain more variance than the third one.

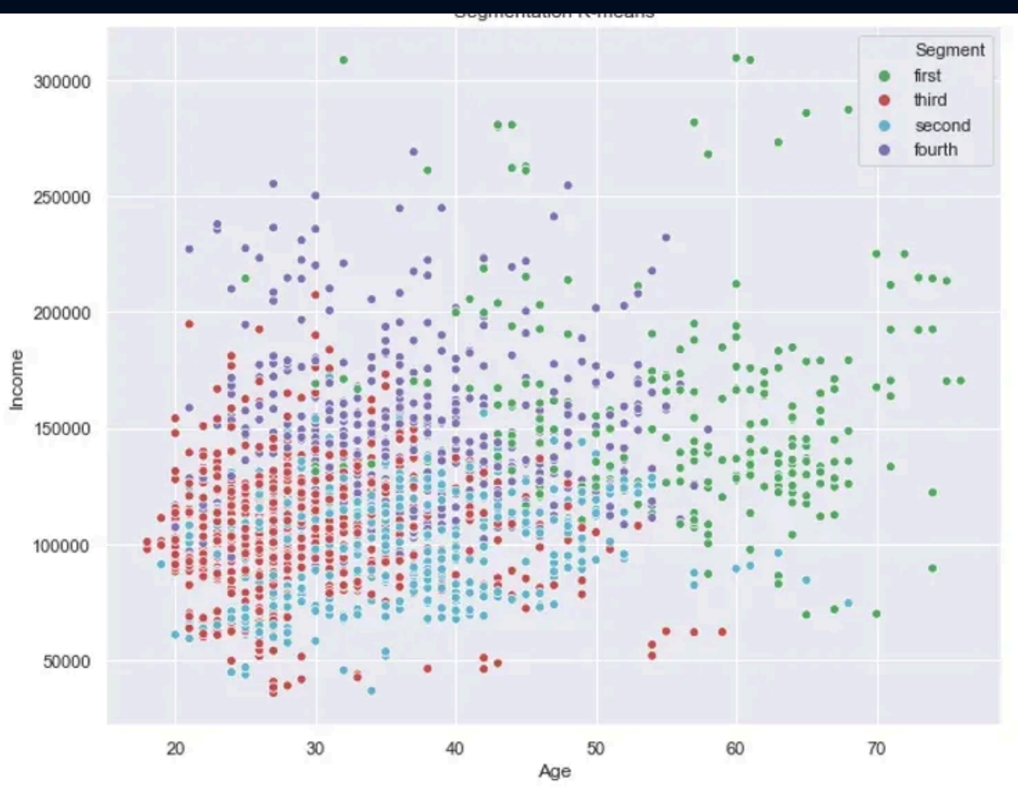
So, let’s visualize the segments with respect to the first two components.

```
# Plot data by PCA components. The Y axis is the first component, X axis is the second.
x_axis = df_segm_pca_kmeans['Component 2']
y_axis = df_segm_pca_kmeans['Component 1']
plt.figure(figsize = (10, 8))
sns.scatterplot(x_axis, y_axis, hue = df_segm_pca_kmeans['Segment'], palette = ['g', 'r', 'c', 'm'])
plt.title('Clusters by PCA Components')
plt.show()
```



The X-axis here is our ‘Component 2’. The y-axis, on the other hand, is the first ‘Component 1’.

We can now observe the separate clusters.



In this instance, only the green cluster is visually separated from the rest. The remaining three clusters are jumbled all together.

However, when we employ PCA prior to using K-means we can visually separate almost the entire data set. That was one of the biggest goals of PCA - to reduce the number of variables by combining them into bigger, more meaningful features.

Not only that, but they are 'orthogonal' to each other. This means that the difference between components is as big as possible.

There is some overlap between the red and blue segments. But, as a whole, all four segments are clearly separated. The spots where the two overlap are ultimately determined by the third component, which is not available on this graph.

Combining PCA and K-Means Clustering: Overview

Finally, it is important to note that our data set contained only a few features from the get-go. So, when we further reduced the dimensionality, using 'P C

That’s the reason why even a two-dimensional plot is enough to see the separation.

This might not always be the case. You may have more features and more components respectively. Then you might need a different way to represent the results of PCA.

We hope you’ll find this tutorial helpful and try out a K-means and PCA approach using your own data.

If you’re interested in more practical insights into Python, check out our step-by-step Python tutorials.

In case you’re new to Python, this comprehensive article on learning Python programming will guide you all the way. From the installation, through Python IDEs, Libraries, and frameworks, to the best Python career paths and job outlook.

Ready to Take the Next Step Towards a Data Science Career?

Elitsa Kaloyanova
Instructor at 365 Data Science

Elitsa is a Computational Biologist with a strong Bioinformatics background. Her courses in the 365 Data Science Program - Data Visualization, Customer Analytics, and Fashion Analytics - have helped thousands of students master the most in-demand data science tools and enhance their practical skillset. In her spare time, apart from writing expert publications, Elitsa loves hiking and windsurfing.

We Think You'll Also Like

Python Tutorials

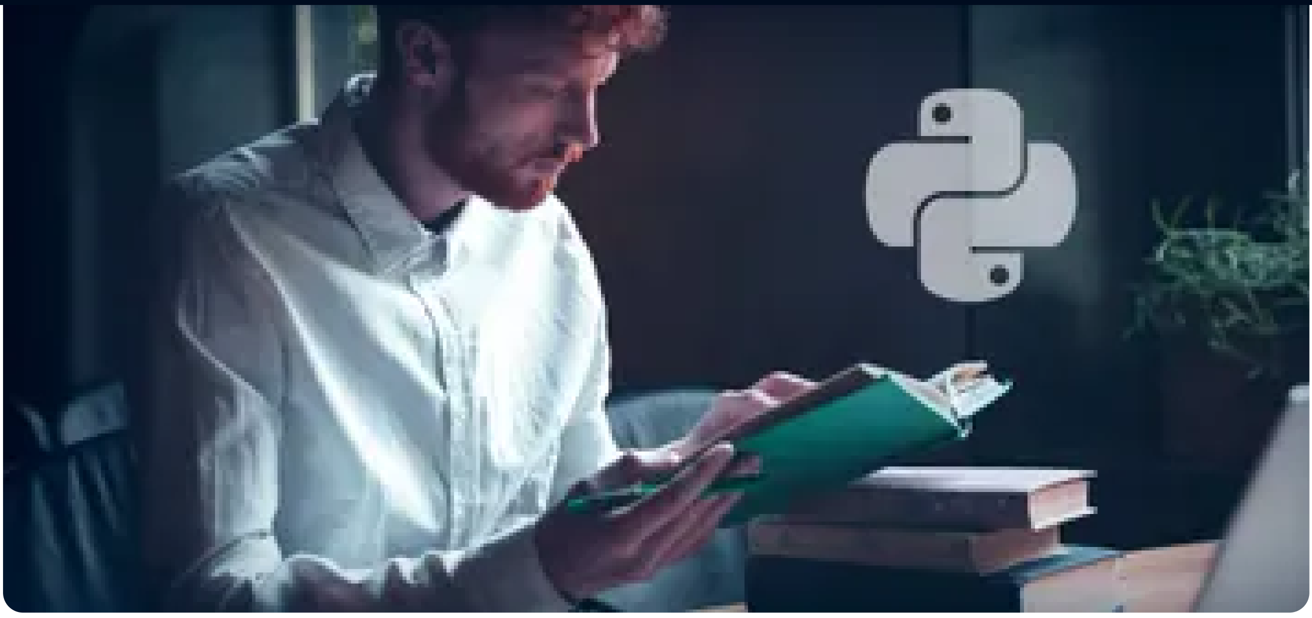
What Is the Difference Between PCA and LDA?

by **Eugenia Anello** • 10 min read

Python Tutorials

Why Python for Data Science and Why Use Jupyter Notebook to Code in Python

by **Martin Ganchev** • 11 min read



Python Tutorials

10 Best Python Books for Beginners and Skilled Programmers

7 min read

Python Tutorials

Combining Python Conditional Statements and Functions Exercise

by **Martin Ganchev** • 1 min read

365DataScience

365DataScience

All the Data Science Courses You Need

Platform

About

Learn

Resources

Success stories

About Us

Courses

Course Notes

Q&A Hub

Meet the Instructors

Career Tracks

Templates

Help Center

Career Certificate

Career Guides

Verify Certificate

Become an Instructor

Course Certificate

Practice Exams

Calculators

Contact Us

Flashcards

Pricing

Blog

Sitemap Terms of Use Privacy Policy Cookies