Published in Towards Data Science

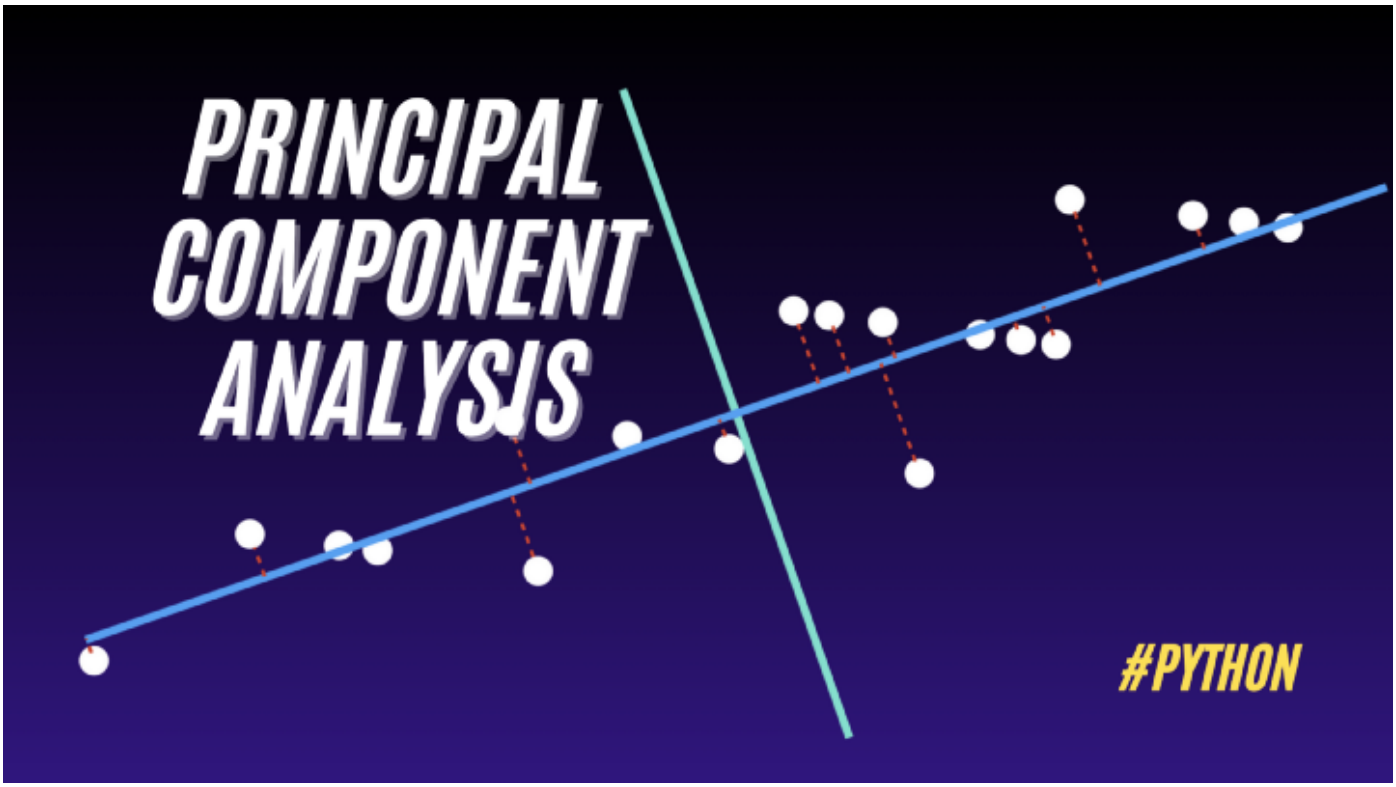
Saul Dobilas Follow

Jul 25, 2021 · 8 min read ·

MACHINE LEARNING

PCA: Principal Component Analysis — How to Get Superior Results with Fewer Dimensions?

One of The Best Techniques for Dimensionality Reduction



Principal Component Analysis. Image by author.

In this article, I explain how PCA works and give you an example of how to perform such analysis in Python.

Contents

What category of Machine Learning techniques does Principal Component Analysis (PCA) belong to?

While PCA is often referred to as a dimensionality reduction technique, it is actually a data transformation.

Nevertheless, PCA makes it very easy to use the resulting principal components to reduce the number of dimensions as it ranks them from “most useful” (captures a lot of the data variance) to “least useful” (captures very little of the data variance).

Hence, there is really no harm in putting it under the dimensionality reduction group of algorithms within the unsupervised learning branch of ML.

The below graph is **interactive,** so please click on different categories to **enlarge and reveal more** 👇 .

Machine Learning algorithm classification. Interactive chart created by the author.

If you enjoy Data Science and Machine Learning, please subscribe to get an email whenever I publish a new story.

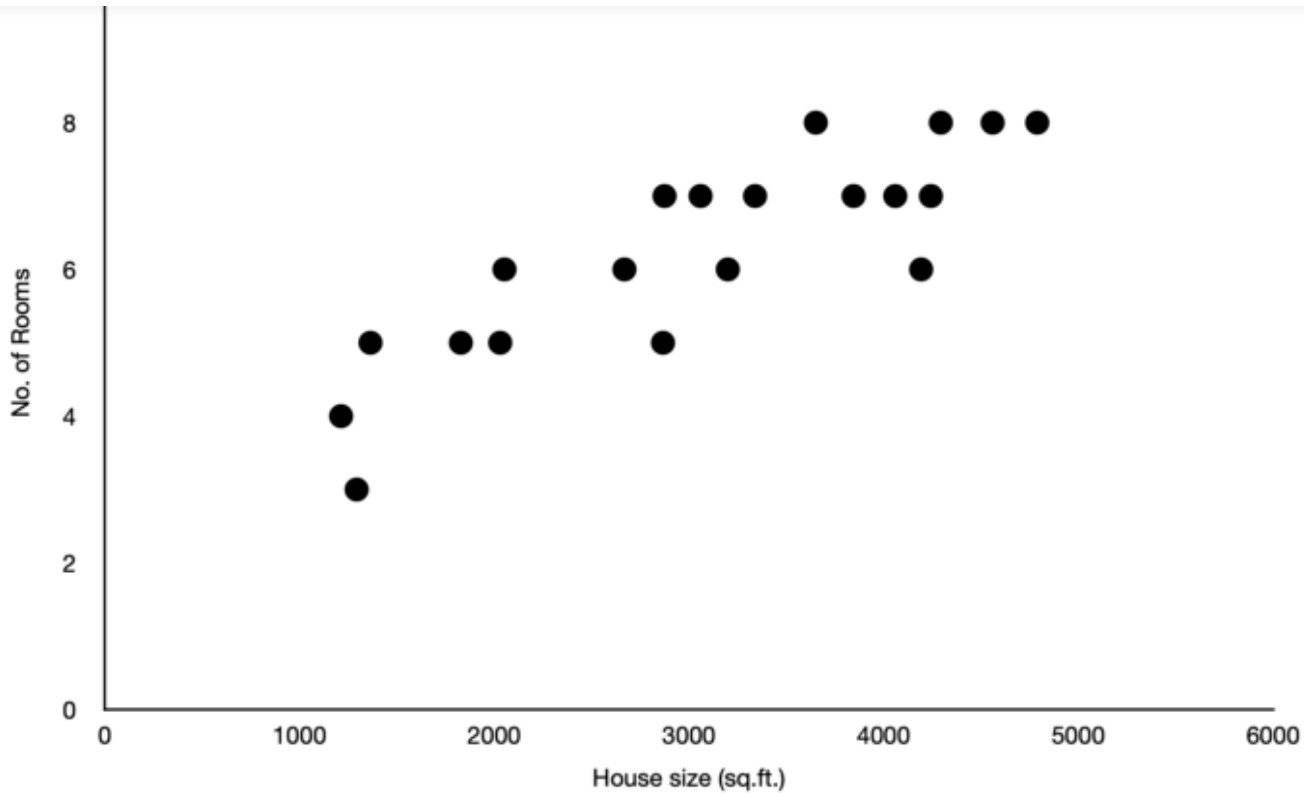
How does Principal Component Analysis (PCA) work?

Preparing the data

In simple terms, PCA helps us find new axes (principal components) of our dimensions that can better capture the variance of the data.

For example, you may have two attributes — house size (sq.ft.) and the number of rooms in that house. Not surprisingly, these two attributes are highly correlated as bigger houses tend to have more rooms.

Let's visualize our made-up example by plotting the two attributes on a 2D scatterplot:



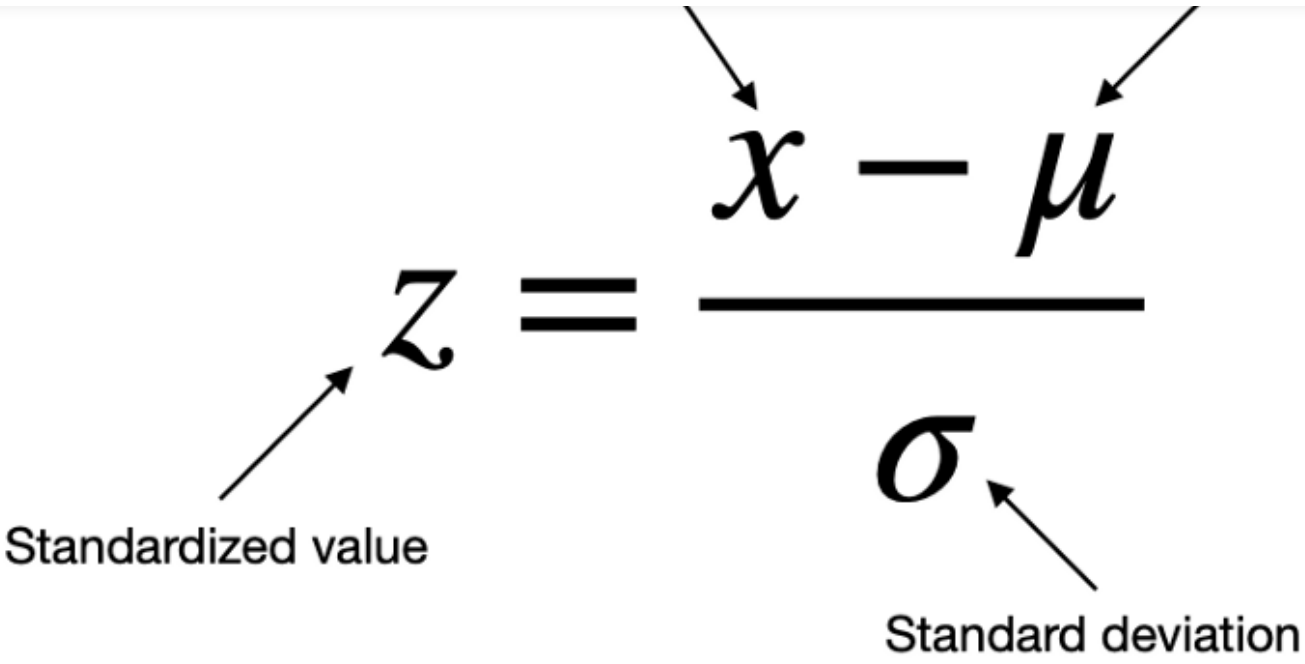
Relationship between house size and the number of rooms. Image by author.

We can see that there is a positive correlation between house size and the number of rooms. Also, there seems to be more variance in house sizes when compared to the number of rooms.

However, before we make any premature conclusions, let's pause for a minute and take a look at the scale of x and y. We can see that these two attributes use two different scales; hence the above picture is not really representative.

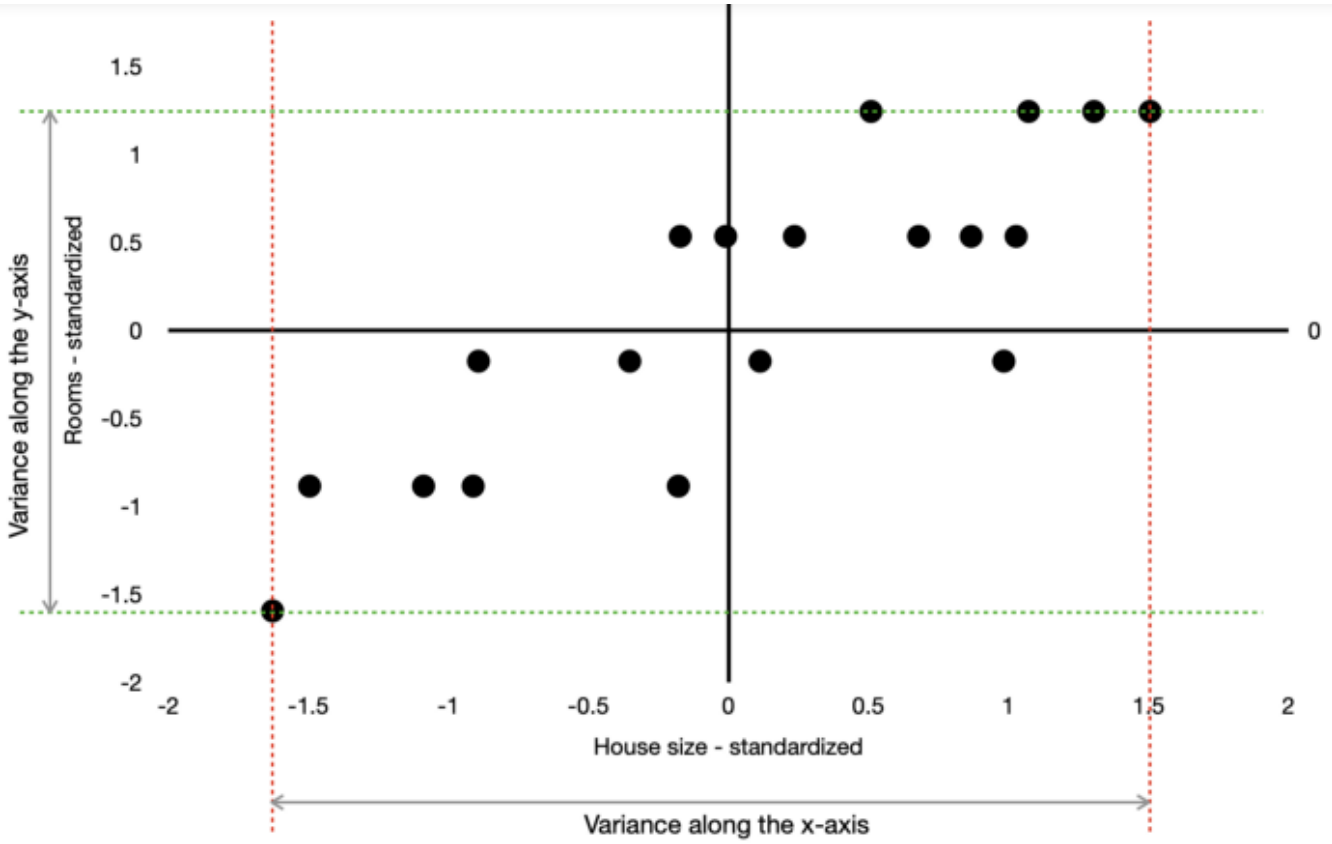
To make a more objective interpretation, let's standardize our data.

> *Note, standartization is a data transformation technique that rescales the data so each attribute has a mean of 0 and a standard deviation of 1. This transformation can be described with the below formula:*



Standardization. Image by author.

So, this is what the same data looks like after we standardized it:



Relationship between house size and the number of rooms (standardized data). Image by author.

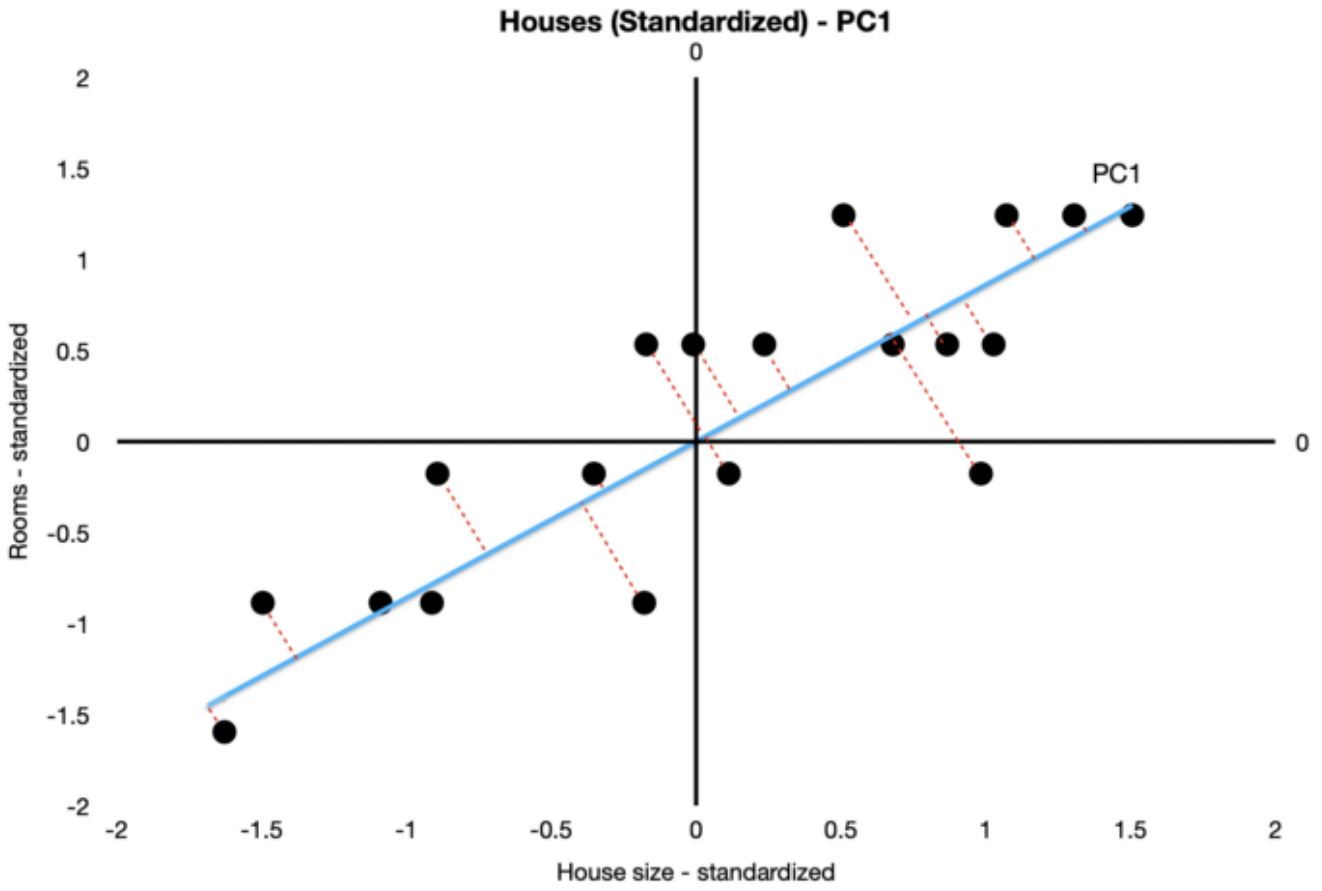
We can see that the data is now centered around the origin because both standardized attributes have a mean of 0. At the same time, it is now easier to visually compare the spread (variance) of the two attributes. As previously speculated, there is indeed a wider spread in house sizes than the number of rooms.

Finding principal components — PC1

I will intentionally avoid going into the mathematics behind finding principal components because my goal is to give you an **intuitive/visual** understanding of how PCA works.

If you do enjoy a bit of algebra and, in particular, matrix multiplications, you can check out this story from Zichen Wang on PCA and SVD explanation using NumPy.

Now that we have the data standardized and centered around the origin, we can look



"Best-fit" line of the data, which is also axis for PC1. Image by author.

Interestingly, this "best-fit" line also happens to be the axis for Principal Component 1 (PC1). This is because minimizing the distances from the line also **maximizes the spread of data point projections on that same line.**

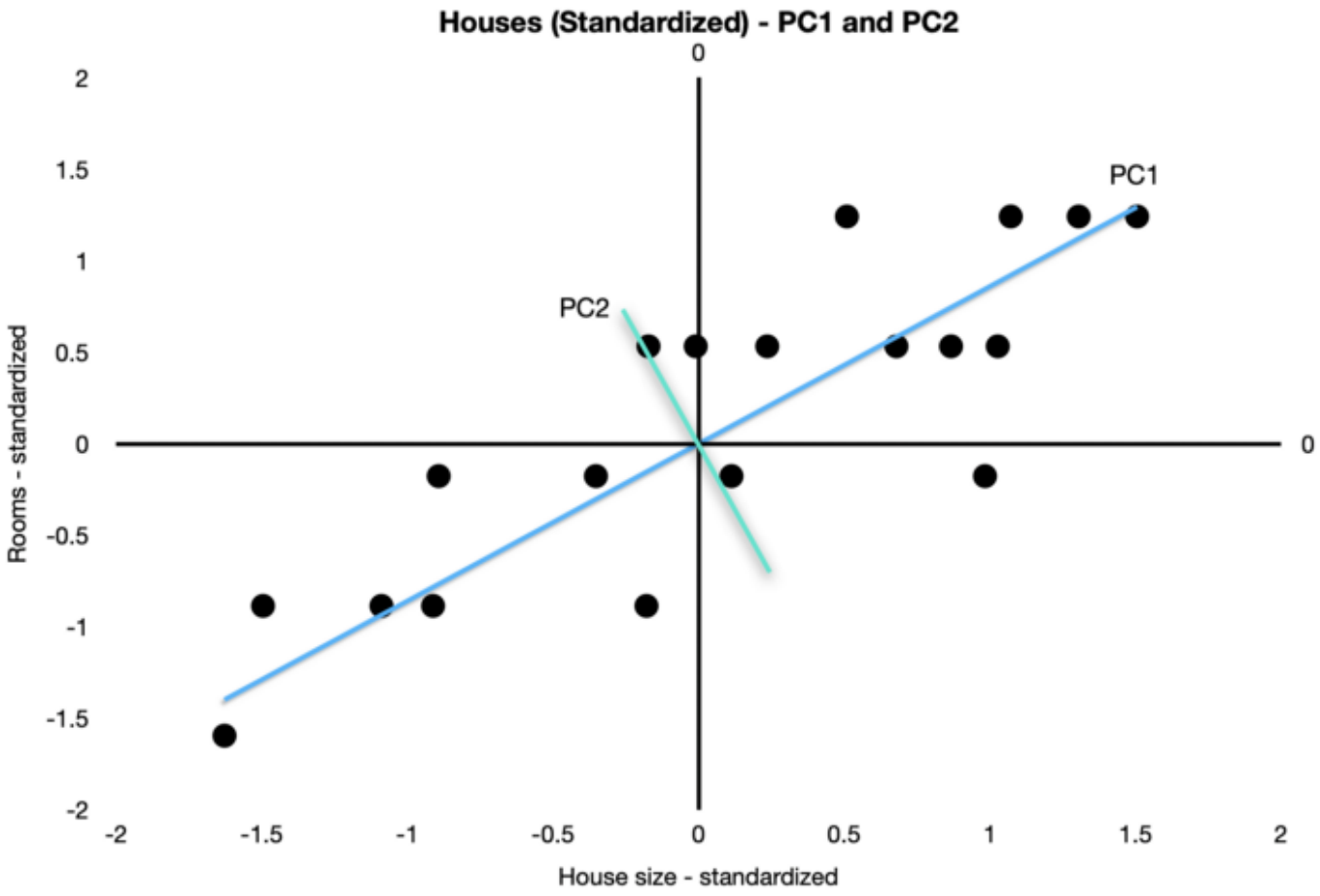
In other words, we have found a new axis that captures the maximum amount of variance of the data in that dimension.

If you decide to study the mathematics of this process separately, you will come across terms like **eigenvectors** and **eigenvalues.** They, in a sense, describe this "best-fit" line and are mathematically found through doing **eigendecomposition** of the covariance matrix or through **Singular Value Decomposition** analysis (SVD for short).

in our data. Said that, let's find PC2.

Since we only have two attributes, in this example, it is straightforward to find PC2 once we know PC1. It is simply an orthogonal (90 degrees angle) line to PC1 that goes through the origin.

Here it is on the graph:



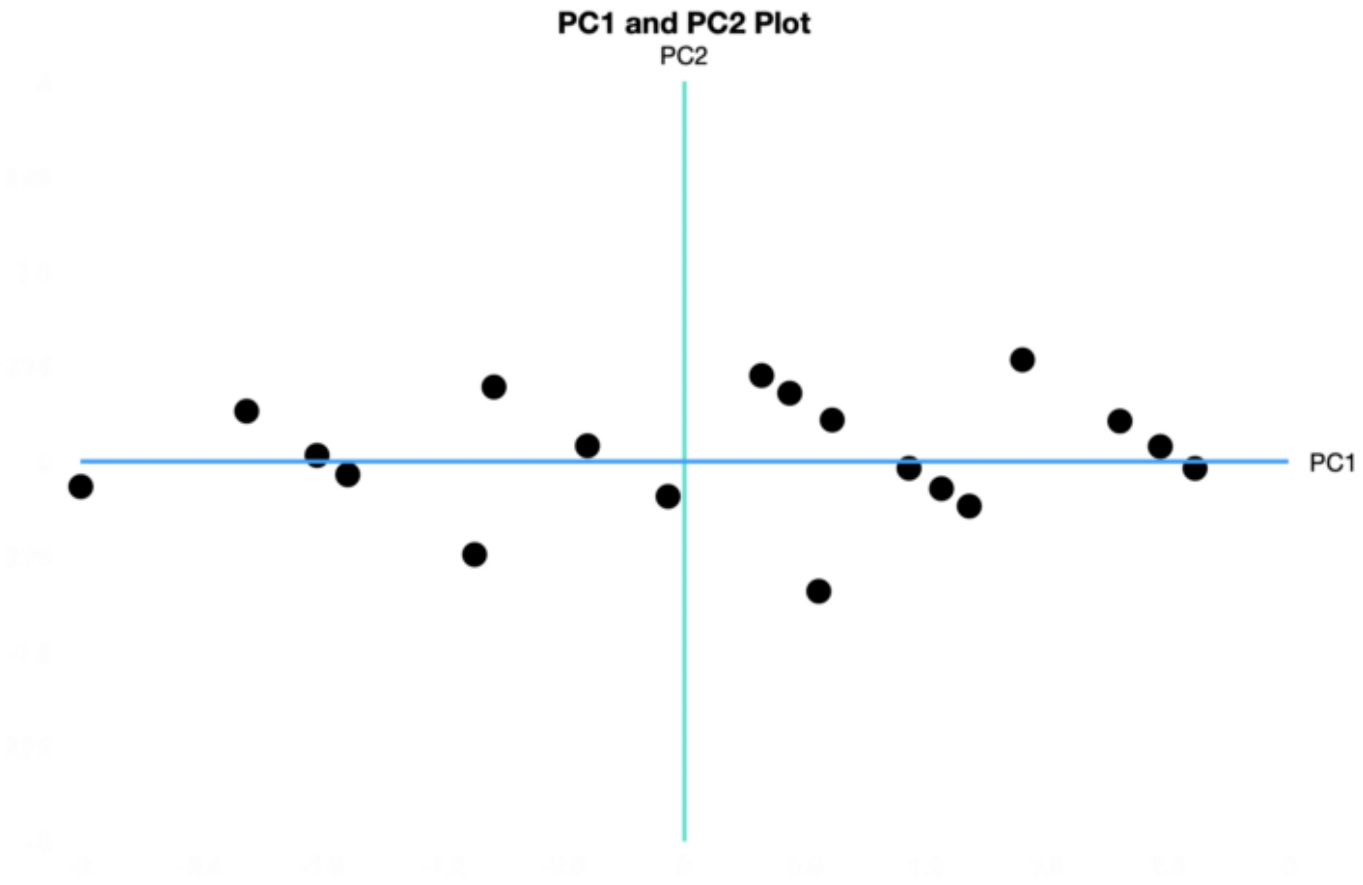
Principal Component 2 (PC2). Image by author.

Note, if we had three attributes, then PC2 would be the line that goes through the origin, is orthogonal to PC1, and minimizes distances from data points to their projections on the PC2 line. Then PC3 would be the line that goes through the origin and is orthogonal to both PC1 and PC2.

captures the highest proportion of the **remaining** variance that PC1 could not capture. Similarly, PC3 would be the dimension capturing the highest proportion of the remaining variance that PC1 and PC2 could not capture, etc.

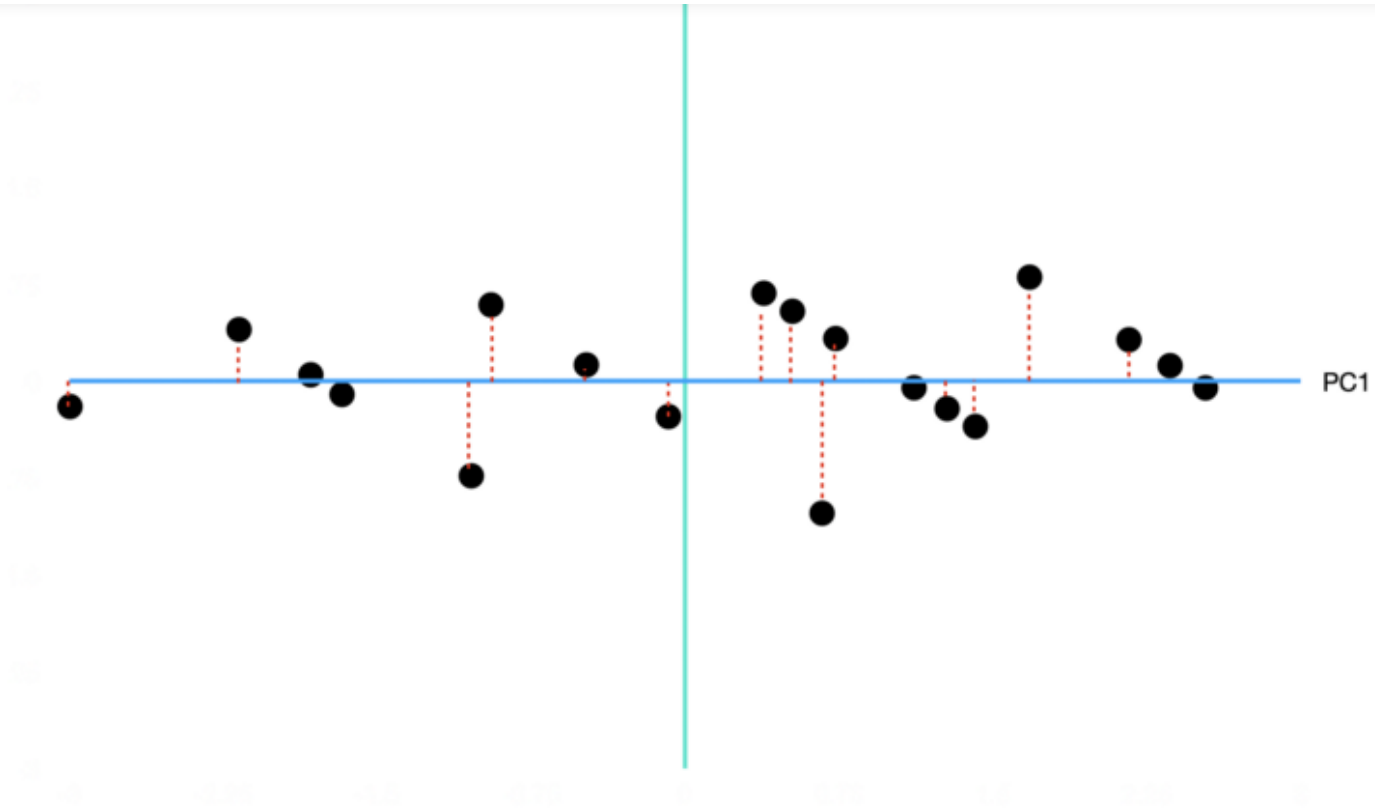
> The goal of PCA is to transform the data in a way that enables us to capture the maximum amount of variance in each subsequent dimension.

Now that we have found PC1 and PC2 in our two-dimensional example, we can rotate the graph and make PC1 and PC2 our new axes:



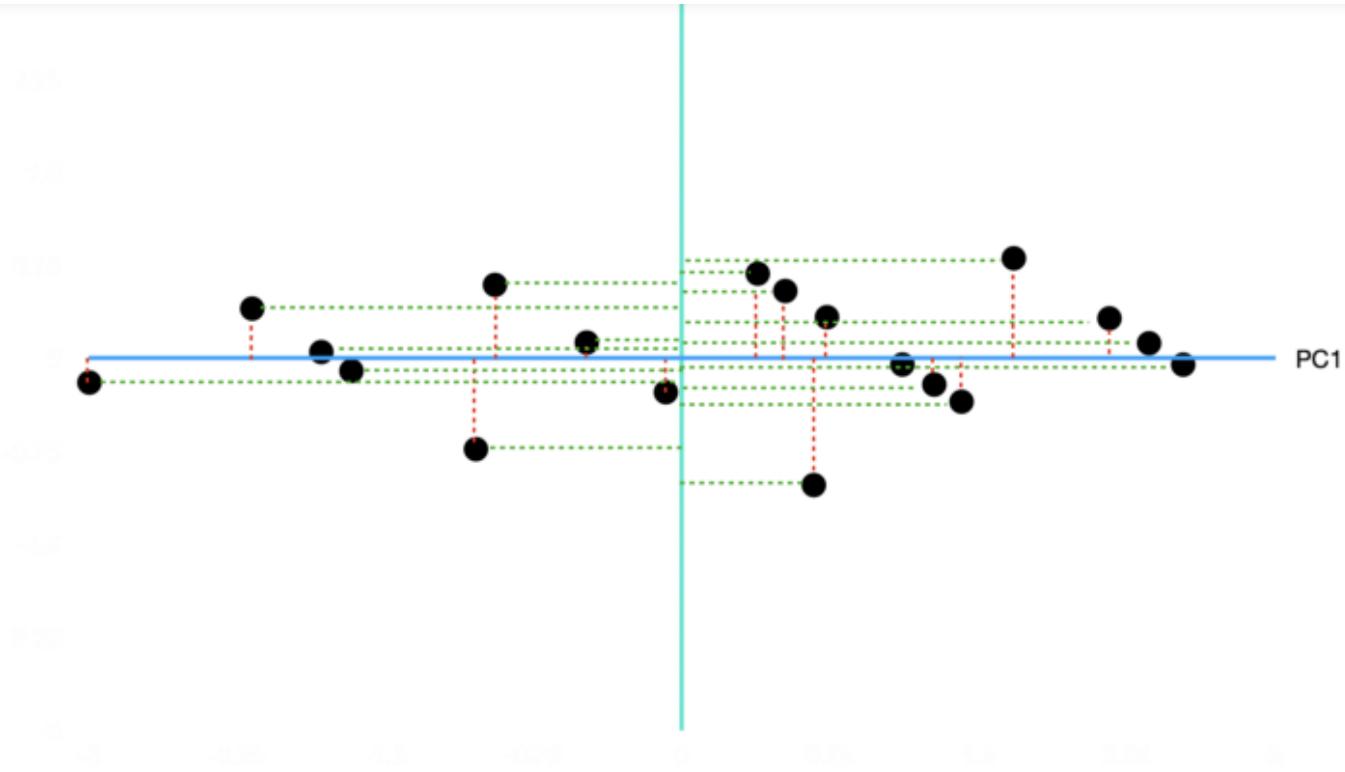
PC1 — PC2 plot. Image by author.

Also, if we wish, we can perform dimensionality reduction by projecting points onto



Projecting data points onto PC1. Image by author.

Finally, if, for whatever reason, we did not like PC1, we could reduce dimensions by projecting points onto PC2 instead, enabling us to get rid of PC1. However, we would lose a lot of the data variance if we did that.



Projecting data points onto PC1 and PC2. Image by author.

Principal Component Analysis in Python using real-life data

Let's now get our hands dirty and perform PCA on real-life data.

Setup

We will use the following data and libraries:

- Australian weather data from Kaggle
- scikit-learn's StandardScaler for standardizing our data and PCA for performing Principal Component Analysis

Let's import all the libraries:

Then, we download and ingest Australian weather data from Kaggle. We also derive a couple of new features at the same time.

Here’s a snippet of what the data looks like:

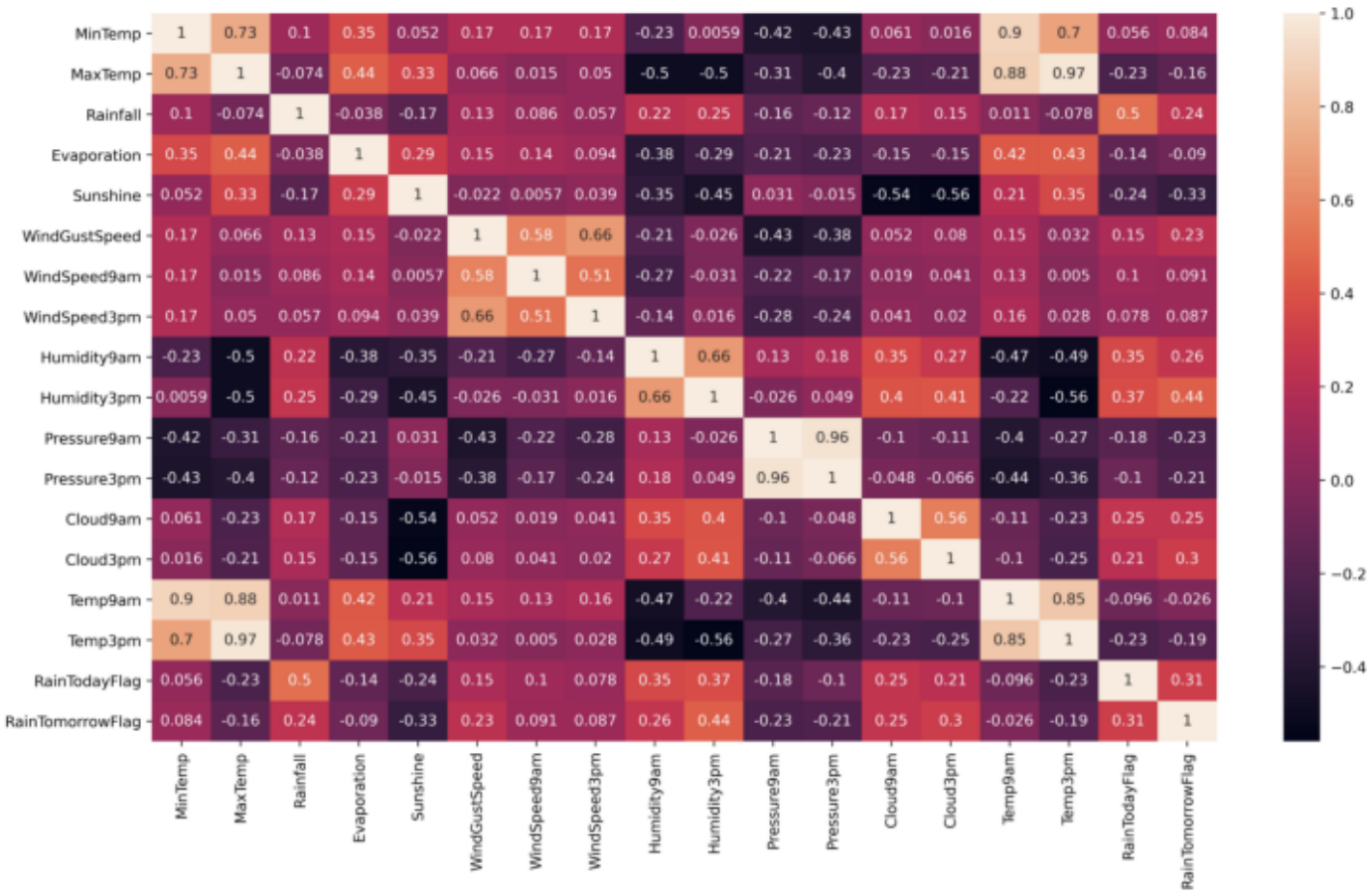
1	2008-12-02	Albury	7.4	25.1	0.0	5.469824	7.624853	WNW	44.0	NNW	WSW	4.0
2	2008-12-03	Albury	12.9	25.7	0.0	5.469824	7.624853	WSW	46.0	W	WSW	19.0
3	2008-12-04	Albury	9.2	28.0	0.0	5.469824	7.624853	NE	24.0	SE	E	11.0
4	2008-12-05	Albury	17.5	32.3	1.0	5.469824	7.624853	W	41.0	ENE	NW	7.0
...	...	...	...	...	...	...	...	...	...	...	...	...
145454	2017-06-20	Uluru	3.5	21.8	0.0	5.469824	7.624853	E	31.0	ESE	E	15.0
145455	2017-06-21	Uluru	2.8	23.4	0.0	5.469824	7.624853	E	31.0	SE	ENE	13.0
145456	2017-06-22	Uluru	3.6	25.3	0.0	5.469824	7.624853	NNW	22.0	SE	N	13.0
145457	2017-06-23	Uluru	5.4	26.9	0.0	5.469824	7.624853	N	37.0	SE	WNW	9.0
145458	2017-06-24	Uluru	7.8	27.0	0.0	5.469824	7.624853	SE	28.0	SSE	N	13.0

142193 rows × 25 columns

A snippet of Kaggle's Australian weather data. Image by author.

Data Correlation

Before performing PCA analysis, let's better understand our data by looking at the correlation plot.



Correlation matrix of the weather data. Image by author.

As we can see, we have plenty of highly correlated variables. For example, MaxTemp is highly negatively correlated to Humidity at 9 am and 3 pm. At the same time, WindGustSpeed is highly positively correlated to WindSpeed at 9 am and 3 pm.

A strong correlation between many variables indicates that PCA will help us capture a large amount of variance within a potentially much smaller number of dimensions.

Performing PCA

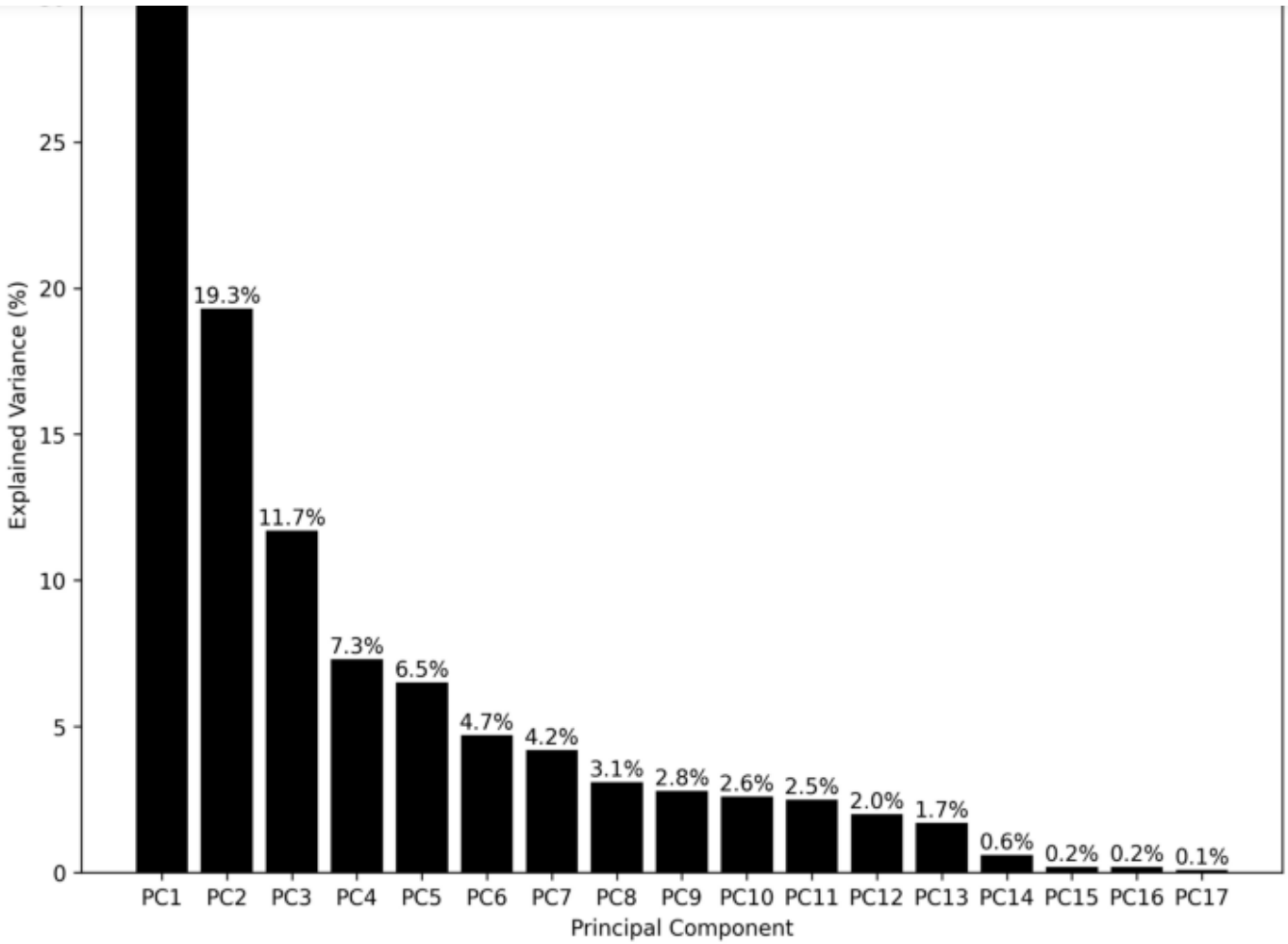
Finally, let's use the algorithm to perform Principal Component Analysis.

The gives us the following results:

```
*************** PCA Summary ***************
No. of features:  17
No. of samples:  142193
No. of components:  17
Explained variance ratio:  [0.30473734 0.19327251 0.11661159 0.07343159 0.06477021 0.04686651
 0.04244543 0.03124602 0.0283     0.02562401 0.02453296 0.01950429
 0.01702357 0.00621854 0.00223226 0.0017376  0.00144558]
```

PCA results. Image by author.

The explained variance ratio tells us how much of the variance has been captured by each subsequent principal component. Let's plot it on a bar chart so we can inspect it



PCA explained variance plot. Image by author.

The plot enables us to easily see that the majority of variance is captured just by the first few PCs:

- PC1 alone captures 31% of the variance
- Top 2 PCs capture 50% of the variance
- Top 6 PCs capture 80% of the variance
- Top 11 PCs capture 95% of the variance
- Top 13 PCs capture 99% of the variance

So, if you are building a prediction model, you can definitely make the training more

> *Note, you can do the selection of the top few components by simply taking a subset of the* ***X_trans*** *array, or rerunning the PCA analysis by putting* ***n_components****=6 or whatever number of dimensions you want to keep.*

Conclusions

We have learned how the Principal Component Analysis works and how to apply it to our data. I sincerely hope you can use this new knowledge and the code I've provided to build better, more efficient models.

Before you go, one last interesting idea to keep in mind. Despite losing some amount of data variance by reducing the number of dimensions, it often results in **better** model performance.

While this may sound counterintuitive, it is worth remembering that some prediction algorithms suffer from the curse of dimensionality, meaning that too many dimensions make the data sparse and harder to model. Hence, reducing the number of dimensions helps to identify the connections between the attributes leading to improved performance.

If you found this article useful or have any questions or suggestions, please do not hesitate to reach out.

Cheers! 👏
Saul Dobilas

A couple of related articles you may find interesting:

UMAP Dimensionality Reduction — An Incredibly Robust Machine Learning Algorithm

How does Uniform Manifold Approximation and Projection (UMAP) work, and how to use it in Python

towardsdatascience.com

DBSCAN Clustering Algorithm — How to Build Powerful Density-Based Models

A detailed guide to using Density-Based Spatial Clustering of Applications with Noise

towardsdatascience.com