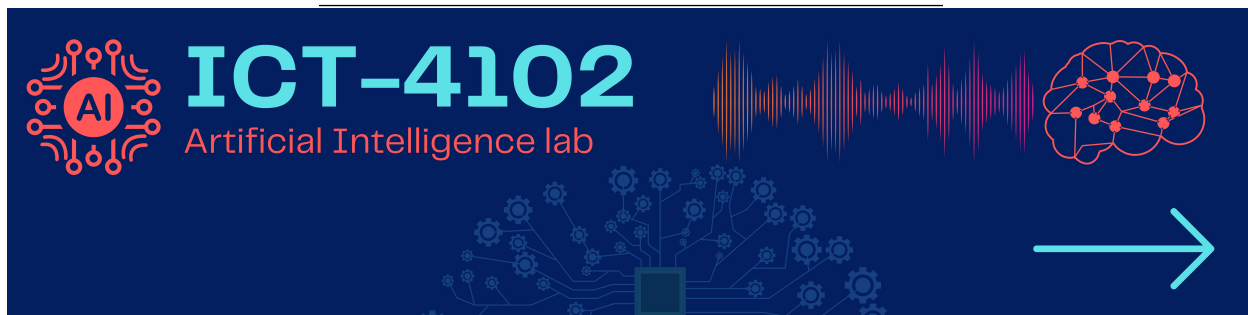


01-Principal Component Analysis

September 5, 2023



1 Principal Component Analysis

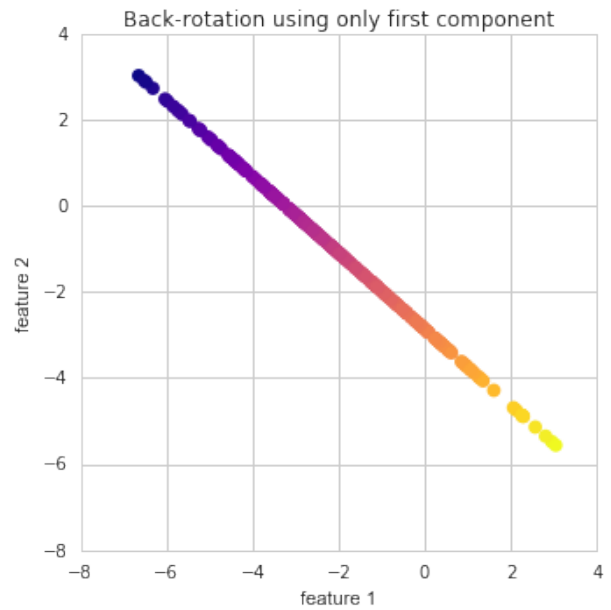
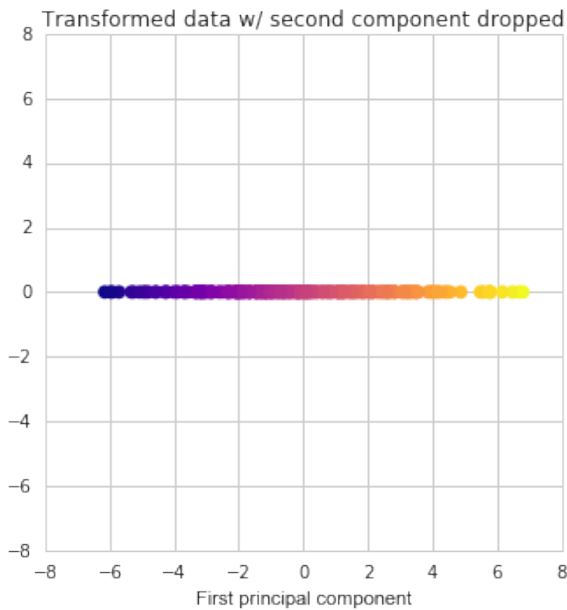
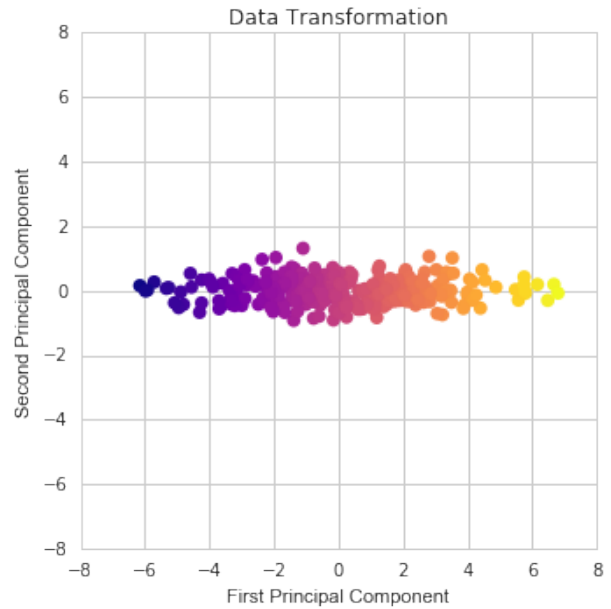
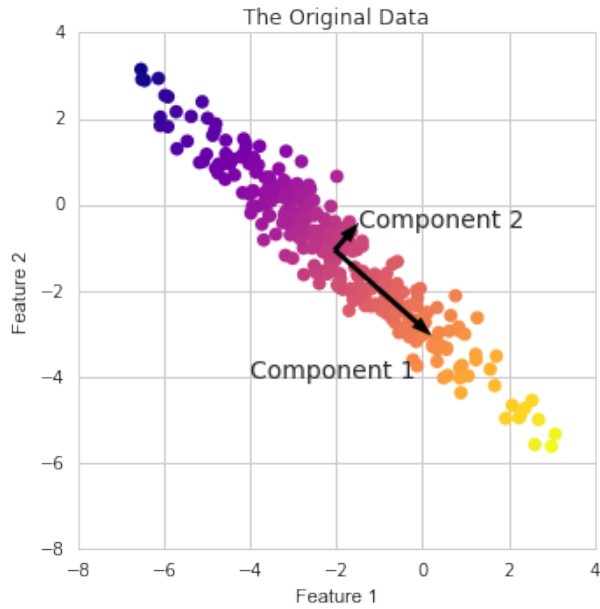
Let's discuss PCA! Since this isn't exactly a full machine learning algorithm, but instead an unsupervised learning algorithm.

1.1 Introduction :






Principal component analysis (PCA) is a popular technique for analyzing large datasets containing a high number of dimensions/features per observation, increasing the interpretability of data while preserving the maximum amount of information, and enabling the visualization of multidimensional data.

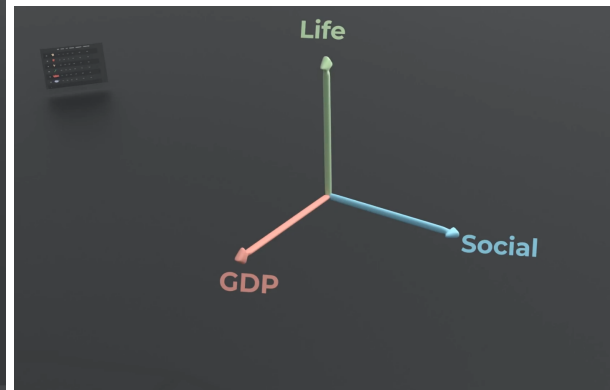
Formally, PCA is a statistical technique for reducing the dimensionality of a dataset. This is accomplished by linearly transforming the data into a new coordinate system where (most of) the variation in the data can be described with fewer dimensions than the initial data. Many studies use the first two principal components in order to plot the data in two dimensions and to visually identify clusters of closely related data points.

Principal component analysis is the process of computing the principal components and using them to perform a change of basis on the data, sometimes using only the first few principal components and ignoring the rest.

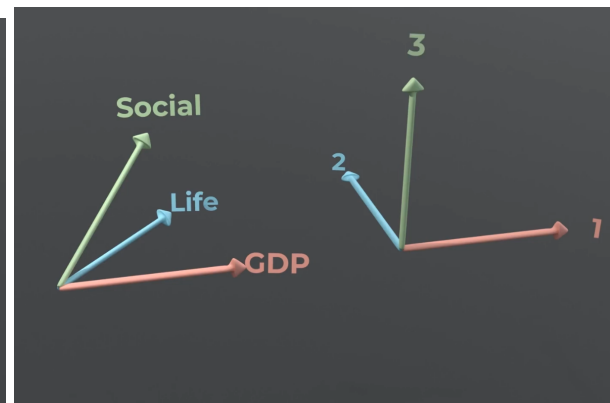
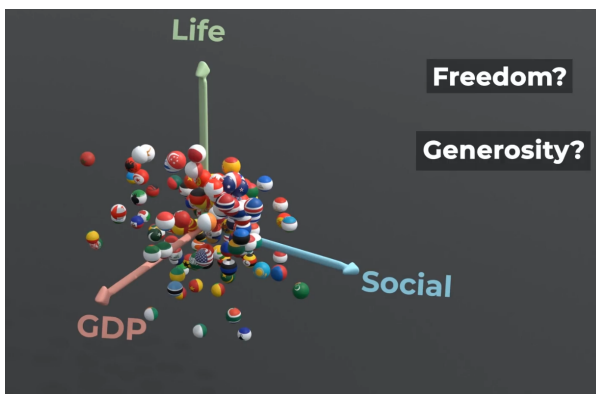


In data analysis, the first principal component of a set of p variables, presumed to be jointly normally distributed, is the derived variable formed as a linear combination of the original variables that explains the most variance. The second principal component explains the most variance in what is left once the effect of the first component is removed, and we may proceed through p iterations until all the variance is explained. PCA is most commonly used when many of the variables are highly correlated with each other and it is desirable to reduce their number to an independent set.

		GDP	SOCIAL	LIFE	FREEDOM	GENEROSITY	CORRUPTION
FR		1.1	1.1	1.3	0.3	-0.9	-0.9
DE		1.2	0.8	1.1	0.7	0.2	-1.5
IN		-0.6	-1.8	-0.6	0.9	0.7	0.3
MA		-0.5	-2.2	0.2	-0.2	-1.5	0.4
TR		0.7	0.1	0.3	-1.9	-0.8	0.3
US		1.4	0.9	0.5	0.4	0.8	-0.2



From the above figure we see that after changing to three dimensions we may lose some of the features .



PCA is used in exploratory data analysis and for making predictive models. It is commonly used for dimensionality reduction by projecting each data point onto only the first few principal components to obtain lower-dimensional data while preserving as much of the data's variation as possible. The first principal component can equivalently be defined as a direction that maximizes the variance of the projected data. The i -th principal component can be taken as a direction orthogonal to the first $i-1$ th principal components that maximizes the variance of the projected data.

1.2 PCA Review

Remember that PCA is just a transformation of your data and attempts to find out what features explain the most variance in your data.

1.3 Libraries

```
[61]: import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
```

1.4 The Data

Let's work with the cancer data set again since it had so many features.

```
[62]: from sklearn.datasets import load_breast_cancer

[63]: cancer = load_breast_cancer()

[64]: cancer.keys()

[64]: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
               'filename', 'data_module'])

[ ]: print(cancer['DESCR'])

[66]: type(cancer['DESCR'])

[66]: str

[67]: df = pd.DataFrame(cancer['data'], columns=cancer['feature_names'])
      #(['DESCR', 'data', 'feature_names', 'target_names', 'target'])

[68]: df.head(3)
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	\
0	17.99	10.38	122.8	1001.0	0.11840	
1	20.57	17.77	132.9	1326.0	0.08474	
2	19.69	21.25	130.0	1203.0	0.10960	

	mean compactness	mean concavity	mean concave points	mean symmetry	\
0	0.27760	0.3001	0.14710	0.2419	
1	0.07864	0.0869	0.07017	0.1812	
2	0.15990	0.1974	0.12790	0.2069	

	mean fractal dimension	...	worst radius	worst texture	worst perimeter	\
0	0.07871	...	25.38	17.33	184.6	
1	0.05667	...	24.99	23.41	158.8	
2	0.05999	...	23.57	25.53	152.5	

	worst area	worst smoothness	worst compactness	worst concavity	\
0	2019.0	0.1622	0.6656	0.7119	
1	1956.0	0.1238	0.1866	0.2416	
2	1709.0	0.1444	0.4245	0.4504	

	worst concave points	worst symmetry	worst fractal dimension
0	0.2654	0.4601	0.11890
1	0.1860	0.2750	0.08902
2	0.2430	0.3613	0.08758

[3 rows x 30 columns]

1.5 PCA Visualization

As we've noticed before it is difficult to visualize high dimensional data, we can use PCA to find the first two principal components, and visualize the data in this new, two-dimensional space, with a single scatter-plot. Before we do this though, we'll need to scale our data so that each feature has a single unit variance.

```
[69]: from sklearn.preprocessing import StandardScaler
```

```
[70]: scaler = StandardScaler()  
      scaler.fit(df)
```

```
[70]: StandardScaler()
```

```
[71]: scaled_data = scaler.transform(df)
```

PCA with Scikit Learn uses a very similar process to other preprocessing functions that come with SciKit Learn. We instantiate a PCA object, find the principal components using the fit method, then apply the rotation and dimensionality reduction by calling transform().

We can also specify how many components we want to keep when creating the PCA object.

```
[72]: from sklearn.decomposition import PCA
```

```
[73]: pca = PCA(n_components=2)
```

```
[74]: pca.fit(scaled_data)
```

```
[74]: PCA(n_components=2)
```

Now we can transform this data to its first 2 principal components.

```
[75]: x_pca = pca.transform(scaled_data)
```

```
[76]: scaled_data.shape
```

```
[76]: (569, 30)
```

```
[77]: x_pca.shape
```

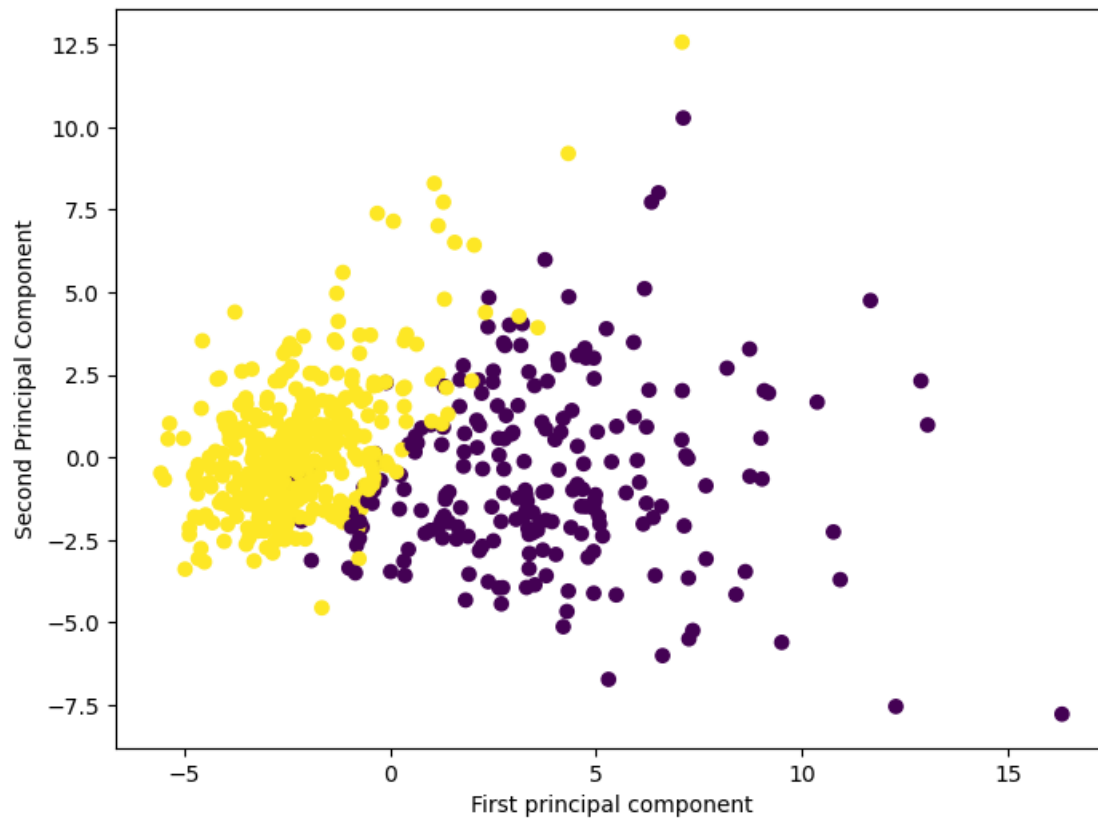
```
[77]: (569, 2)
```

Great! We've reduced 30 dimensions to just 2! Let's plot these two dimensions out!

```
[78]: plt.figure(figsize=(8,6))  
      plt.scatter(x_pca[:,0],x_pca[:,1],c=cancer['target'],cmap='viridis')  
      plt.xlabel('First principal component')
```

```
plt.ylabel('Second Principal Component')
```

```
[78]: Text(0, 0.5, 'Second Principal Component')
```



Clearly by using these two components we can easily separate these two classes.

1.6 Interpreting the components

Unfortunately, with this great power of dimensionality reduction, comes the cost of being able to easily understand what these components represent.

The components correspond to combinations of the original features, the components themselves are stored as an attribute of the fitted PCA object:

```
[79]: pca.components_
```

```
[79]: array([[ 0.21890244,  0.10372458,  0.22753729,  0.22099499,  0.14258969,
           0.23928535,  0.25840048,  0.26085376,  0.13816696,  0.06436335,
           0.20597878,  0.01742803,  0.21132592,  0.20286964,  0.01453145,
           0.17039345,  0.15358979,  0.1834174 ,  0.04249842,  0.10256832,
           0.22799663,  0.10446933,  0.23663968,  0.22487053,  0.12795256,
           0.21009588,  0.22876753,  0.25088597,  0.12290456,  0.13178394],
```

```
[-0.23385713, -0.05970609, -0.21518136, -0.23107671, 0.18611302,
 0.15189161, 0.06016536, -0.0347675, 0.19034877, 0.36657547,
 -0.10555215, 0.08997968, -0.08945723, -0.15229263, 0.20443045,
 0.2327159, 0.19720728, 0.13032156, 0.183848, 0.28009203,
 -0.21986638, -0.0454673, -0.19987843, -0.21935186, 0.17230435,
 0.14359317, 0.09796411, -0.00825724, 0.14188335, 0.27533947]])
```

In this numpy matrix array, each row represents a principal component, and each column relates back to the original features. we can visualize this relationship with a heatmap:

```
[92]: df_comp = pd.DataFrame(pca.components_, columns=cancer['feature_names'], index =_
↳ ['PC1', 'PC2'])
```

```
[93]: df_comp
```

```
[93]:      mean radius  mean texture  mean perimeter  mean area  mean smoothness \
PC1      0.218902      0.103725      0.227537      0.220995      0.142590
PC2     -0.233857     -0.059706     -0.215181     -0.231077      0.186113

      mean compactness  mean concavity  mean concave points  mean symmetry \
PC1      0.239285      0.258400      0.260854      0.138167
PC2      0.151892      0.060165      -0.034768      0.190349

      mean fractal dimension  ...  worst radius  worst texture \
PC1      0.064363  ...      0.227997      0.104469
PC2      0.366575  ...     -0.219866     -0.045467

      worst perimeter  worst area  worst smoothness  worst compactness \
PC1      0.236640      0.224871      0.127953      0.210096
PC2     -0.199878     -0.219352      0.172304      0.143593

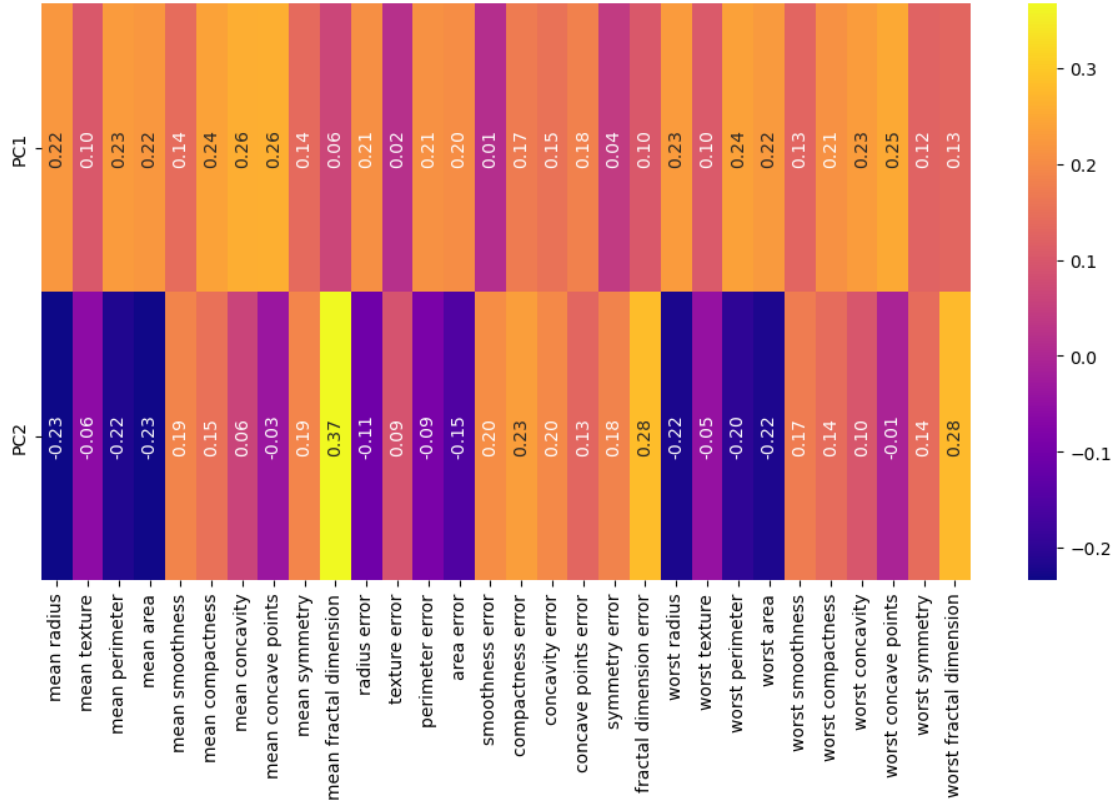
      worst concavity  worst concave points  worst symmetry \
PC1      0.228768      0.250886      0.122905
PC2      0.097964     -0.008257      0.141883

      worst fractal dimension
PC1      0.131784
PC2      0.275339

[2 rows x 30 columns]
```

```
[102]: plt.figure(figsize=(12,6))
sns.heatmap(df_comp,cmap="plasma",
            annot=True, fmt='.2f', annot_kws={'rotation': 90})
```

```
[102]: <Axes: >
```



This heatmap and the color bar basically represent the correlation between the various feature and the principal component itself.

1.7 Advantages of Principal Component Analysis

Dimensionality Reduction: Principal Component Analysis is a popular technique used for dimensionality reduction, which is the process of reducing the number of variables in a dataset. By reducing the number of variables, PCA simplifies data analysis, improves performance, and makes it easier to visualize data.

Feature Selection: Principal Component Analysis can be used for feature selection, which is the process of selecting the most important variables in a dataset. This is useful in machine learning, where the number of variables can be very large, and it is difficult to identify the most important variables.

Data Visualization: Principal Component Analysis can be used for data visualization. By reducing the number of variables, PCA can plot high-dimensional data in two or three dimensions, making it easier to interpret.

Multicollinearity: Principal Component Analysis can be used to deal with multicollinearity, which is a common problem in a regression analysis where two or more independent variables are highly correlated. PCA can help identify the underlying structure in the data and create new, uncorrelated variables that can be used in the regression model.

Noise Reduction: Principal Component Analysis can be used to reduce the noise in data. By removing the principal components with low variance, which are assumed to represent noise, Principal Component Analysis can improve the signal-to-noise ratio and make it easier to identify the underlying structure in the data.

Data Compression: Principal Component Analysis can be used for data compression. By representing the data using a smaller number of principal components, which capture most of the variation in the data, PCA can reduce the storage requirements and speed up processing.

Outlier Detection: Principal Component Analysis can be used for outlier detection. Outliers are data points that are significantly different from the other data points in the dataset. Principal Component Analysis can identify these outliers by looking for data points that are far from the other points in the principal component space.

1.8 Disadvantages of Principal Component Analysis

Interpretation of Principal Components: The principal components created by Principal Component Analysis are linear combinations of the original variables, and it is often difficult to interpret them in terms of the original variables. This can make it difficult to explain the results of PCA to others.

Data Scaling: Principal Component Analysis is sensitive to the scale of the data. If the data is not properly scaled, then PCA may not work well. Therefore, it is important to scale the data before applying Principal Component Analysis.

Information Loss: Principal Component Analysis can result in information loss. While Principal Component Analysis reduces the number of variables, it can also lead to loss of information. The degree of information loss depends on the number of principal components selected. Therefore, it is important to carefully select the number of principal components to retain.

Non-linear Relationships: Principal Component Analysis assumes that the relationships between variables are linear. However, if there are non-linear relationships between variables, Principal Component Analysis may not work well.

Computational Complexity: Computing Principal Component Analysis can be computationally expensive for large datasets. This is especially true if the number of variables in the dataset is large.

Overfitting: Principal Component Analysis can sometimes result in overfitting, which is when the model fits the training data too well and performs poorly on new data. This can happen if too many principal components are used or if the model is trained on a small dataset.

2 Great Job!