Principle Component Analysis (PCA)

Possible dimensionality reduction technique that is helpful to reduce data into a lower number of dimensions than the original input dimensions.

Was previously used to reduce data storage, but is now more commonly used to visualize data into a simple 1D, 2D, or even 3D plot.

Note: in this notebook, the words dimensions and features are interchangeable.

The axis to which we reduce the data to must capture a great deal of the variation in the original data. It must be informative enough to enable using the reduced data to make informative decisions.

After performing PCA, the goal a common technique is to reconstruct the original data from the reduced data set.

There are two main targets while conducting the Principle Component Analysis:

1- Minimize the reconstruction error 2- Maximize the variance of reconstructions

It can be proved that trying to achieve one of the two targets simultaneously gurantees the other.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer

# Load the breast cancer disgnostic dataset
dataset = load_breast_cancer()
features = dataset.data
target = dataset.target
```

To properly implement PCA, the data must be centered. Therefore, the mean will be subtracted from each data point. Moreover, the features will also be standardized as some of the given features might have a larger scale than others. If this point is not to be done, the resulting PCA could possibly be skewed.

```
# Centering and standardizing the data
# Subtract the mean from each feature and divide it by its standard
# Subtracting the mean --> centers the data
# Dividing by the standard deviation --> ensures proper scaling.

featuresStandardized = (features - np.mean(features, axis=0)) / np.
```

The next step is to compute the covariance matrix. The covariance matrix is essentially helpful as it helps visualize mathematically the relations between data and shows how features are correlated to each other. Helps in finding the direction having the most variation.

```
# Computing the covariance matrix:

covarianceMatrix = np.cov(featuresStandardized.T)

# Can also be written as covarianceMatrix = np.cov(featuresStandard
# This point just aims to explain that every column is a feature, n

print(covarianceMatrix)
```

```
   2.27079511e-01  3.95694673e-01 -1.03935708e-01 -1.00215889e-03
   9.95457402e-01  3.65741024e-01  1.00176056e+00  9.79299180e-01
   2.37191461e-01  5.30339746e-01  6.19432713e-01  8.17759288e-01
   2.69967228e-01  1.39201504e-01]
 [ 9.42739295e-01  3.44150782e-01  9.43207466e-01  9.60902082e-01
   2.07082304e-01  5.10500995e-01  6.77177350e-01  8.11055024e-01
   1.77505338e-01 -2.32262646e-01  7.52871625e-01 -8.33414586e-02
   7.31999440e-01  8.12836496e-01 -1.82516245e-01  1.99722335e-01
   1.88684259e-01  3.42873752e-01 -1.10537008e-01 -2.27761757e-02
   9.85746984e-01  3.46451160e-01  9.79299180e-01  1.00176056e+00
   2.09513547e-01  4.39067932e-01  5.44287093e-01  7.48734680e-01
   2.09513722e-01  7.97872577e-02]
 [ 1.19826732e-01  7.76398084e-02  1.50814456e-01  1.23740409e-01
   8.06742020e-01  5.66536837e-01  4.49612218e-01  4.53550155e-01
   4.27426215e-01  5.05831058e-01  1.42168410e-01 -7.37873381e-02
   1.30283361e-01  1.25610187e-01  3.15011078e-01  2.27794574e-01
   1.68777943e-01  2.15729735e-01 -1.26840915e-02  1.70868612e-01
   2.16955724e-01  2.25826298e-01  2.37191461e-01  2.09513547e-01
   1.00176056e+00  5.69186845e-01  5.19436186e-01  5.48655147e-01
   4.94707764e-01  6.18711558e-01]
 [ 4.14190751e-01  2.78318729e-01  4.56576647e-01  3.91097651e-01
   4.73300254e-01  8.67333351e-01  7.56297185e-01  6.68628771e-01
   4.74033112e-01  4.59605900e-01  2.87608629e-01 -9.26020990e-02
   3.42521416e-01  2.83755229e-01 -5.56559523e-02  6.79975390e-01
   4.85711424e-01  4.53685716e-01  6.03609620e-02  3.90845741e-01
   4.76657749e-01  3.61467607e-01  5.30339746e-01  4.39067932e-01
   5.69186845e-01  1.00176056e+00  8.93831781e-01  8.02490717e-01
   6.15522263e-01  8.11881713e-01]
 [ 5.27839123e-01  3.01555198e-01  5.64872009e-01  5.13508396e-01
   4.35691429e-01  8.17712354e-01  8.85659158e-01  7.53724145e-01
   4.34484601e-01  3.46843443e-01  3.81254678e-01 -6.90776223e-02
   4.19636314e-01  3.85778129e-01 -5.84010247e-02  6.40271956e-01
   6.63730620e-01  5.50559967e-01  3.71843990e-02  3.80643631e-01
   5.74985227e-01  3.69014138e-01  6.19432713e-01  5.44287093e-01
   5.19436186e-01  8.93831781e-01  1.00176056e+00  8.56939906e-01
   5.33457264e-01  6.87719567e-01]
 [ 7.45524434e-01  2.95835766e-01  7.72598608e-01  7.23287782e-01
   5.03939011e-01  8.17009092e-01  8.62839447e-01  9.11757700e-01
   4.31054176e-01  1.75634121e-01  5.31997297e-01 -1.19848153e-01
   5.55874162e-01  5.39113790e-01 -1.02186386e-01  4.84059046e-01
   4.41247742e-01  6.03510257e-01 -3.04669411e-02  2.15582894e-01
   7.88810161e-01  3.60387980e-01  8.17759288e-01  7.48734680e-01
   5.48655147e-01  8.02490717e-01  8.56939906e-01  1.00176056e+00
   5.03413227e-01  5.12013995e-01]
 [ 1.64241985e-01  1.05192783e-01  1.89447989e-01  1.43822678e-01
   3.95003689e-01  5.11121711e-01  4.10185014e-01  3.76405667e-01
   7.01057885e-01  3.34606745e-01  9.47092790e-02 -1.28440488e-01
   1.10123974e-01  7.42567956e-02 -1.07531080e-01  2.78367653e-01
   1.98136040e-01  1.43367633e-01  3.90088053e-01  1.11289544e-01
   2.43957953e-01  2.33437721e-01  2.69967228e-01  2.09513722e-01
   4.94707764e-01  6.15522263e-01  5.33457264e-01  5.03413227e-01
   1.00176056e+00  5.38795122e-01]
 [ 7.07832563e-03  1.19415220e-01  5.11083511e-02  3.74417763e-03
   5.00195447e-01  6.88592503e-01  5.15836457e-01  3.69310185e-01
   4.39185353e-01  7.68647654e-01  4.96466850e-02 -4.57349464e-02
   8.55829815e-02  1.75701742e-02  1.01658978e-01  5.92013208e-01
   4.40102736e-01  3.11201479e-01  7.82169401e-02  5.92369136e-01
   9.36565772e-02  2.19508204e-01  1.39201504e-01  7.97872577e-02
   6.18711558e-01  8.11881713e-01  6.87719567e-01  5.12013995e-01
   5.38795122e-01  1.00176056e+00]]
```

Eigenvalue Decomposition:

Theoretically, after implementing PCA, the most axis conatining the most variance of the data is the set of eigenvectors.

Given that the covariance matrix is a square symmetric matrix, and according to the spectral theorem, there exists an orthogonal set of eigenvectors as a basis.

This means that for a covariance matrix of dimension 30, the data can be represented using a set of 30 dimensions, each orthogonal to one another (right angles from one another). The eigenvalues on the other hand portray the variance inherent in each direction.

If the eigenvectors are sorted with the highest eigenvalues, the n new dimensions can be represented with the highest n eigenvectors.

```
# Compute eigenvalues and eigenvectors:

eigenvalues, eigenvectors = np.linalg.eig(covarianceMatrix)

# Sort the eigenvectors by the highest eigenvalues
# np.argsort(eigenvalues)[::-1] returns the indicies of the highest
eigenvectors = eigenvectors[np.argsort(eigenvalues)[::-1]]
```

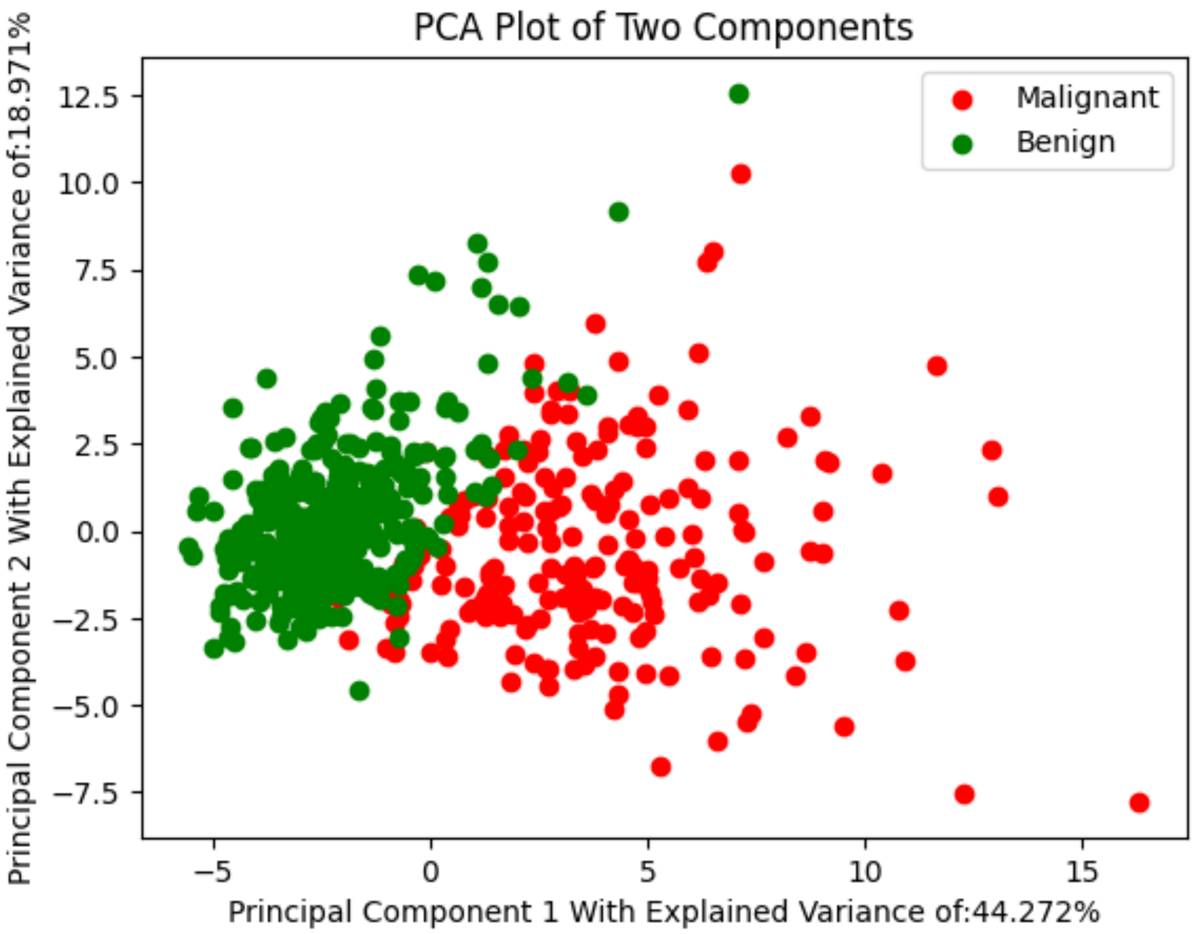
Explained Variance Ratio:

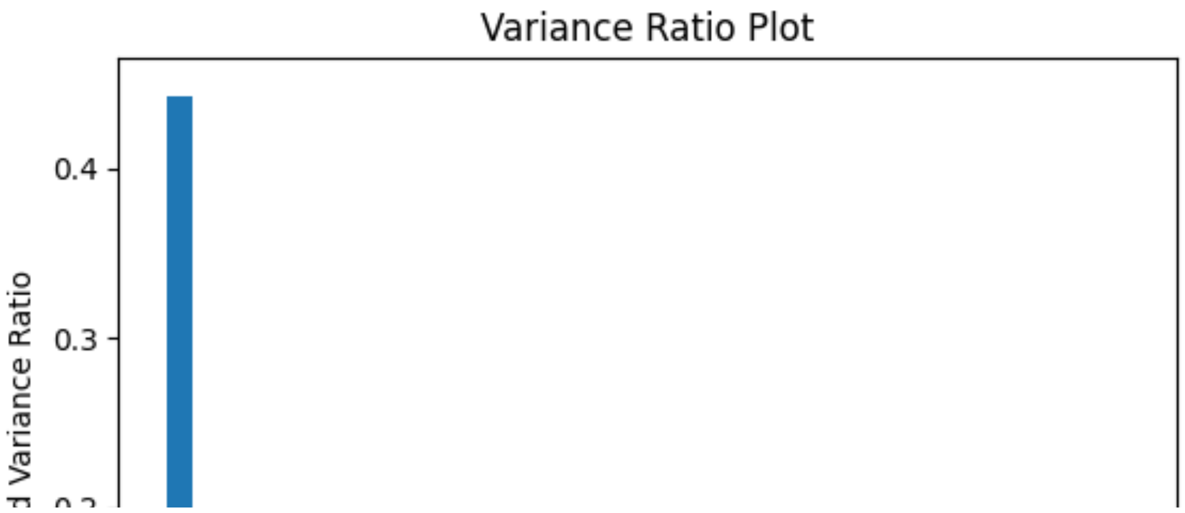
"Variance Ratio refers to the ratio of variance explained by each principal component in a Principal Component Analysis (PCA). It is used to determine the optimal number of dimensions needed to explain the variance in a dataset, with the total ratio summing up to 1." https://www.sciencedirect.com/topics/computer-science/variance-ratio

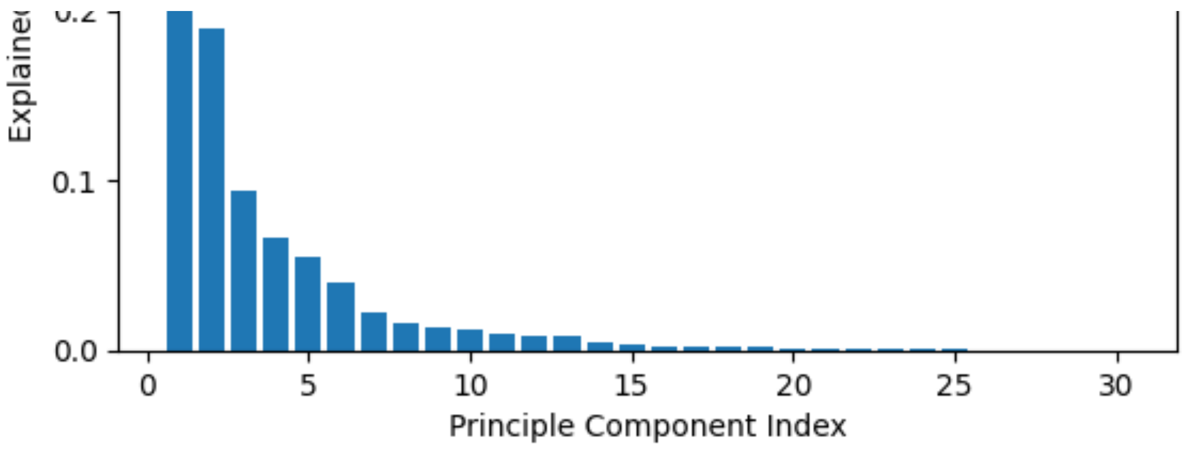
```
# Simplified previous operations to a class.

class PCA:
  def __init__(self, numberOfPrincipleComponents=2):
    self.numberOfPrincipleComponents = numberOfPrincipleComponents
    self.eigenvectors = None
    self.eigenvalues = None
    self.mean = None
    self.std = None
    self.covarianceMatrix = None
    self.explainedVarianceRatio = None
    self.pca = None
    self.reconstructed = None

  def fit(self, features):
    self.mean = np.mean(features, axis=0)
    self.std = np.std(features, axis=0)

    # Precautionary step to ensure no featurs having a standard devi
    self.std[self.std == 0] = 1

    standardized = (features - self.mean) / self.std
    self.covarianceMatrix = np.cov(standardized.T)
    eigenvalues, eigenvectors = np.linalg.eig(self.covarianceMatrix)
    # By default, np.argsort sorts in ascending order. For this mode
    sortedIndicies = np.argsort(eigenvalues)[::-1]
    self.eigenvalues = eigenvalues[sortedIndicies]
    # Sorts columns, not rows.
    self.eigenvectors = eigenvectors[:,sortedIndicies]
    totalVariance = np.sum(eigenvalues)
    self.explainedVarianceRatio = self.eigenvalues / totalVariance

    return self

  def project(self,features):
    standardized = (features - self.mean) / self.std
    # The np.dot function projects the input data onto the provided 
    pca = np.dot(standardized, self.eigenvectors[:, :self.numberOfPr
    self.pca = pca
    return pca

  def reconstruct(self, projectedFeatures):
    # Reconstruct the data after projecting it on the prinicple axes
    reconstructed = np.dot(projectedFeatures, self.eigenvectors[:, :
    self.reconstructed = reconstructed
    return reconstructed

  def reconstructionError(self, features):
    # Calculating the error in the reconstructed data
    projectedData = self.project(features)
    reconstructedData = self.reconstruct(projectedData)

    error = np.mean((features - reconstructedData)**2)

    print(f"Reconstruction Error using MSE:{error:.3f}")
    return error


  def plot(self,target):
    # Plotting a scatter plot
    plt.figure()
    color = ['r','g']
    names = ['Malignant', 'Benign']
    for i in range(len(color)):
      plt.scatter(self.pca[target==i,0], self.pca[target==i,1], c=co
    plt.title("PCA Plot of Two Components")
    plt.xlabel(f"Principal Component 1 With Explained Variance of:{s
    plt.ylabel(f"Principal Component 2 With Explained Variance of:{s
    plt.legend()

    plt.figure()
    # Plot a bar chart for the explained variance of all features in
    barComponents=np.arange(1, len(self.explainedVarianceRatio)+1)
    plt.bar(barComponents, self.explainedVarianceRatio,align='center

    plt.ylabel('Explained Variance Ratio')
    plt.xlabel('Principle Component Index')
    plt.xlabel('Principle Component Index')
    plt.title('Variance Ratio Plot')
```

```
# Testing PCA class:

pca = PCA(numberOfPrincipleComponents=2)
pca.fit(features)
pca.project(features)
pca.plot(target)
reconstructionError = pca.reconstructionError(features)
```

```
Reconstruction Error using MSE:809.793
```







Explained
0.2
0.1
0.0
0
5
10
15
20
25
30
Principle Component Index