Machine Learning (CS 3035)

Principal Component Analysis (PCA) Assignment

Principal Component Analysis (PCA)

Dataset: You can choose any dataset of your interest from publicly available datasets or use your own dataset.

Tasks:

- Load the dataset and perform necessary preprocessing steps, such as handling missing values, scaling, etc.

- Implement PCA from scratch using Python, NumPy, and Matplotlib, and apply it to the dataset.

- Use the scikit-learn library to apply PCA to the dataset and compare the results with the implementation from scratch.

- Visualize the results of PCA using Matplotlib or any other visualization library of your choice.

- Evaluate the performance of PCA by calculating the explained variance ratio for each principal component and selecting the appropriate number of principal components for the dataset.

- Use the selected number of principal components to reconstruct the original dataset and calculate the reconstruction error.

- Compare the results of PCA with and without dimensionality reduction using a classification algorithm of your choice, such as logistic regression, k-nearest neighbors, or support vector machines.

Write a brief report summarizing your findings and observations.

Note: You are encouraged to use additional libraries or tools if you think they could be helpful for your analysis.

In [8]:
```
# SYED FAISAL | ML_CS-9 | 21052295

import pandas as pd
import numpy as np

# Dataset of scikit learn
from sklearn.datasets import load_breast_cancer
```

In [9]:
```
# Load the Breast Cancer dataset
cancer = load_breast_cancer(as_frame=True)

# Creating dataframe
df = cancer.frame

# Checking the shape of the original dataframe
print(f"Original Dataframe Shape: {df.shape}")

# Extracting input features
```

```
X = df[cancer['feature_names']]
print(f"Input Dataframe shape: {X.shape}")
```

```
Original Dataframe Shape: (569, 31)
Input Dataframe shape: (569, 30)
```

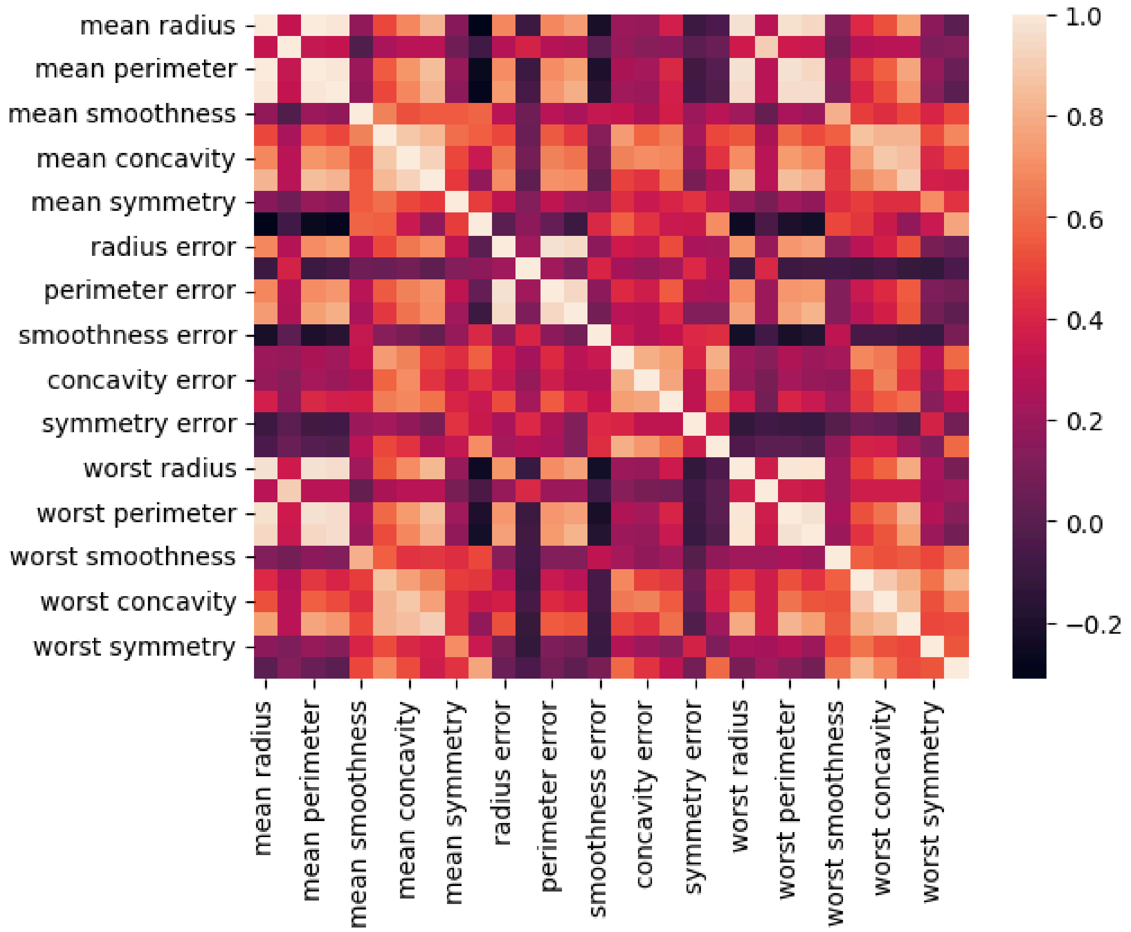
In [12]:
```
# Mean
X_mean = X.mean()

# Standard Deviation
X_std = X.std()

# Standardization
Z = (X - X_mean) / X_std
```

In [14]:
```
# Compute the Covariance matrix
c = Z.cov()

# Plot the covariance matrix
import matplotlib.pyplot as plt
import seaborn as sns
sns.heatmap(c)
plt.show()
```



In [15]:
```
# Compute Eigenvalues and Eigenvectors

eigenvalues, eigenvectors = np.linalg.eig(c)
print('Eigen Values:\n', eigenvalues)
print('Eigen Values Shape:', eigenvalues.shape)
print('Eigen Vector Shape:', eigenvectors.shape)
```

```
Eigen Values:
 [1.32816077e+01 5.69135461e+00 2.81794898e+00 1.98064047e+00
 1.64873055e+00 1.20735661e+00 6.75220114e-01 4.76617140e-01
 4.16894812e-01 3.50693457e-01 2.93915696e-01 2.61161370e-01
 2.41357496e-01 1.57009724e-01 9.41349650e-02 7.98628010e-02
 5.93990378e-02 5.26187835e-02 4.94775918e-02 1.33044823e-04
 7.48803097e-04 1.58933787e-03 6.90046388e-03 8.17763986e-03
 1.54812714e-02 1.80550070e-02 2.43408378e-02 2.74394025e-02
 3.11594025e-02 2.99728939e-02]
Eigen Values Shape: (30,)
Eigen Vector Shape: (30, 30)
```

In [17]:

```
# Sorting eigenvalues and corresponding eigenvectors

# Index the eigenvalues in descending order
idx = eigenvalues.argsort()[::-1]

# Sort the eigenvalues in descending order
eigenvalues = eigenvalues[idx]

# sort the corresponding eigenvectors accordingly
eigenvectors = eigenvectors[:,idx]
```

In [18]:

```
explained_var = np.cumsum(eigenvalues) / np.sum(eigenvalues)
explained_var
```

Out[18]:

```
array([0.44272026, 0.63243208, 0.72636371, 0.79238506, 0.84734274,
       0.88758796, 0.9100953 , 0.92598254, 0.93987903, 0.95156881,
       0.961366  , 0.97007138, 0.97811663, 0.98335029, 0.98648812,
       0.98915022, 0.99113018, 0.99288414, 0.9945334 , 0.99557204,
       0.99657114, 0.99748579, 0.99829715, 0.99889898, 0.99941502,
       0.99968761, 0.99991763, 0.99997061, 0.99999557, 1.        ])
```

In [21]:

```
n_components = np.argmax(explained_var >= 0.50) + 1
n_components
```

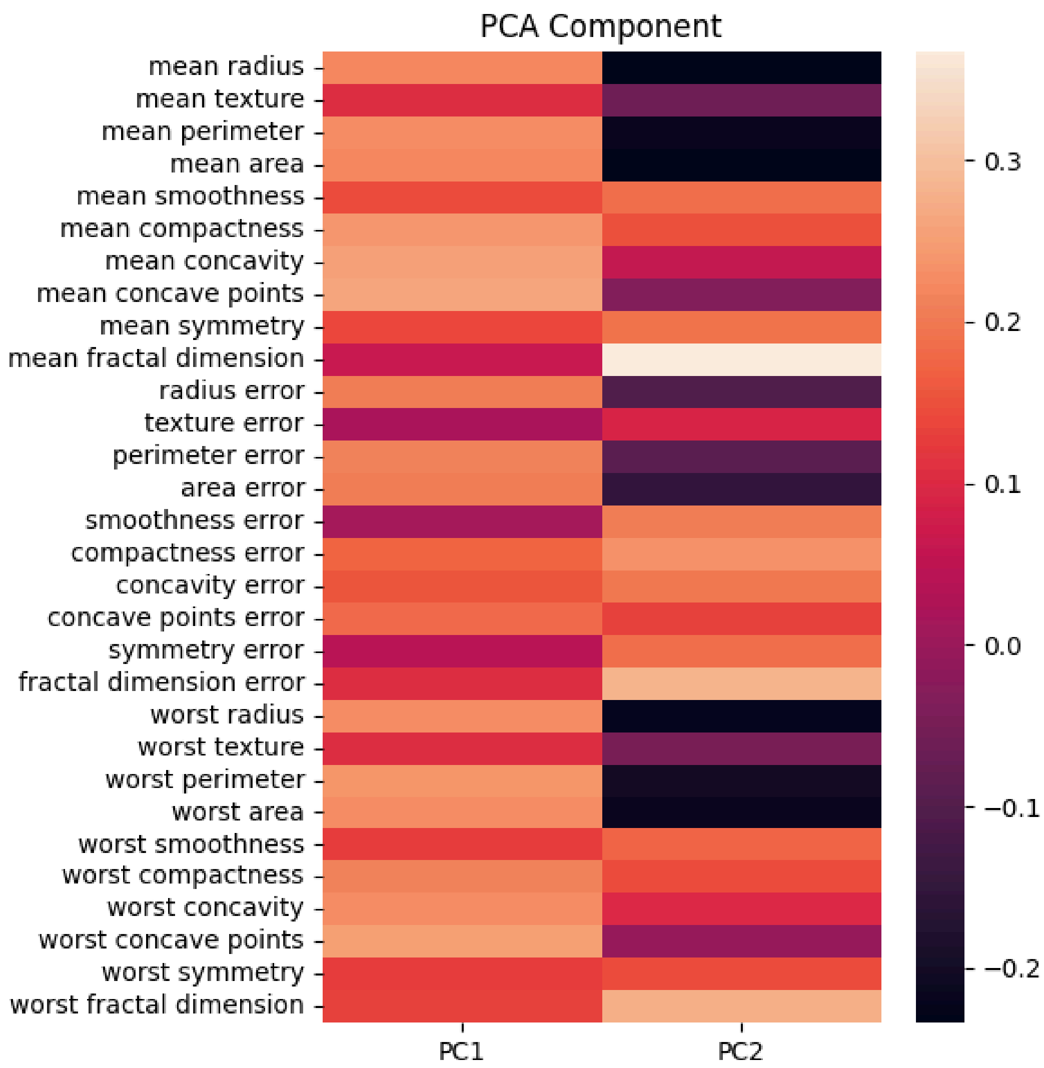
Out[21]:

```
2
```

In [23]:

```
# PCA component or unit matrix
u = eigenvectors[:,:n_components]

# Create dataframe for PCA components
pca_component = pd.DataFrame(u,
                             index = cancer['feature_names'],
                             columns = ['PC1','PC2']
                            )
# plotting heatmap
plt.figure(figsize =(5, 7))
sns.heatmap(pca_component)
plt.title('PCA Component')
plt.show()
```



In [24]:
```
# Matrix multiplication or dot Product
Z_pca = Z @ pca_component

# Rename the columns name
Z_pca.rename({'PC1': 'PCA1', 'PC2': 'PCA2'}, axis=1, inplace=True)

# Print the  Pricipal Component values
print(Z_pca)
```

```
          PCA1       PCA2
0     9.184755   1.946870
1     2.385703  -3.764859
2     5.728855  -1.074229
3     7.116691  10.266556
4     3.931842  -1.946359
..         ...        ...
564   6.433655  -3.573673
565   3.790048  -3.580897
566   1.255075  -1.900624
567  10.365673   1.670540
568  -5.470430  -0.670047

[569 rows x 2 columns]
```

In [25]:
```
# Importing PCA
from sklearn.decomposition import PCA

# Let's say, components = 2 applying PCA using scikit-learn
pca = PCA(n_components=2)
pca.fit(Z)
```

```
x_pca = pca.transform(Z)

# Create the dataframe
df_pca1 = pd.DataFrame(x_pca,
                       columns=['PC{}'.
                       format(i+1)
                        for i in range(n_components)])
print(df_pca1)
```

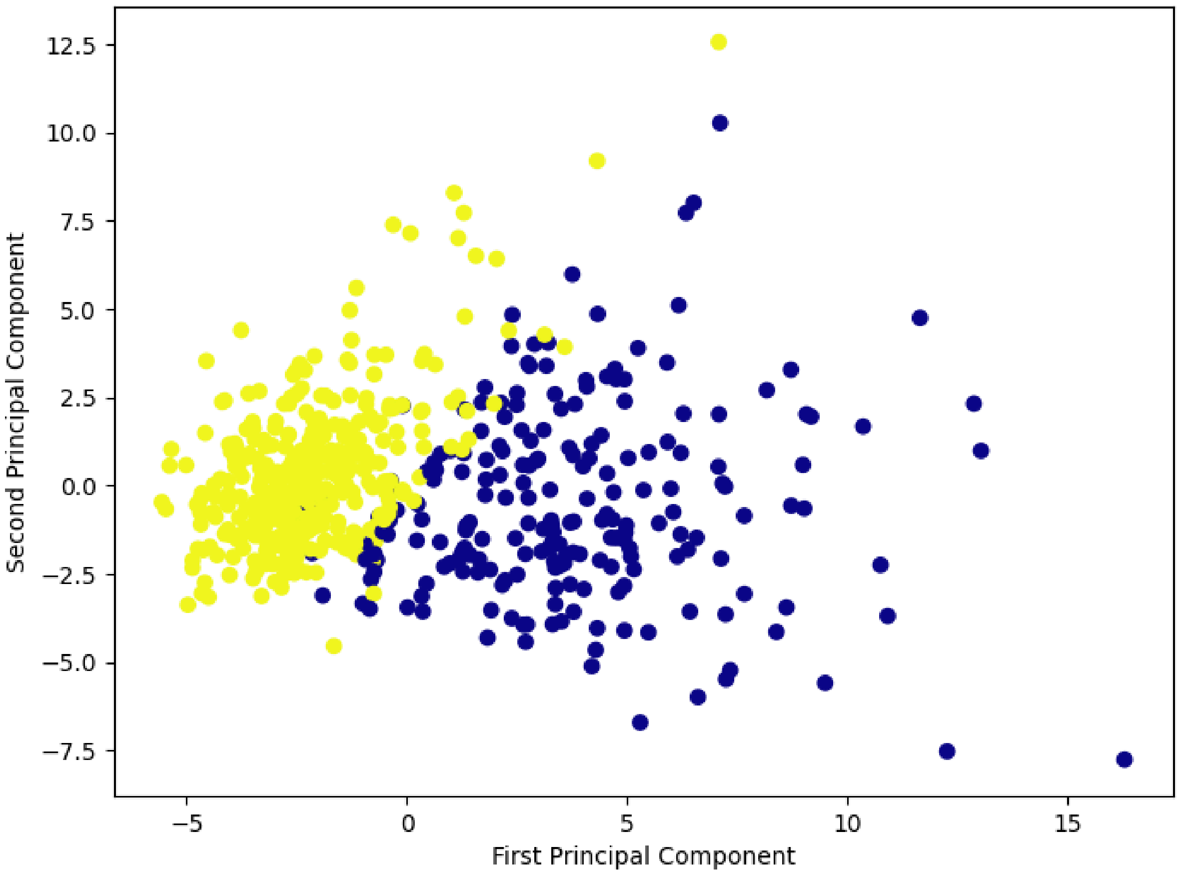
```
           PC1        PC2
0     9.184755   1.946870
1     2.385703  -3.764859
2     5.728855  -1.074229
3     7.116691  10.266556
4     3.931842  -1.946359
..         ...        ...
564   6.433655  -3.573673
565   3.790048  -3.580897
566   1.255075  -1.900624
567  10.365673   1.670540
568  -5.470430  -0.670047

[569 rows x 2 columns]
```

In [27]:
```
# Viusalizing
plt.figure(figsize=(8, 6))

plt.scatter(x_pca[:, 0], x_pca[:, 1],
            c=cancer['target'],
            cmap='plasma')

# Labeling x and y axes
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()
```



Reconstruction from Principal Component

In [28]:
```
reduced_eigen_space = eigenvectors[:, :350]
```

In [31]:
```
print(f'Shape of X_Scaled: {X.shape}')
print(f'Shape of Reduced_Eigen_Space: {reduced_eigen_space.shape}')
X_compressed = np.dot(X, reduced_eigen_space)
print(f'Shape of X_Compressed: {X_compressed.shape}')
```

```
Shape of X_Scaled: (569, 30)
Shape of Reduced_Eigen_Space: (30, 30)
Shape of X_Compressed: (569, 30)
```

In [32]:
```
print(f'Shape of X_Compressed: {X_compressed.shape}')
print(f'Shape of Reduced_Eigen_Space: {reduced_eigen_space.shape}')
X_reconstructed = np.dot(X_compressed,reduced_eigen_space.T)
print(f'Shape of X_Reconstructed: {X_reconstructed.shape}')
```

```
Shape of X_Compressed: (569, 30)
Shape of Reduced_Eigen_Space: (30, 30)
Shape of X_Reconstructed: (569, 30)
```

In [34]:
```
# Calculating MSE
mse = np.mean(np.square(X - X_reconstructed))

print(f'Reconstruction Error (MSE): {mse}')
```

```
Reconstruction Error (MSE): 1.1414438948328e-22
```

Comparison: Compare the results of PCA with and without dimensionality reduction using a classification algorithm of logistic regression

Importing the rest of the dependencies.

In [36]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

Data Collection & Processing

In [37]:
```
# Loading data from scikit Learn
breast_cancer_dataset = load_breast_cancer()
```

In [39]:
```
print(breast_cancer_dataset)
```

```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]), 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]), 'frame': None, 'target_nam
es': array(['malignant', 'benign'], dtype='<U9'), 'DESCR': '.. _breast_cancer_dataset:\n\nBr
east cancer wisconsin (diagnostic) dataset\n--------------------------------------------\n\n
**Data Set Characteristics:**\n\n:Number of Instances: 569\n\n:Number of Attributes: 30 nume
ric, predictive attributes and the class\n\n:Attribute Information:\n    - radius (mean of d
istances from center to points on the perimeter)\n    - texture (standard deviation of gray-
scale values)\n    - perimeter\n    - area\n    - smoothness (local variation in radius leng
ths)\n    - compactness (perimeter^2 / area - 1.0)\n    - concavity (severity of concave por
tions of the contour)\n    - concave points (number of concave portions of the contour)\n    
- symmetry\n    - fractal dimension ("coastline approximation" - 1)\n\n    The mean, standar
d error, and "worst" or largest (mean of the three\n    worst/largest values) of these featu
res were computed for each image,\n    resulting in 30 features.  For instance, field 0 is M
ean Radius, field\n    10 is Radius SE, field 20 is Worst Radius.\n\n    - class:\n        
- WDBC-Malignant\n            - WDBC-Benign\n\n:Summary Statistics:\n\n=====================
================ ====== ======\n                                        Min    Max\n========
============================ ====== ======\nradius (mean):                      6.981  2
8.11\ntexture (mean):                     9.71   39.28\nperimeter (mean):                  
43.79  188.5\narea (mean):                        143.5  2501.0\nsmoothness (mean):         
0.053  0.163\ncompactness (mean):                 0.019  0.345\nconcavity (mean):           
0.0    0.427\nconcave points (mean):              0.0    0.201\nsymmetry (mean):            
0.106  0.304\nfractal dimension (mean):           0.05   0.097\nradius (standard error):    
0.112  2.873\ntexture (standard error):           0.36   4.885\nperimeter (standard erro
r):           0.757  21.98\narea (standard error):              6.802  542.2\nsmoothness 
(standard error):          0.002  0.031\ncompactness (standard error):         0.002  0.135
\nconcavity (standard error):           0.0    0.396\nconcave points (standard error):     
0.0    0.053\nsymmetry (standard error):           0.008  0.079\nfractal dimension (standar
d error):   0.001  0.03\nradius (worst):                     7.93   36.04\ntexture (wors
t):                     12.02  49.54\nperimeter (worst):                  50.41  251.2\na
rea (worst):                        185.2  4254.0\nsmoothness (worst):                 0.
```

```
071  0.223\ncompactness (worst):                0.027  1.058\nconcavity (worst):
0.0    1.252\nconcave points (worst):             0.0    0.291\nsymmetry (worst):
0.156  0.664\nfractal dimension (worst):          0.055  0.208\n==========================
=========== ====== ======\n\n:Missing Attribute Values: None\n\n:Class Distribution: 212 - M
alignant, 357 - Benign\n\n:Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasa
rian\n\n:Donor: Nick Street\n\n:Date: November, 1995\n\nThis is a copy of UCI ML Breast Canc
er Wisconsin (Diagnostic) datasets.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a d
igitized image of a fine needle\naspirate (FNA) of a breast mass.  They describe\ncharacteri
stics of the cell nuclei present in the image.\n\nSeparating plane described above was obtai
ned using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree\nConstruction Via
Linear Programming." Proceedings of the 4th\nMidwest Artificial Intelligence and Cognitive S
cience Society,\npp. 97-101, 1992], a classification method which uses linear\nprogramming t
o construct a decision tree.  Relevant features\nwere selected using an exhaustive search in
the space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual linear program used to o
btain the separating plane\nin the 3-dimensional space is that described in:\n[K. P. Bennett
and O. L. Mangasarian: "Robust Linear\nProgramming Discrimination of Two Linearly Inseparabl
e Sets",\nOptimization Methods and Software 1, 1992, 23-34].\n\nThis database is also availa
ble through the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-l
earn/WDBC/\n\n|details-start|\n**References**\n|details-split|\n\n- W.N. Street, W.H. Wolber
g and O.L. Mangasarian. Nuclear feature extraction\n  for breast tumor diagnosis. IS&T/SPIE
1993 International Symposium on\n  Electronic Imaging: Science and Technology, volume 1905,
pages 861-870,\n  San Jose, CA, 1993.\n- O.L. Mangasarian, W.N. Street and W.H. Wolberg. Bre
ast cancer diagnosis and\n  prognosis via linear programming. Operations Research, 43(4), pa
ges 570-577,\n  July-August 1995.\n- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machin
e learning techniques\n  to diagnose breast cancer from fine-needle aspirates. Cancer Letter
s 77 (1994)\n  163-171.\n\n|details-end|\n', 'feature_names': array(['mean radius', 'mean te
xture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23'), 'filename': 'breast_canc
er.csv', 'data_module': 'sklearn.datasets.data'}
```

In [41]:
```
# Loading the data to a data frame
data_frame = pd.DataFrame(breast_cancer_dataset.data, columns = breast_cancer_dataset.featu
```

In [42]:
```
# Print the first 5 rows of the dataframe
data_frame.head()
```

Out[42]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	di
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	

5 rows × 30 columns

In [45]:
```
# Adding the 'target' column to the data frame
data_frame['label'] = breast_cancer_dataset.target
data_frame.tail()
```

Out[45]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587

5 rows × 31 columns

In [47]:
```
# number of rows and columns in the dataset
data_frame.shape
data_frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  label                    569 non-null    int32  
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

In [49]:
```
# Checking for missing values
data_frame.isnull().sum()
```

Out[49]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
label                      0
dtype: int64
```

In [50]:
```
# Statistical measures about the data
data_frame.describe()
```

Out[50]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	c
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0

8 rows × 31 columns

In [51]:
```
# Checking the distribution of Target Varibale
data_frame['label'].value_counts()
```

Out[51]:
```
label
1    357
0    212
Name: count, dtype: int64
```

In [53]:

```
# Printing the attributes with label 0 and 1
data_frame.groupby('label').mean()
```

Out[53]:

label	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points
0	17.462830	21.604906	115.365377	978.376415	0.102898	0.145188	0.160775	0.087990
1	12.146524	17.914762	78.075406	462.790196	0.092478	0.080085	0.046058	0.025717

2 rows × 30 columns

In [55]:

```
# Displaying dataframe
X = data_frame.drop(columns='label', axis=1)
Y = data_frame['label']
print(X)
```

```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0          17.99         10.38          122.80     1001.0          0.11840   
1          20.57         17.77          132.90     1326.0          0.08474   
2          19.69         21.25          130.00     1203.0          0.10960   
3          11.42         20.38           77.58      386.1          0.14250   
4          20.29         14.34          135.10     1297.0          0.10030   
..           ...           ...             ...        ...              ...   
564        21.56         22.39          142.00     1479.0          0.11100   
565        20.13         28.25          131.20     1261.0          0.09780   
566        16.60         28.08          108.30      858.1          0.08455   
567        20.60         29.33          140.10     1265.0          0.11780   
568         7.76         24.54           47.92      181.0          0.05263   

     mean compactness  mean concavity  mean concave points  mean symmetry  \
0             0.27760         0.30010              0.14710         0.2419   
1             0.07864         0.08690              0.07017         0.1812   
2             0.15990         0.19740              0.12790         0.2069   
3             0.28390         0.24140              0.10520         0.2597   
4             0.13280         0.19800              0.10430         0.1809   
..                ...             ...                  ...            ...   
564           0.11590         0.24390              0.13890         0.1726   
565           0.10340         0.14400              0.09791         0.1752   
566           0.10230         0.09251              0.05302         0.1590   
567           0.27700         0.35140              0.15200         0.2397   
568           0.04362         0.00000              0.00000         0.1587   

     mean fractal dimension  ...  worst radius  worst texture  \
0                   0.07871  ...        25.380          17.33   
1                   0.05667  ...        24.990          23.41   
2                   0.05999  ...        23.570          25.53   
3                   0.09744  ...        14.910          26.50   
4                   0.05883  ...        22.540          16.67   
..                      ...  ...           ...            ...   
564                 0.05623  ...        25.450          26.40   
565                 0.05533  ...        23.690          38.25   
566                 0.05648  ...        18.980          34.12   
567                 0.07016  ...        25.740          39.42   
568                 0.05884  ...         9.456          30.37   

     worst perimeter  worst area  worst smoothness  worst compactness  \
0             184.60      2019.0           0.16220            0.66560   
1             158.80      1956.0           0.12380            0.18660   
2             152.50      1709.0           0.14440            0.42450   
3              98.87       567.7           0.20980            0.86630   
4             152.20      1575.0           0.13740            0.20500   
..               ...         ...               ...                ...   
564           166.10      2027.0           0.14100            0.21130   
565           155.00      1731.0           0.11660            0.19220   
566           126.70      1124.0           0.11390            0.30940   
567           184.60      1821.0           0.16500            0.86810   
568            59.16       268.6           0.08996            0.06444   

     worst concavity  worst concave points  worst symmetry  \
0             0.7119                0.2654          0.4601   
1             0.2416                0.1860          0.2750   
2             0.4504                0.2430          0.3613   
3             0.6869                0.2575          0.6638   
4             0.4000                0.1625          0.2364   
..               ...                   ...             ...   
564           0.4107                0.2216          0.2060   
565           0.3215                0.1628          0.2572   
566           0.3403                0.1418          0.2218   
567           0.9387                0.2650          0.4087   
568           0.0000                0.0000          0.2871   

     worst fractal dimension  
0                    0.11890  
```

```
1                      0.08902
2                      0.08758
3                      0.17300
4                      0.07678
..                         ...
564                    0.07115
565                    0.06637
566                    0.07820
567                    0.12400
568                    0.07039

[569 rows x 30 columns]
```

In [56]:
```python
print(Y)
```

```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: label, Length: 569, dtype: int32
```

In [60]:
```python
# Split the dataset into train and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
print(X.shape, X_train.shape, X_test.shape)
```

```
(569, 30) (455, 30) (114, 30)
```

PCA without Dimensionality Reduction

In [61]:
```python
pca_full = PCA()
X_train_pca_full = pca_full.fit_transform(X_train)
X_test_pca_full = pca_full.transform(X_test)
```

PCA with Dimensionality Reduction

In [63]:
```python
num_components = 2                            # Specify the number of principal components
pca_reduced = PCA(n_components=num_components)
X_train_pca_reduced = pca_reduced.fit_transform(X_train)
X_test_pca_reduced = pca_reduced.transform(X_test)
```

Classification Algorithm (Logistic Regression)

Without dimensionalty reduction

In [65]:
```python
model_full = LogisticRegression()
model_full.fit(X_train_pca_full, Y_train)
y_pred_full = model_full.predict(X_test_pca_full)
accuracy_full = accuracy_score(Y_test, y_pred_full)
```

```
C:\Users\KIIT\AppData\Roaming\Python\Python312\site-packages\sklearn\linear_model\_logistic.
py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

With dimensionality reduction

In [67]:
```
model_reduced = LogisticRegression()
model_reduced.fit(X_train_pca_reduced, Y_train)
y_pred_reduced = model_reduced.predict(X_test_pca_reduced)
accuracy_reduced = accuracy_score(Y_test, y_pred_reduced)
```

Model Reduction

In [70]:
```
print(f"Accuracy without dimensionality reduction: {accuracy_full}")
print(f"Accuracy with dimensionality reduction: {accuracy_reduced}")
```

```
Accuracy without dimensionality reduction: 0.9385964912280702
Accuracy with dimensionality reduction: 0.9035087719298246
```

Summary -

- Principal Component Analysis (PCA) is a dimensionality reduction technique used to transform high-dimensional datasets into a lower-dimensional space while preserving the important information.
- Dataset Preprocessing: The Breast Cancer dataset from scikit-learn was used for this analysis. The dataset was loaded into a DataFrame, missing values were handled, and standardization was applied to scale the features.
- PCA was implemented from using NumPy, and scikit-learn was utilized for comparison, including visualization of the results through heatmaps and scatter plots in Matplotlib.
- The original dataset was reconstructed using selected principal components, and logistic regression was employed to compare classification accuracy between PCA with and without dimensionality reduction.
- PCA reduces dataset dimensionality while preserving most of the variance, validated by similar results between manual and scikit-learn implementations. Dimensionality reduction through PCA improves computational efficiency without significant loss in classification accuracy, as indicated by the reconstruction error and comparison with logistic regression.

Presented By -

SYED FAISAL
ML_CS-9
21052295