Experiment 10

Dimensionality Reduction: Principal Component Analysis (PCA)

Aim:

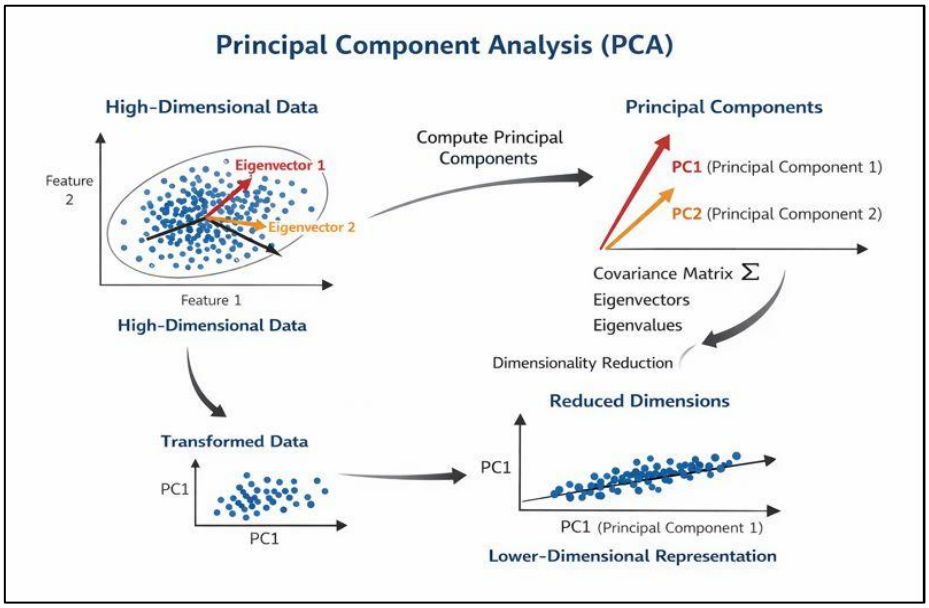
To apply Principal Component Analysis (PCA) for dimensionality reduction by transforming high-dimensional data into a lower-dimensional space while retaining the maximum variance.

Theory:

Principal Component Analysis (PCA) is a dimensionality reduction technique used in unsupervised learning to transform a dataset with many variables into a smaller set that still contains most of the essential information. PCA helps us get rid of non important features. It helps in faster training and inference even data visualization becomes easier. High-dimensional data often includes correlated features, which increase computational complexity and make data interpretation difficult. PCA addresses this problem by transforming the original variables into a new set of uncorrelated variables, called principal components, while retaining as much of the original information as possible.

The objective of PCA is to identify directions in the feature space along which the variance of the data is maximized. The diagram shows how a dataset with many correlated features is first represented in a high-dimensional space. PCA then identifies new axes called principal components that point in the directions of maximum data spread. The original data points are projected onto these new axes, resulting in a reduced-dimensional dataset that still preserves most of the important information.

The mathematical foundation of PCA lies in linear algebra, specifically in eigenvectors and eigenvalues. Eigenvectors define the directions of maximum variance in the data, while eigenvalues indicate the amount of variance explained by each direction.



For a square matrix$A$, an eigenvector $v$and its corresponding eigenvalue λsatisfy the equation:

$$Av = \lambda v$$

In PCA, eigenvectors and eigenvalues are derived from the covariance matrix of the data. Eigenvectors associated with larger eigenvalues correspond to more informative principal components. These eigenvectors are normalized to unit length before being used for projection.

By projecting the data onto a smaller number of important principal components, PCA reduces the number of features while keeping most of the useful information. Only the components that capture a large amount of variation in the data are selected. This way, PCA makes the dataset simpler and easier to work with without losing its essential patterns.

Merits of Using PCA:

- Makes large and complex data easier to handle by reducing the number of features
- Removes repeated or similar information from the dataset
- Helps models run faster and more efficiently

Demerits of Using PCA:

- New features created by PCA are hard to understand and explain
- Some useful information may be lost during reduction
- Does not work well when data has non-linear patterns

Code and Result:

Block 1: Importing Required Libraries

INPUT:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

import ipywidgets as widgets

from IPython.display import display

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

from ipywidgets import interact, IntSlider
```

print("Libraries Imported")

OUTPUT:

Libraries Imported

Block 2: Reading Data

INPUT:

df = pd.read_csv("/content/data.csv")

print("Data Read Successfully")

OUTPUT:

Data Read Successfully

Block 3: Displays the first five rows of the dataset

INPUT:

df.head()

OUTPUT:

Block 4: Displays the last five rows of the dataset

INPUT:

df.tail()

OUTPUT:

Block 5:Displays the size of dataset

INPUT:

df.shape

OUTPUT:

(569, 32)

Block 6:Displays summary of dataset, including column names, data types, and non-null counts.

INPUT:

df.info()

OUTPUT:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 569 entries, 0 to 568

Data columns (total 32 columns):

#	Column	Non-Null Count	Dtype
0	id	569 non-null	int64
1	radius_mean	569 non-null	float64
2	texture_mean	569 non-null	float64
3	perimeter_mean	569 non-null	float64
4	area_mean	569 non-null	float64
5	smoothness_mean	569 non-null	float64
6	compactness_mean	569 non-null	float64
7	concavity_mean	569 non-null	float64
8	concave points_mean	569 non-null	float64
9	symmetry_mean	569 non-null	float64
10	fractal_dimension_mean	569 non-null	float64
11	radius_se	569 non-null	float64
12	texture_se	569 non-null	float64
13	perimeter_se	569 non-null	float64
14	area_se	569 non-null	float64
15	smoothness_se	569 non-null	float64
16	compactness_se	569 non-null	float64
17	concavity_se	569 non-null	float64
18	concave points_se	569 non-null	float64

19 symmetry_se 569 non-null float64

20 fractal_dimension_se569 non-null float64

21 radius_worst569 non-null float64

22 texture_worst569 non-null float64

23 perimeter_worst569 non-null float64

24 area_worst 569 non-null float64

25 smoothness_worst 569 non-null float64

26 compactness_worst 569 non-null float64

27 concavity_worst 569 non-null float64

28 concave points_worst 569 non-null float64

29 symmetry_worst 569 non-null float64

30 fractal_dimension_worst569 non-null float64

31 diagnosis 569 non-null object

dtypes: float64(30), int64(1), object(1)

Block 7:Displays the statistics of dataset

INPUT:

df.describe()

OUTPUT:

Block 8:Counts the frequency of each unique category

INPUT:

df['diagnosis'].value_counts()

OUTPUT:

Diagnosis	Count
B	357
M	212

Block 9:Separate features and target variable

INPUT:

X = df.drop('diagnosis', axis=1)

y = df['diagnosis']

print("Feature shape:", X.shape)

print("Target shape:", y.shape)

OUTPUT:

Feature shape: (569, 31)

Target shape: (569,)

Block 10:Visualize correlations among features before applying PCA

INPUT:

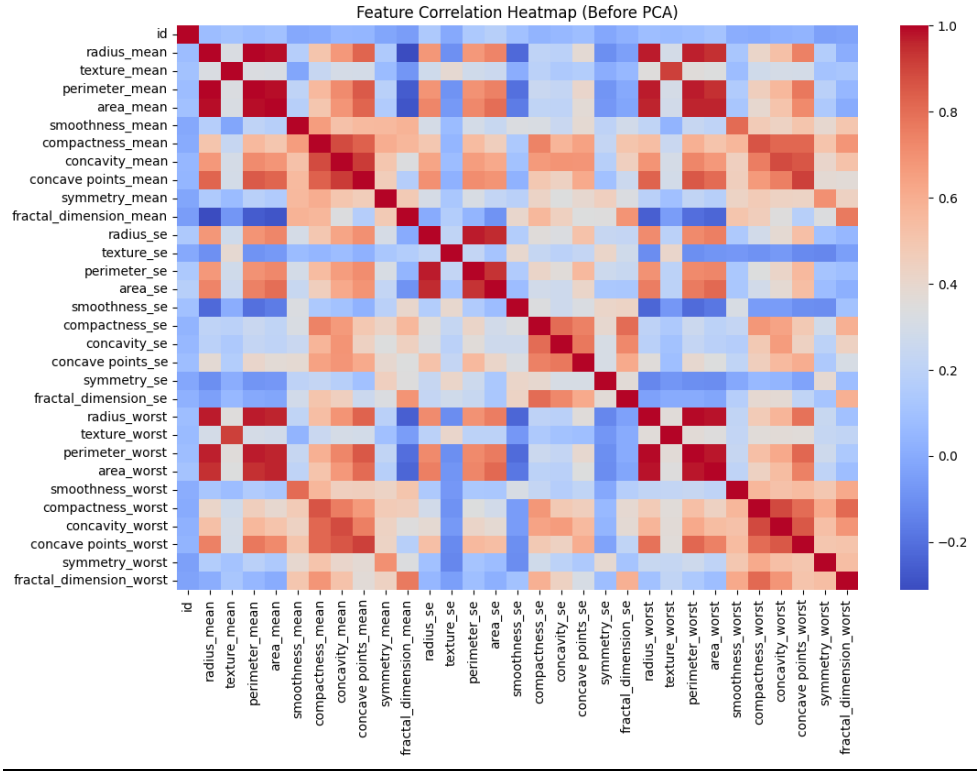
plt.figure(figsize=(12,8))

sns.heatmap(X.corr(), cmap='coolwarm')

plt.title("Feature Correlation Heatmap (Before PCA)")

plt.show()

OUTPUT:



Block 11:Standardizing features

INPUT:

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print("Features Standardized")
```

OUTPUT:

```
Features Standardized
```

Block 12:Data Splitting

INPUT:

```
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)
print("Data has been Split")
```

OUTPUT:

```
Data has been Split
```

Block 13: Apply PCA, and visualize variance

INPUT:

```
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
print("PCA applied with 95% variance retained.")
```

OUTPUT:

```
PCA applied with 95% variance retained.
```

Block 14: Model training using Logistic Regression

INPUT:

```
clf = LogisticRegression(max_iter=2000)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
acc_before = accuracy_score(y_test, y_pred)
clf_pca = LogisticRegression(max_iter=2000)
```

```
clf_pca.fit(X_train_pca, y_train)
y_pred_pca = clf_pca.predict(X_test_pca)
acc_after = accuracy_score(y_test, y_pred_pca)
print("Model Training completed.")
```

OUTPUT:

```
Model Training completed.
```

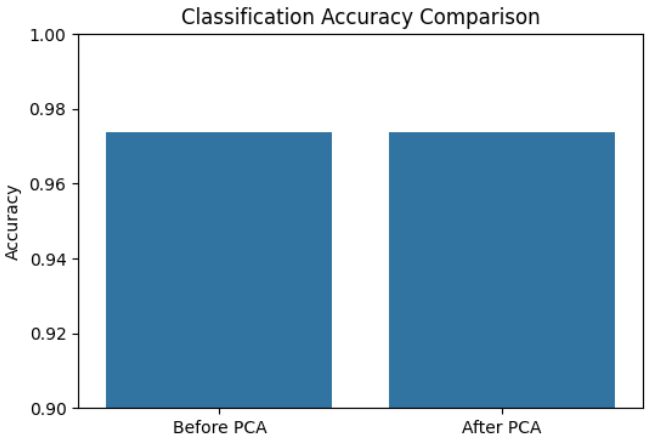
Block 15: Compare classification accuracy before and after applying PCA

INPUT:

```
print("Accuracy BEFORE PCA:", round(acc_before*100, 2), "%")
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)
print("Reduced Dimensions:", X_pca.shape[1])
X_train_pca, X_test_pca, y_train, y_test = train_test_split(
    X_pca, y, test_size=0.2, random_state=42, stratify=y)
print("Accuracy AFTER PCA:", round(acc_after*100, 2), "%")
plt.figure(figsize=(6,4))
sns.barplot(
    x=['Before PCA', 'After PCA'],
    y=[acc_before, acc_after])
plt.ylabel("Accuracy")
plt.title("Classification Accuracy Comparison")
plt.ylim(0.9, 1.0)
plt.show()
```

OUTPUT:

```
Accuracy BEFORE PCA: 97.37 %
Reduced Dimensions: 11
Accuracy AFTER PCA: 97.37 %
```



Block 16:Apply PCA and compute feature loadings for each principal component

INPUT:

```
pca = PCA(n_components=10)

X_pca = pca.fit_transform(X_scaled)

loadings = pd.DataFrame( pca.components_.T, columns=[f"PC{i+1}" for i in range(10)], index=X.columns)

print("PCA feature loadings calculated for the top 10 principal components.")
```

OUTPUT:

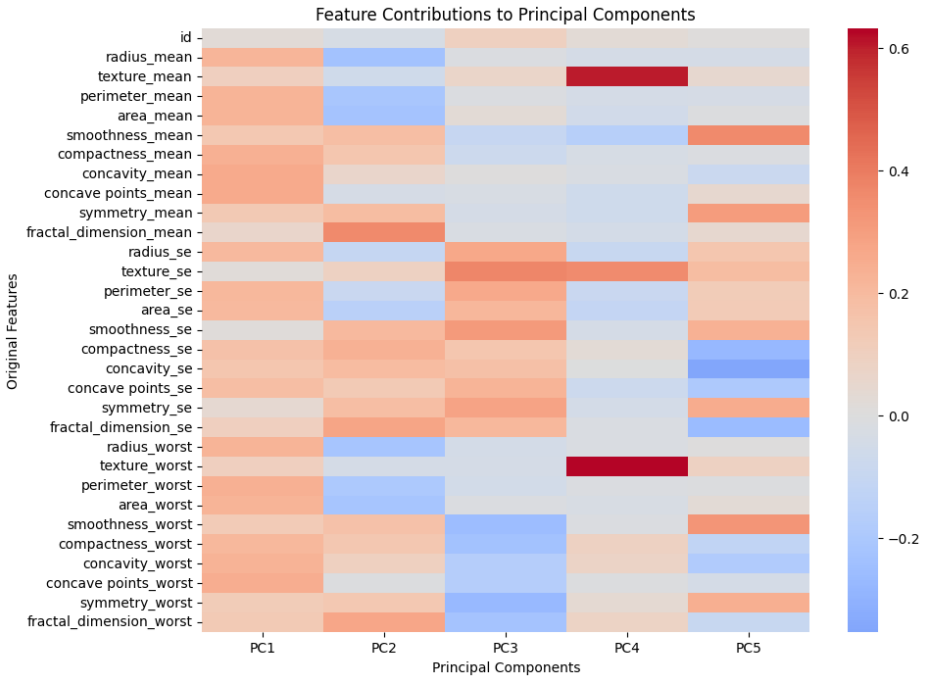
PCA feature loadings calculated for the top 10 principal components.

Block 17: Visualize feature contributions to the first few principal components using a heatmap

INPUT:

```
plt.figure(figsize=(10,8))

sns.heatmap(loadings.iloc[:, :5], cmap="coolwarm",center=0)

plt.title("Feature Contributions to Principal Components")

plt.xlabel("Principal Components")

plt.ylabel("Original Features")

plt.show()
```

OUTPUT:



Block 18: Interactively analyze and visualize top feature contributions for a selected principal component

INPUT:

```
def interactive_pca(pc_num=1):
    pc = f"PC{pc_num}"
    print(f"Principal Component: {pc}")
    print(f"Explained Variance Ratio: {pca.explained_variance_ratio_[pc_num-1]:.4f}")
    print(f"Cumulative Variance till {pc}: {pca.explained_variance_ratio_[:pc_num].sum():.4f}\n")
    top_features = loadings[pc].abs().sort_values(ascending=False).head(10)
    plt.figure(figsize=(8,4))
    sns.barplot( x=top_features.values, y=top_features.index)
    plt.xlim(0, top_features.max() * 1.1)
    plt.xlabel("Feature Contribution Strength")
    plt.ylabel("Features")
    plt.title(f"Top 10 Feature Contributions to {pc}")
    plt.grid(True)
    plt.show()
widgets.interact(interactive_pca,
pc_num=widgets.IntSlider(min=1,max=10,step=1,value=1,description="PCA Component"));
```

OUTPUT:

PCA Component= 1

Principal Component: PC1

Explained Variance Ratio: 0.4286

Cumulative Variance till PC1: 0.4286