Implement Dimensionality reduction using Principal Component Analysis (PCA) method.

PCA

1) Step 1 Normalise the data

mean = compute mean of the feature

normalized data = data - mean

2) compute covariance matrix

cov_mat = covariance (normalized data)

3) compute Eigen values & vectors

eigen val, eigen vectors = Eigen Decomposition (co...

4) Sort the eigen values & eigen vectors

sort_indices = sort eigen val in decend order.

principal comp = eigen vectors[sort_indices]

5) select top num eigen vectors

selected = principal comp[:num-comp]

6) reduced = dot (normalized, selected)

7) return reduced.

Code:

\+ Code + Text

RAM
Disk

[1]
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[2]
```python
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

[3]
```python
df = pd.read_csv('/content/drive/MyDrive/breast-cancer.csv')
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54

5 rows × 32 columns

[4]
```python
df.drop('id', axis=1, inplace=True) #drop redundant columns
```

[5]
```python
df['diagnosis'] = (df['diagnosis'] == 'M').astype(int) #encode the label into 1/0
```

[6]
```python
corr = df.corr()
```

[8]
```python
# Get the absolute value of the correlation
cor_target = abs(corr["diagnosis"])

# Select highly correlated features (thresold = 0.2)
relevant_features = cor_target[cor_target>0.2]

# Collect the names of the features
names = [index for index, value in relevant_features.items()]

# Drop the target variable from the results
names.remove('diagnosis')

# Display the results
print(names)
```

```
['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave p
```

[9]
```python
X = df[names].values
```

```
[9] X = df[names].values
```

```
[11] class PCA:
        """
        Principal Component Analysis (PCA) class for dimensionality reduction.
        """

        def __init__(self, n_components):
            """
            Constructor method that initializes the PCA object with the number of components to retain.

            Args:
            - n_components (int): Number of principal components to retain.
            """
            self.n_components = n_components
        def fit(self, X):
            """
            Fits the PCA model to the input data and computes the principal components.

            Args:
            - X (numpy.ndarray): Input data matrix with shape (n_samples, n_features).
            """
            # Compute the mean of the input data along each feature dimension.
            mean = np.mean(X, axis=0)

            # Subtract the mean from the input data to center it around zero.
            X = X - mean

            # Compute the covariance matrix of the centered input data.
            cov = np.cov(X.T)
```

```
[11]
            # Compute the covariance matrix of the centered input data.
            cov = np.cov(X.T)

            # Compute the eigenvectors and eigenvalues of the covariance matrix.
            eigenvalues, eigenvectors = np.linalg.eigh(cov)
            # Reverse the order of the eigenvalues and eigenvectors.
            eigenvalues = eigenvalues[::-1]
            eigenvectors = eigenvectors[:,::-1]

            # Keep only the first n_components eigenvectors as the principal components.
            self.components = eigenvectors[:,:self.n_components]

            # Compute the explained variance ratio for each principal component.
            # Compute the total variance of the input data
            total_variance = np.sum(np.var(X, axis=0))

            # Compute the variance explained by each principal component
            self.explained_variances = eigenvalues[:self.n_components]

            # Compute the explained variance ratio for each principal component
            self.explained_variance_ratio_ = self.explained_variances / total_variance
        def transform(self, X):
            """
            Transforms the input data by projecting it onto the principal components.

            Args:
            - X (numpy.ndarray): Input data matrix with shape (n_samples, n_features).

            Returns:
            - transformed_data (numpy.ndarray): Transformed data matrix with shape (n_samples, n_components).
            """
            # Center the input data around zero using the mean computed during the fit step.
            X = X - np.mean(X, axis=0)
```

Output:

```
fig = px.scatter(x=X_transformed[:,0], y=X_transformed[:,1])
fig.update_layout(
    title="PCA transformed data for breast cancer dataset",
    xaxis_title="PC1",
    yaxis_title="PC2"
)
fig.show()
```