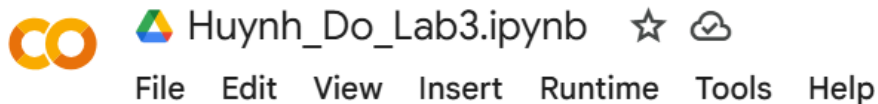


Huynh Do Lab#3:

Objective:

This lab assignment works with Principal Component Analysis (PCA) to expose how they can determine the best explained variance ratio. This is a statistical procedure that is used to reduce



1. Import libraries

```
[5] # Import required Python packages
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import sklearn
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA
import seaborn as sns # Optional for better visuals
from google.colab import files
from sklearn.preprocessing import StandardScaler
```

The code above imports several Python libraries commonly used in data analysis.

1. **matplotlib.pyplot** – Used to create basic graphs like scatter plots, line charts, and bar graphs.
2. **pandas** – Loads and manages tabular data (like CSVs) into an easy-to-use DataFrame.
3. **numpy** – Powers fast mathematical operations, especially with large arrays of numbers.
4. **sklearn** – Provides machine learning algorithms and data processing tools.
5. **scale** – Instantly standardizes your data to have a mean of 0 and variance of 1.
6. **PCA** – Reduces the number of variables while keeping the important patterns in your data.
7. **seaborn** – Makes fancier, cleaner-looking graphs with less code than plain matplotlib.
8. **google.colab.files** – Lets you upload and download files when working inside Google Colab.
9. **StandardScaler** – Another way to standardize data, but more control across train/test splits.

2. Upload file b5.csv

```
[6] # Upload 'Cereals.csv' file
    uploaded = files.upload()

    # Load the dataset
    df = pd.read_csv("b5.csv")
    df.head()
```

The above screen shot is used to upload and load a CSV file named **b5.csv** into a Pandas Data Frame in a Google Colab environment.

When upload is done:

 Choose Files | b5.csv

- **b5.csv**(text/csv) - 1912084 bytes, last modified: 4/21/2025 - 100% done

Saving b5.csv to b5 (2).csv

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	...	01	02	03	04	05	06	07	08	09	010
0	4	2	5	2	5	1	4	3	5	1	...	4	1	3	1	5	1	4	2	5	5
1	2	2	3	3	3	3	1	5	1	5	...	3	3	3	3	2	3	3	1	3	2
2	5	1	1	4	5	1	1	5	5	1	...	4	5	5	1	5	1	5	5	5	5
3	2	5	2	4	3	4	3	4	4	5	...	4	3	5	2	4	2	5	2	5	5
4	3	1	3	3	3	1	3	1	3	5	...	3	1	1	1	3	1	3	1	5	3

5 rows × 50 columns

3. Process PCA

Step 1

```
# Step 1: Standardize the dataset
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

- **scaler = StandardScaler()** Creates a scaler object to standardize features (zero mean, unit variance).
- **df_scaled = scaler.fit_transform(df)** Fits the scaler to the dataset and transforms the data into a standardized form.

Step 2

```
[18] # Step 2: Calculate covariance matrix
      cov_matrix = np.cov(df_scaled.T)
```

- **cov_matrix = np.cov(df_scaled.T)** Calculates the covariance matrix by transposing the standardized data so features are treated as variables.

Step 3

```
[19] # Step 3: Compute eigenvalues and eigenvectors
      eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)
```

- **eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)** Computes the eigenvalues and eigenvectors of the covariance matrix to find directions and magnitudes of variance.

Step 4

```
[21] # Step 4: Sort eigenvalues and eigenvectors in descending order
      sorted_indices = np.argsort(eigenvalues)[::-1]
      eigenvalues_sorted = eigenvalues[sorted_indices]
      eigenvectors_sorted = eigenvectors[:, sorted_indices]
```

- **sorted_indices = np.argsort(eigenvalues)[::-1]** Finds indices that sort eigenvalues from largest to smallest.
- **eigenvalues_sorted = eigenvalues[sorted_indices]** Reorders eigenvalues in descending order.

Step 5

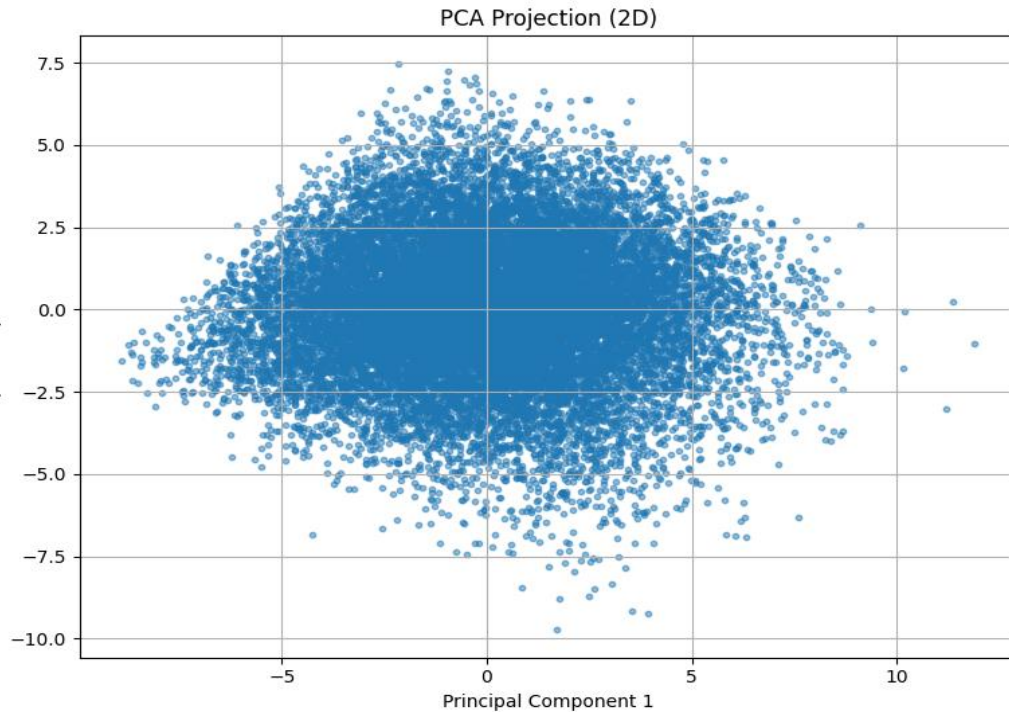
```
[22] # Step 5: Select top k eigenvectors and project the data (2D)
      k2 = 2
      top_k2_eigenvectors = eigenvectors_sorted[:, :k2]
      df_pca_2d = df_scaled.dot(top_k2_eigenvectors)
      df_pca_2d = pd.DataFrame(df_pca_2d, columns=[f'PC{i+1}' for i in range(k2)])
```

- **k2 = 2** Sets the number of principal components to keep at 2 for 2D projection.
- **top_k2_eigenvectors = eigenvectors_sorted[:, :k2]** Selects the first two sorted eigenvectors.
- **df_pca_2d = df_scaled.dot(top_k2_eigenvectors)** Projects the standardized data onto the 2D principal component space.
- **df_pca_2d = pd.DataFrame(df_pca_2d, columns=[f'PC{i+1}' for i in range(k2)])** Converts the projected data into a labeled DataFrame with PC1 and PC2.

Step 6

```
[▶] # Step 6: Visualize 2D PCA
plt.figure(figsize=(8, 6))
plt.scatter(df_pca_2d['PC1'], df_pca_2d['PC2'], alpha=0.5, s=10)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA Projection (2D)')
plt.grid(True)
plt.tight_layout()
plt.show()
```

- **plt.figure(figsize=(8, 6))** Creates a new figure with specified size.
- **plt.scatter(df_pca_2d['PC1'], df_pca_2d['PC2'], alpha=0.5, s=10)** Plots the 2D PCA result as a scatter plot.
- **plt.xlabel('Principal Component 1')** Labels the x-axis.
- **plt.ylabel('Principal Component 2')** Labels the y-axis.
- **plt.title('PCA Projection (2D)')** Adds a title to the plot.
- **plt.grid(True)** Enables grid lines for better readability.



Step 7

```
[24] # Step 7: Select top k eigenvectors and project the data (3D)
      k3 = 3
      top_k3_eigenvectors = eigenvectors_sorted[:, :k3]
      df_pca_3d = df_scaled.dot(top_k3_eigenvectors)
      df_pca_3d = pd.DataFrame(df_pca_3d, columns=[f'PC{i+1}' for i in range(k3)])
```

- **k3 = 3** Sets the number of principal components to 3 for 3D projection.
- **top_k3_eigenvectors = eigenvectors_sorted[:, :k3]** Selects the first three sorted eigenvectors.
- **df_pca_3d = df_scaled.dot(top_k3_eigenvectors)** Projects the standardized data onto the 3D principal component space.
- **df_pca_3d = pd.DataFrame(df_pca_3d, columns=[f'PC{i+1}' for i in range(k3)])** Converts the projected data into a labeled DataFrame with PC1, PC2, and PC3.

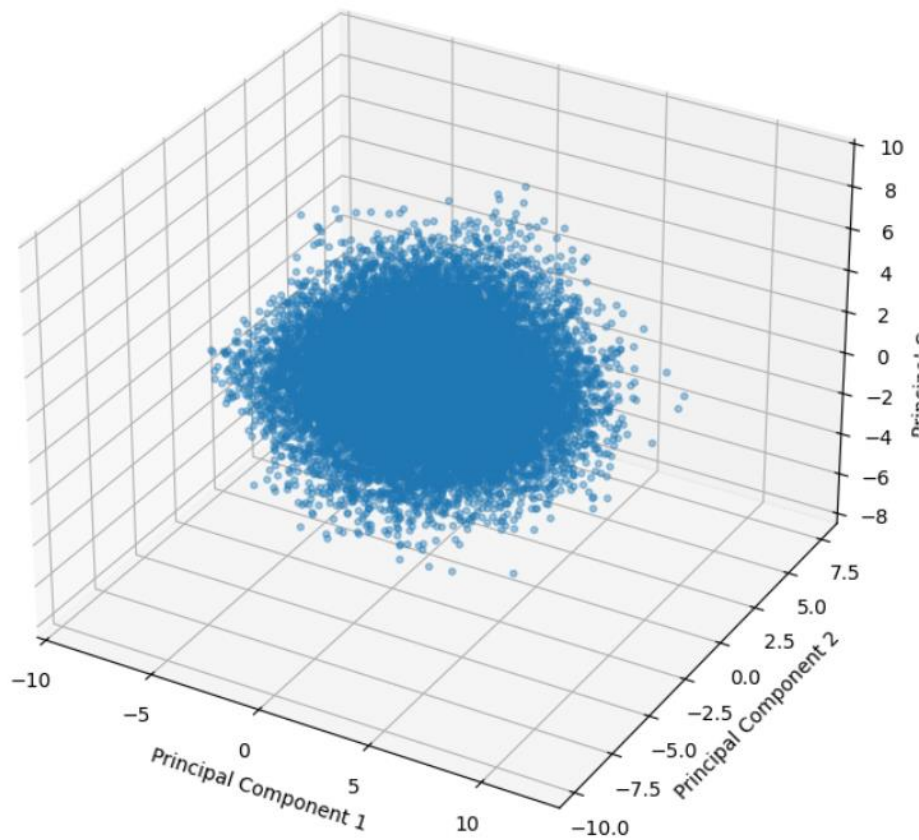
Step 8

```
# Step 8: Visualize 3D PCA
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df_pca_3d['PC1'], df_pca_3d['PC2'], df_pca_3d['PC3'], alpha=0.4, s=10)
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
ax.set_title('PCA Projection (3D)')
plt.tight_layout()
plt.show()
```

- **from mpl_toolkits.mplot3d import Axes3D** Imports 3D plotting tools for matplotlib.
- **fig = plt.figure(figsize=(10, 7))** Creates a new figure with specified size.
- **ax = fig.add_subplot(111, projection='3d')** Adds a 3D subplot to the figure.
- **ax.scatter(df_pca_3d['PC1'], df_pca_3d['PC2'], df_pca_3d['PC3'], alpha=0.4, s=10)** Plots the 3D PCA result as a scatter plot.
- **ax.set_xlabel('Principal Component 1')** Labels the x-axis.
- **ax.set_ylabel('Principal Component 2')** Labels the y-axis.
- **ax.set_zlabel('Principal Component 3')** Labels the z-axis.
- **ax.set_title('PCA Projection (3D)')** Adds a title to the 3D plot.

PCA Projection (3D)



Total summary

```
# Total sum of all eigenvalues
total_variance = np.sum(eigenvalues_sorted)

# Variance explained by each principal component
variance_explained = eigenvalues_sorted / total_variance

# Print variance explained by PC1, PC2, PC3
for i in range(3):
    ... print(f"PC{i+1} explains {variance_explained[i]*100:.2f}% of the total variance.")
```

PC1 explains 16.10% of the total variance.

PC2 explains 9.25% of the total variance.

PC3 explains 7.53% of the total variance.