# PROBLEM 1

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

Importing the required packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

Part (a)

```
# Loading the dataset and Visualization of the images
df = pd.read_csv('/content/drive/MyDrive/SEM 7/ID5055/Assignment
1/Problem 1/noisy_mnist.csv', header = None)

def plot_mnist_images(data):
    fig, arr = plt.subplots(10, 10, figsize = (5, 5))
    arr = arr.flatten()
    for img, a in zip(data, arr):
        a.imshow(img.reshape(16, 16), cmap = 'Greys')
        a.axis('off')

data = np.array(df)
plot_mnist_images(data)
```

Part (b)

```
# Normalize the data to the [0, 1] range
normalized_data = (data - np.min(data)) / (np.max(data) -
np.min(data))

# Train Test Split
data_train, data_test = train_test_split(normalized_data, train_size =
0.8)

# SVD
u, s, vt = np.linalg.svd(data_train)

# Function to get Principle Components
def get_PC(data, vt, n):
    PC = data@(vt.T)
    return PC[:, :n]
```

Part (c)

```
# Function to get Reconstruct
def reconst_datamatrix(PC, vt, n):
    new_data = PC@(vt[:n, :])
    return new_data

# Reconstructing the images
n = 100
```

```
PC_image = get_PC(data_train, vt, n)
reconstructed_images = reconst_datamatrix(PC_image, vt, n)
```

Part (d)

```
# Plotting the actual and reconstructed images

plot_mnist_images(data_train)
plot_mnist_images(reconstructed_images)
```

Observations

1. When we used n = 1 principle component then the reconstructed image was blur and detials were not clear.
2. As we increased the value of n image became more and more visible and clear and after a certain value of n no further significant improvements were observed.
3. For n = 256, the reconstruced image will be exactly equal to the original image because we are using all the components.

# Assignment 1 (PCA)

2) Given data is not standardized and columns have zero mean. (data is centered)

- Let data = $X^{m \times n}$

Now, SVD of $X$ = $U \Sigma V^T$

Covariance of $X$ ($C$) = $\left(\frac{1}{m}\right) X^T X$

$$C = \left(\frac{1}{m}\right) (U \Sigma V^T)^T (U \Sigma V^T)$$

$$C = \left(\frac{1}{m}\right) (V \Sigma U^T)(U \Sigma V^T)$$

$$\left[C = \left(\frac{1}{m}\right) (V \Sigma^2 V^T)\right] \quad \because (U^T U = I)$$

Since $C$ is a symmetric matrix it's eigen~~value~~ decomposition is given by

$$[C = v \Lambda v^T]$$

Comparing the above equations for $C$, we see that

- $V$ is analogous to $v$. (orthogonal matrix of eigenvectors)
- $\left(\frac{1}{m}\right)\Sigma^2$ is analogous to $\Lambda$ (diagonal matrix of eigenvalues)

# Thus, the eigenvalues of covariance matrix ($C$) are given by the squares of the singular values in $\Sigma$, scaled by $(1/m)$.

$$\boxed{\lambda_i = \left(\frac{1}{m}\right) \sigma_i^2}$$

# PROBLEM 3

Importing the required packages

```
import numpy as np
import pandas as pd
```

(a) Generating Data with 200 samples and 10 features and Performing PCA

```
np.random.seed(0)
num_samples = 200
num_features = 10
X = np.random.normal(size = (num_samples, num_features))

# Calculate the mean of the data
mean_vec = np.mean(X, axis=0)

# Center the data by subtracting the mean
X_centered = X - mean_vec

# Perform SVD on the centered data
U, S, Vt = np.linalg.svd(X_centered)

# Calculate the principal components
principal_components = Vt.T[:, :2]
```

(b) Calculate the Sum of Distances

```
# Project the centered data onto the plane formed by the first 2
principal components
projected_data = X_centered.dot(principal_components)

# Calculate distances between samples and the plane
distances = np.linalg.norm(X_centered -
projected_data.dot(principal_components.T), axis=1)

print("Sum of the distances between samples and plane:",
sum(distances))
```

```
Sum of the distances between samples and plane: 516.8355639975055
```

(c) Generate Random Planes and Calculate Distances

```
# Function to calculate sum of distances between samples and a given
plane
def sum_distances_to_plane(plane_normal, plane_point, samples):
    distances = np.abs(np.dot(samples - plane_point, plane_normal))
    return np.sum(distances)
```

```
# Generate 50 random planes and calculate their sum distances
num_random_planes = 50
sum_distances_random_planes = []

for _ in range(num_random_planes):
    # Generate a random plane by selecting two random data points
    random_indices = np.random.choice(num_samples, 2, replace=False)
    plane_point = X[random_indices[0]]
    plane_normal = X[random_indices[1]] - plane_point
    plane_normal /= np.linalg.norm(plane_normal)

    # Calculate sum of distances
    sum_distance = sum_distances_to_plane(plane_normal, plane_point,
X_centered)
    sum_distances_random_planes.append(sum_distance)

# Calculate sum of distances for the principal components plane
sum_distance_principal_components =
sum_distances_to_plane(principal_components[:, 0], mean_vec,
X_centered)

print("Sum of distances for random planes:",
sum_distances_random_planes)
print("Sum of distances for principal components plane:",
sum_distance_principal_components)
print("Is the sum of distances least for the principal components
plane?",
      sum_distance_principal_components <=
min(sum_distances_random_planes))
```

```
Sum of distances for random planes: [532.4151425093023,
504.54980425868644, 383.24388690332296, 569.3151214799045,
347.8249958494118, 463.83052069042196, 495.74597617112954,
406.91416471828677, 369.4232201230117, 313.645521812919,
708.3188777736088, 502.3677065704577, 682.212380523835,
513.0834713308132, 306.97550172055793, 568.5341764779935,
545.0026515857459, 563.6499801014721, 445.6141961483419,
192.7526016473585, 723.4837287706162, 563.5268825470104,
251.32073243060225, 254.375835095368, 287.8296837327546,
578.7507338852724, 261.4196006115002, 451.3739255806056,
317.320929203768, 310.58919427183474, 580.0401418572764,
842.5976986582161, 318.06469374394146, 640.9027759548535,
439.77815638866116, 765.5999913716325, 414.1323930231304,
454.35926577271465, 309.8732762784041, 689.4453784896366,
372.8319253106386, 567.2659974546001, 261.2452741489482,
199.94319239805793, 432.2276505588093, 426.5191597484502,
428.81335915460187, 559.1882378712977, 327.99835859539667,
372.0734546668329]
Sum of distances for principal components plane: 179.66979828741438
Is the sum of distances least for the principal components plane? True
```

Hence it is verified that the distance is minimum for the plane obtained by the PCA

# PROBLEM 4

Importing the necessary packages

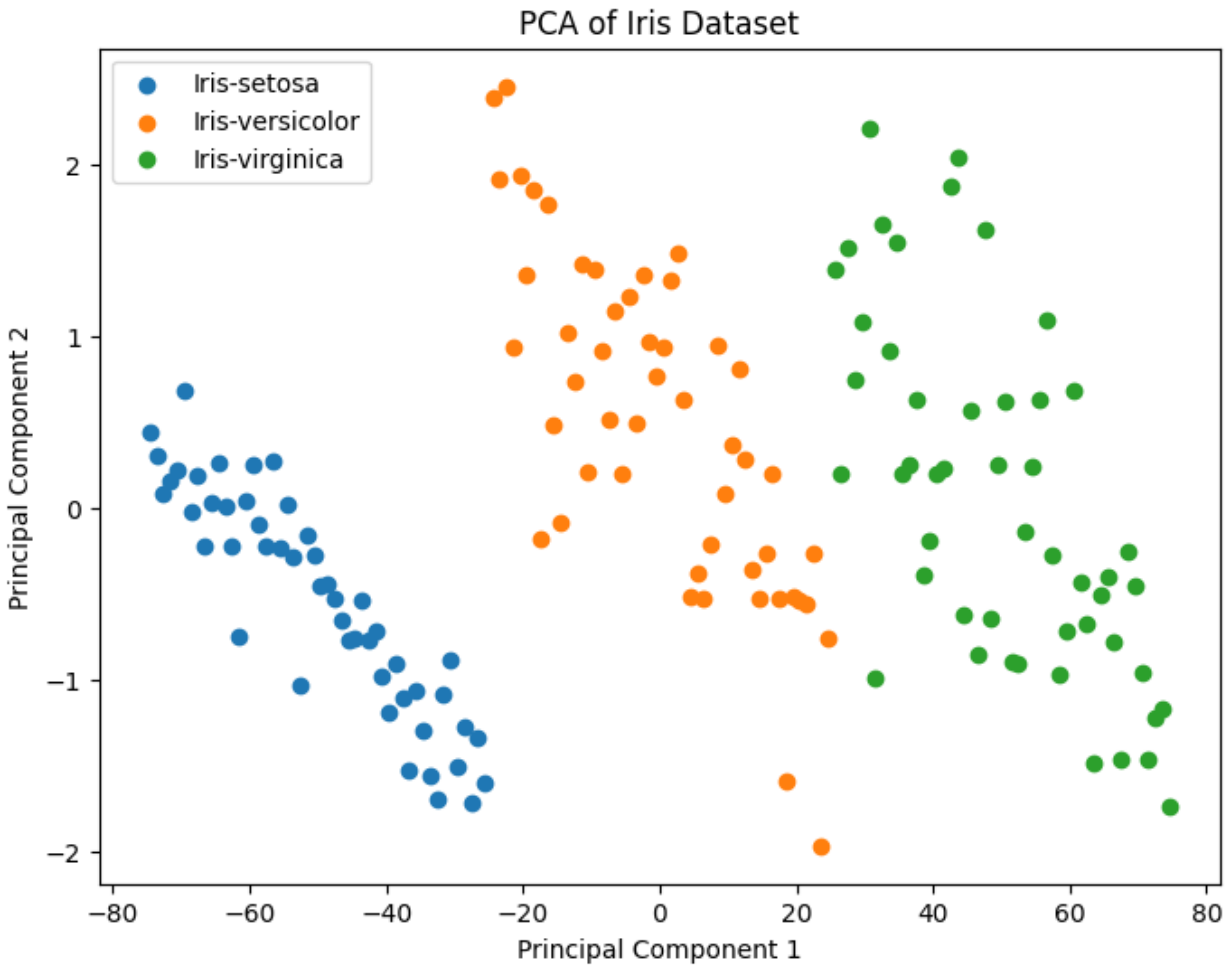
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

(a) Perform PCA on the given dataset and reduce the dimension to two

```
df = pd.read_csv('/content/drive/MyDrive/SEM 7/ID5055/Assignment
1/Problem 4/Iris.csv')
df.columns = ['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm',
'PetalWidthCm', 'Species']
x = df.drop('Species', axis = 1)
y = df['Species']

# Centralise the data and SVD calculation
mean = np.mean(x, axis = 0)
x_center = x - mean
u, s, vt = np.linalg.svd(x_center)

# First two Principle Component Calculations
pca_components = vt.T[:, :2]
x_pca = x_center @ (vt.T[:, :2])
```

(b) Plot the newly obtained features (Principal components) discriminating the species type

```
plt.figure(figsize=(8, 6))
for species in np.unique(y):
    plt.scatter(x_pca.loc[y == species, 0], x_pca.loc[y == species,
1], label=species)

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Iris Dataset')
plt.legend()
plt.show()
```




(c) Identify which one of the four features can be used to discriminate between the species (Use the coefficient of principal components to answer)

```
coefficients = pca_components.T  # Coefficients of principal components

# Print the coefficients of the principal components for each original feature
for i, feature_name in enumerate(x.columns):
    print(f'Coefficient for {feature_name}: {coefficients[:, i]}')
```

```
Coefficient for Id: [ 0.99913157 -0.04071433]
Coefficient for SepalLengthCm: [0.01365776 0.49432522]
Coefficient for SepalWidthCm: [-0.00396685 -0.01499187]
Coefficient for PetalLengthCm: [0.0358369  0.82236709]
Coefficient for PetalWidthCm: [0.01579744 0.2783389 ]
```

The larger the absolute value of the coefficient for a particular feature, the more influence that feature has on the corresponding principal component. In other words, a feature with a larger coefficient contributes more to the separation of the data in the PCA space.

From these coefficients, we can see that the feature "PetalLengthCm" has the largest absolute coefficients for both principal components. This indicates that "PetalLengthCm" contributes the most to the separation of the data points in the PCA space. Therefore, "PetalLengthCm" is likely the feature that can be used to discriminate between the species most effectively based on the PCA results.

# PROBLEM 5

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

Part (a)

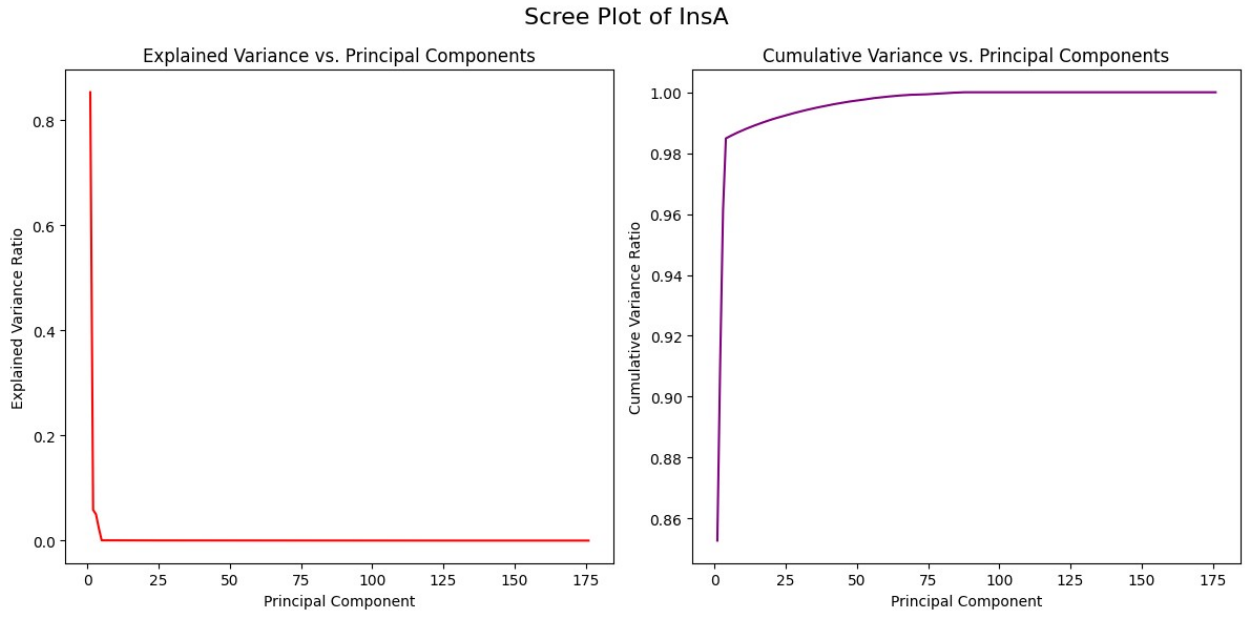
```
# Function to get principle components
def get_pc(x):
  u, s, vt = np.linalg.svd(x)
  principle_components = vt.T
  return principle_components

# Function to get the explained variance for the Scree plot
def exp_var(X):
  eigenvalues = np.linalg.eigvals(np.cov(X.T))
  explained_variance_ratio = eigenvalues / np.sum(eigenvalues)
  return explained_variance_ratio
```

InsA Analysis

```
dataA = pd.read_csv('/content/drive/MyDrive/sem 7/ID5055/Assignment
1/Problem 5/InsA.csv')

# Centering the data
mean = np.mean(dataA, axis=0)
scaled_data = dataA - mean

# Getting the principle components
pca = scaled_data@get_pc(scaled_data)

# Scree plot to visualize explained variance by each principal
component
explained_variance = exp_var(pca)
cumulative_variance = np.cumsum(explained_variance)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Subplot 1: Explained Variance vs. Principal Components
ax1.plot(range(1, len(explained_variance) + 1), explained_variance,
color='red')
ax1.set_xlabel('Principal Component')
```

```
ax1.set_ylabel('Explained Variance Ratio')
ax1.set_title('Explained Variance vs. Principal Components')

# Subplot 2: Cumulative Variance vs. Principal Components
ax2.plot(range(1, len(cumulative_variance) + 1), cumulative_variance,
color='purple')
ax2.set_xlabel('Principal Component')
ax2.set_ylabel('Cumulative Variance Ratio')
ax2.set_title('Cumulative Variance vs. Principal Components')

fig.suptitle('Scree Plot of InsA', fontsize=16)
plt.tight_layout()
plt.show()

# Implementing the "knee" point detection algorithm
total_variance = np.sum(explained_variance)
knee_threshold = 0.9  # adjustable

num_components = np.argmax(cumulative_variance >= knee_threshold *
total_variance) + 1

print(f"Number of components selected: {num_components}")
```

```
/usr/local/lib/python3.10/dist-packages/matplotlib/cbook/
__init__.py:1335: ComplexWarning: Casting complex values to real
discards the imaginary part
  return np.asarray(x, float)
```

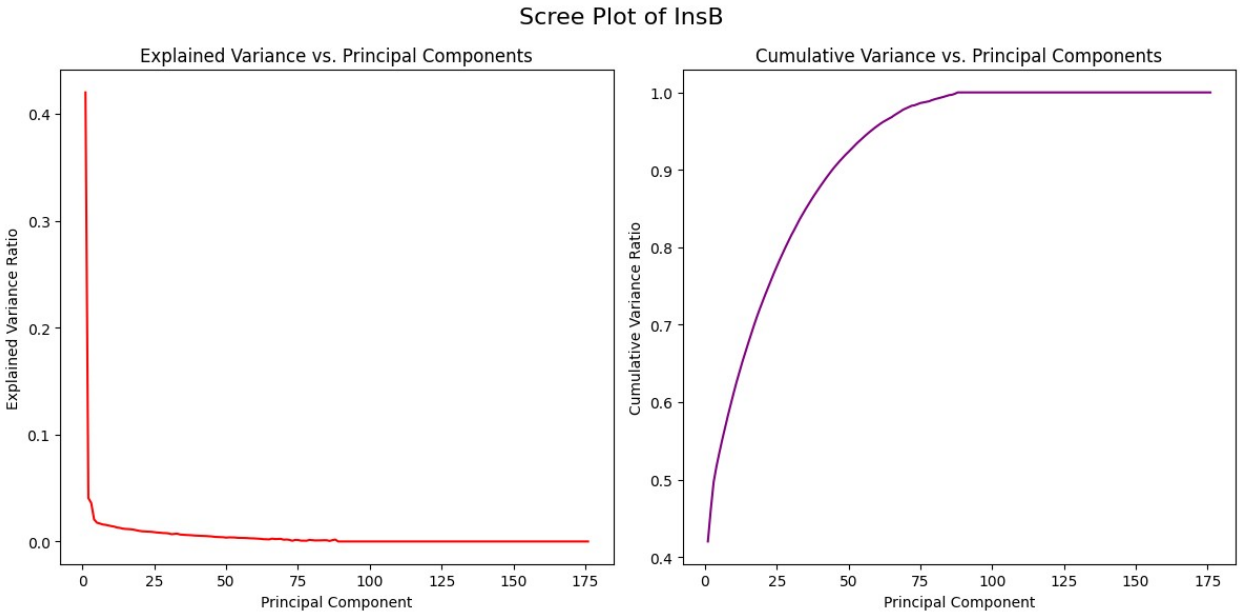



```
Number of components selected: 2
```

InsB Analysis

```
dataB = pd.read_csv('/content/drive/MyDrive/sem 7/ID5055/Assignment
1/Problem 5/InsB.csv')

# Centering the data
mean = np.mean(dataB, axis=0)
scaled_data = dataB - mean

# Getting the principle components
pca = scaled_data@get_pc(scaled_data)

# Scree plot to visualize explained variance by each principal
component
explained_variance = exp_var(pca)
cumulative_variance = np.cumsum(explained_variance)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Subplot 1: Explained Variance vs. Principal Components
ax1.plot(range(1, len(explained_variance) + 1), explained_variance,
color='red')
ax1.set_xlabel('Principal Component')
ax1.set_ylabel('Explained Variance Ratio')
ax1.set_title('Explained Variance vs. Principal Components')

# Subplot 2: Cumulative Variance vs. Principal Components
ax2.plot(range(1, len(cumulative_variance) + 1), cumulative_variance,
color='purple')
ax2.set_xlabel('Principal Component')
ax2.set_ylabel('Cumulative Variance Ratio')
ax2.set_title('Cumulative Variance vs. Principal Components')

fig.suptitle('Scree Plot of InsB', fontsize=16)
plt.tight_layout()
plt.show()

# Implementing the "knee" point detection algorithm
total_variance = np.sum(explained_variance)
knee_threshold = 0.9  # adjustable

num_components = np.argmax(cumulative_variance >= knee_threshold *
total_variance) + 1

print(f"Number of components selected: {num_components}")
```

```
/usr/local/lib/python3.10/dist-packages/matplotlib/cbook/
__init__.py:1335: ComplexWarning: Casting complex values to real
discards the imaginary part
  return np.asarray(x, float)
```



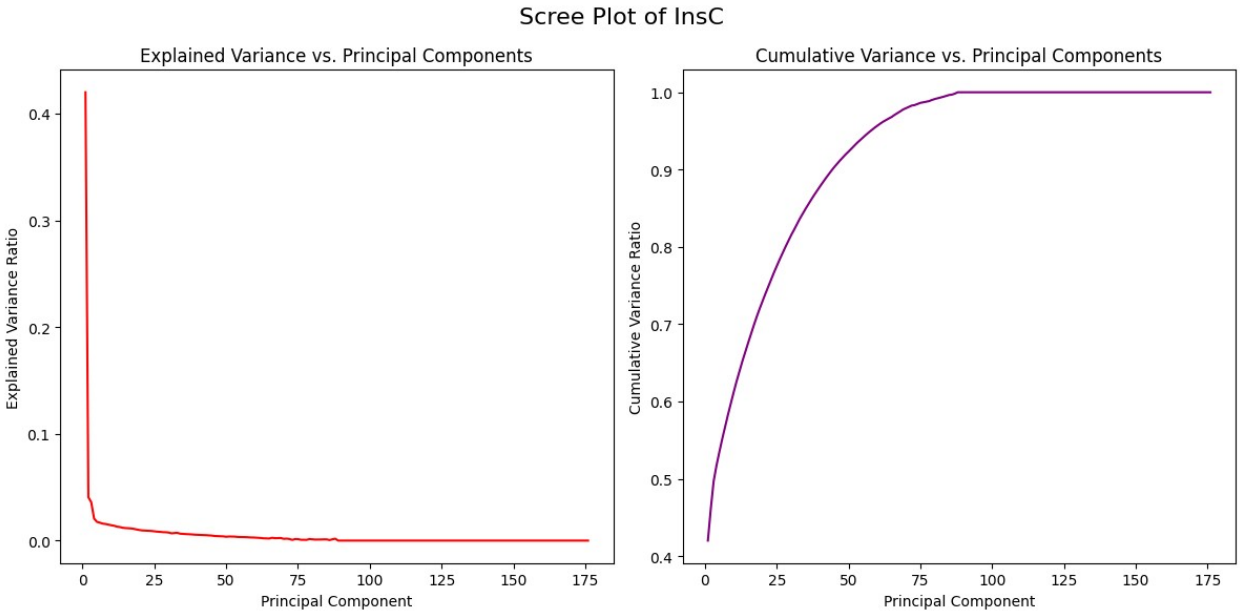

```
Number of components selected: 45
```

InsC Analysis

```
dataC = pd.read_csv('/content/drive/MyDrive/sem 7/ID5055/Assignment
1/Problem 5/InsC.csv')

# Centering the data
mean = np.mean(dataC, axis=0)
scaled_data = dataC - mean

# Getting the principle components
pca = scaled_data@get_pc(scaled_data)

# Scree plot to visualize explained variance by each principal
component
explained_variance = exp_var(pca)
cumulative_variance = np.cumsum(explained_variance)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Subplot 1: Explained Variance vs. Principal Components
ax1.plot(range(1, len(explained_variance) + 1), explained_variance,
color='red')
ax1.set_xlabel('Principal Component')
ax1.set_ylabel('Explained Variance Ratio')
ax1.set_title('Explained Variance vs. Principal Components')

# Subplot 2: Cumulative Variance vs. Principal Components
ax2.plot(range(1, len(cumulative_variance) + 1), cumulative_variance,
color='purple')
```

```
ax2.set_xlabel('Principal Component')
ax2.set_ylabel('Cumulative Variance Ratio')
ax2.set_title('Cumulative Variance vs. Principal Components')

fig.suptitle('Scree Plot of InsC', fontsize=16)
plt.tight_layout()
plt.show()

# Implementing the "knee" point detection algorithm
total_variance = np.sum(explained_variance)
knee_threshold = 0.9  # adjustable

num_components = np.argmax(cumulative_variance >= knee_threshold *
total_variance) + 1

print(f"Number of components selected: {num_components}")
```

```
/usr/local/lib/python3.10/dist-packages/matplotlib/cbook/
__init__.py:1335: ComplexWarning: Casting complex values to real
discards the imaginary part
  return np.asarray(x, float)
```




```
Number of components selected: 45
```

Part (b)

```
# Load the new test data
test_data = pd.read_csv('/content/drive/MyDrive/sem
7/ID5055/Assignment 1/Problem 5/Test_data.csv')

mean = np.mean(test_data, axis=0)
```

```
test_data_centered = test_data - mean

# Perform PCA on the test data using the same PCA object
pca = get_pc(test_data_centered)[:, :9]
# Using the rule of thumb I opted to use first 9 principle components

transformed_data = test_data_centered @ pca
reconstructed_data = np.dot(transformed_data, pca.T)
reconstructed_data = reconstructed_data + mean.values

# Calculate the reconstruction error for each test sample
reconstruction_errors = np.mean(np.square(test_data -
reconstructed_data), axis=1)

# Identify the contaminated sample
contaminated_sample_index = np.argmax(reconstruction_errors)
print(f"The contaminated sample is at index:
{contaminated_sample_index}")
```

```
The contaminated sample is at index: 3
```

The contaminated sample might have higher residuals compared to the rest of the samples. So I looked for samples with significantly larger residuals than the others. This could be indicative of the sample with contaminants and it turned out that sample at index 3 has the highest residual.

6) Given $X = [x_1, x_1 \dots x_0]^T$, $X \in R^{D \times N}$

a) we have $X = U \Sigma V^T = U_1 \Sigma_1 V_1^T + U_2 \Sigma_2 V_2^T$

$U_1 \in R^{D \times M}$ $\quad U_2 \in R^{D \times (D-M)}$

$V_1 \in R^{N \times M}$ $\quad V_2 \in R^{N \times (D-M)}$

$\Sigma_1$ = M-dim diagonal matrix $\quad \Sigma_2 \in (D-M)$-dim diagonal matrix.

To prove: $[U_{2,i}^T \cdot X = 0]$ , ($U_{2,i}$ = $i^{th}$ column of $U_2$)

- By proving this we are proving that $U_{2,i}^T$ is null space of X.

- for any column vector x, we can express it using $U \cdot \alpha$, where $\alpha$ is a D-dimensional vector representing the coefficients of x in the basis of U's columns.

- Since x lies in the subspace spanned by U. then the projection of x onto $U_2$ column is orthogonal to $U_1 \Rightarrow U_1^T \cdot U_{2,i} = 0$

Now,

$$x = U\alpha$$

$$U_{2,i}^T x = (U_{2,i}^T)(U\alpha)$$

Also, $U_{2,i} = U_2 e_i$ ( $e_i$: standard basis vector)

Now, $U_{2,i}^T \cdot x = e_i^T U_2^T U \alpha$

$\therefore U_1^T U_2 = 0$ So $e_i^T U_2^T U \cdot \alpha = e_i^T U_1^T U \cdot \alpha = 0$

Hence $\boxed{U_{2,i}^T \cdot X = 0}$

b) given $X = U \Sigma V^T$

where $U \in R^{D \times D}$, $\Sigma \in R^{D \times N}$, $V \in R^{N \times N}$

$U^T U = I$

$V^T V = V V^T = I$

we have $Z = \Sigma^{-1/2} U^T X$

$$Cov(Z) = \frac{1}{D} [Z Z^T]$$

$$= \frac{1}{D} \left[ (\Sigma^{-1/2} U^T X)(\Sigma^{-1/2} U^T X)^T \right]$$

$$= \frac{1}{D} \left[ \Sigma^{-1/2} U^T X \cdot X^T U \Sigma^{-1/2} \right]$$

$$= \frac{1}{D} \left[ \Sigma^{-1/2} U^T (U \Sigma V^T) \cdot (V \Sigma U^T) U \Sigma^{-1/2} \right]$$

$$= \frac{1}{D} \left[ \Sigma^{-1/2} (U^T U)(\Sigma^2)(U^T U) \Sigma^{-1/2} \right]$$

$$Cov(Z) = \frac{1}{D} [\Sigma]$$

So, finally we have $Cov(Z) = \frac{1}{D} (\Sigma)$

which is identity matrix after multiplication by $(\frac{1}{D})$