```
import pandas as pd
import numpy as np
import sys
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

filename = sys.argv[1]
# url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/magic/magic04.data"
data = pd.read_csv(filename, names = ["Column_"+str(i) for i in range(1, 12)])

numeric_data = data.select_dtypes(include=[np.number])
mean = np.mean(numeric_data, axis=0)
std = np.std(numeric_data, axis=0)
normalized_data = (numeric_data - mean) / std
data[numeric_data.columns] = normalized_data
last_column_values = data.iloc[:, -1]
data = data.iloc[:, :-1]

#Using np.cov
covariance_matrix = np.cov(data, rowvar=False)
rounded_covariance_matrix = np.round(covariance_matrix, 9)
print("="*100)
print("\nNumpy covarience matrix:\n ", rounded_covariance_matrix)
print()

#Computing Manually
num_columns = data.shape[1]
covariance_matrix = []

for i in range(num_columns):
    row = []
    for j in range(num_columns):
        cov = ((data.iloc[:, i] - data.iloc[:, i].mean()) * (data.iloc[:, j] -
data.iloc[:, j].mean())).sum() / data.shape[0]
        row.append(cov)
    covariance_matrix.append(row)

covariance_matrix = pd.DataFrame(covariance_matrix)
print("\nManual covarience matrix:\n ", covariance_matrix)
print()

#Using linalg
eigenvalues, eigenvectors = np.linalg.eig(rounded_covariance_matrix)
dominant_eigenvalue_index = np.argmax(eigenvalues)
dominant_eigenvalue = eigenvalues[dominant_eigenvalue_index]
dominant_eigenvector = eigenvectors[:, dominant_eigenvalue_index]

print("="*100)
print("\nDominant Eigenvalue(Numpy):\n", dominant_eigenvalue)
print("Dominant Eigenvector(Numpy):\n", dominant_eigenvector)
print()

#Computing Manually
def power_iteration(matrix, num_iterations=1000, tolerance=1e-6):
    n = matrix.shape[0]
    vector = np.random.rand(n)
    vector /= np.linalg.norm(vector)
```

```
    for _ in range(num_iterations):
        result = np.dot(matrix, vector)
        norm = np.linalg.norm(result)
        vector = result / norm
        eigenvalue = np.dot(vector, np.dot(matrix, vector))
        if np.abs(eigenvalue - norm) < tolerance:
            break
    return eigenvalue, vector

dominant_eigenvalue, dominant_eigenvector =
power_iteration(rounded_covariance_matrix)
print("\nDominant Eigenvalue:\n", dominant_eigenvalue)
print("Dominant Eigenvector:\n", dominant_eigenvector)
magnitude = np.linalg.norm(dominant_eigenvector)
normalized_dominant_eigenvector = dominant_eigenvector / magnitude
print("Normalized Dominant Eigenvector:\n")
print(normalized_dominant_eigenvector)
print()


# d
eigenvalues, eigenvectors = np.linalg.eig(rounded_covariance_matrix)
eigenvalue_order = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[eigenvalue_order]
eigenvectors = eigenvectors[:, eigenvalue_order]
dominant_eigenvectors = eigenvectors[:, :2]
projected_data = np.dot(data, dominant_eigenvectors)
variance = np.var(projected_data, axis=0)
print("="*100)
print("\nDominant Eigenvectors (first two):\n", dominant_eigenvectors)
print("\nProjected Data:", projected_data)
print("Variance in the Projected Subspace:\n", variance)
print()


#e
eigenvalues, eigenvectors = np.linalg.eig(rounded_covariance_matrix)
eigenvalue_order = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[eigenvalue_order]
eigenvectors = eigenvectors[:, eigenvalue_order]
eigen_decomposition_covariance = np.dot(np.dot(eigenvectors, np.diag(eigenvalues)),
np.linalg.inv(eigenvectors))
print("="*100)
print("\nEigenvalues:", eigenvalues)
print("Covariance Matrix in Eigen Decomposition Form:\n",
eigen_decomposition_covariance)
print()

#f
dominant_eigenvectors = eigenvectors[:, :2]
projected_data = np.dot(data, dominant_eigenvectors)
reconstructed_data = np.dot(projected_data, dominant_eigenvectors.T)
mse = np.mean(np.square(data - reconstructed_data), axis=0)
sum_eigenvalues_except_first_two = np.sum(eigenvalues[2:])
print("="*100)
print("\nMean Square Error (MSE):\n", mse)
print("Sum of Eigenvalues except the First Two:\n",
sum_eigenvalues_except_first_two)
```

```
#g
pca = PCA(n_components=2)
principal_components = pca.fit_transform(data.iloc[:, :-1].values )

# Create a scatter plot with different colors for each class
plt.figure(figsize=(10, 6))
classes = np.unique(last_column_values)
colors = ['y', 'k']

for i, cls in enumerate(classes):
    plt.scatter(
        principal_components[last_column_values == cls][:, 0],
        principal_components[last_column_values == cls][:, 1],
        label=f'Class {cls}',
        color=colors[i],
        alpha=0.7,
    )

plt.title("Data Points After PCA Projection")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend()
plt.show()

#h
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/magic/magic04.data"
data = pd.read_csv(url, names = ["Column_"+str(i) for i in range(1, 12)])
X = data.iloc[:, :-1].values  # Features
y = last_column_values      # Class labels
eigenvalue_order = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[eigenvalue_order]
eigenvectors = eigenvectors[:, eigenvalue_order]
total_variance = np.sum(eigenvalues)
cumulative_variance = np.cumsum(eigenvalues) / total_variance
num_components_to_preserve_variance = np.argmax(cumulative_variance >= 0.95) + 1

selected_principal_components =
eigenvectors[:, :num_components_to_preserve_variance]

projected_data = np.dot(X - np.mean(X, axis=0), selected_principal_components)


print("Coordinates of the first 10 data points in the new basis:\n")
for i in range(10):
    print(f"Data Point {i + 1}: {projected_data[i]}")
```