Assignment 3

Lucas Doan (261212276)

MGSC 661- 275: Multivariate Statistical for Analysts

Professor: Sanjith Gopalakrishnan

Desautels Faculty of Management

Summer 2024

Table of Contents

Part 1: Image Compression using K-Means (50 points)

a. (5 points) Pick any colour image of your choice. You can use an image from your personal collection or download a sample image from the internet. If the run-time of your algorithm is too slow, you may need to choose a lower-resolution image. Use an appropriate Python library to load the image. Alternatively, you can use the flower.jpg from sklearn.datasets.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_sample_image
from sklearn.cluster import KMeans
from sklearn.metrics import mean_squared_error

#a. Pick an image
flower = load_sample_image("flower.jpg")
```

b. (5 points) Convert the image to a two-dimensional array where each row represents a pixel and each column represents a colour channel (RGB values).

```
#b. Convert image to 2D array

image_array = flower.reshape(-1,3)
```

c. (10 points) Implement k-means clustering to cluster the pixel colors into k clusters (experiment with different values of k such as 16, 32, 64, etc.)

```
#c. Implement K-Means Clustering
k = int(input("Enter the number of pixel color clusters: "))
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(image_array)
```

```
Enter the number of pixel color clusters:
16
```

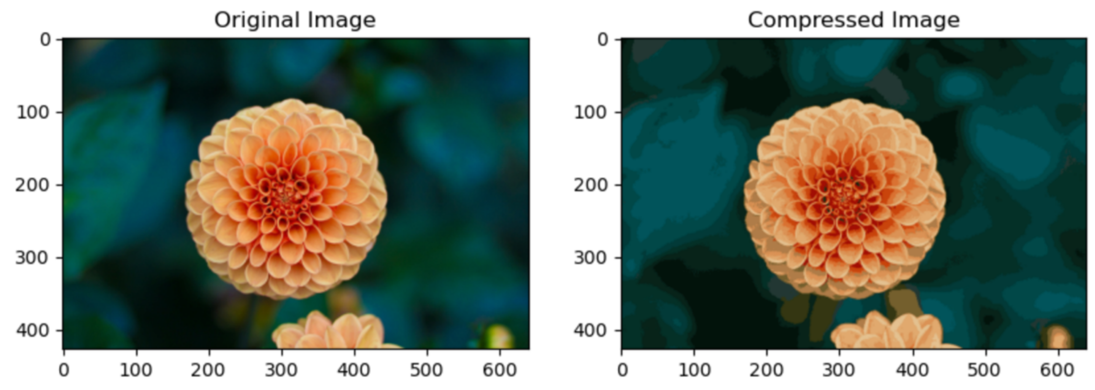
```
[21]
▾ KMeans
KMeans(n_clusters=16, random_state=42)
```

d. (5 points) Replace each pixel's colour with the centroid of the cluster it belongs to.

```
#d. Replace Pixel Colors
new_image_array = kmeans.cluster_centers_[kmeans.labels_].reshape(image_array.shape)
```

e. (10 points) Reconstruct the compressed image from the clustered pixel data. Compare the original image and the compressed image by visualizing them side by side. Define a metric to quantify the compression achieved, if at all, for the image

```
#e. Reconstructing the compressed image
reshaped_image_array = new_image_array.reshape(flower.shape)
compressed_image = reshaped_image_array.astype(np.uint8)

#Visualize and compare image
plt.figure(figsize=(10,5))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(flower)
plt.subplot(1, 2, 2)
plt.title('Compressed Image')
plt.imshow(compressed_image)
plt.show()
```



f. (15 points) Define a metric to quantify the loss of quality, if at all, between the original image and the compressed image. Discuss the trade-off between the number of clusters and the quality of the image and the compression achieved.

```
#f. Loss of quality

quality_loss = mean_squared_error(flower.flatten(), compressed_image.flatten())

print(f'Quality Loss (MSE): {quality_loss}')
```

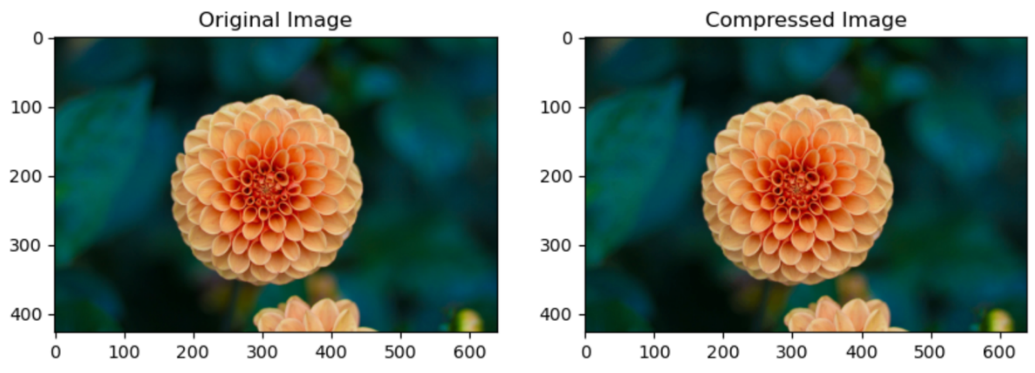
```
Quality Loss (MSE): 41.71264027127244
```

The current quality loss of 41.71 is a result of using 16 clusters. According to the comparison between original and compressed image, it can be see that compressed image is more blurry.
Comparing with 1024 clusters:

```
#c. Implement K-Means Clustering
k = int(input("Enter the number of pixel color clusters: "))
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(image_array)
```

```
▾ KMeans
KMeans(n_clusters=1024, random_state=42)
```

[29]



```
[30] #f. Loss of quality

     quality_loss = mean_squared_error(flower.flatten(), compressed_image.flatten())

     print(f'Quality Loss (MSE): {quality_loss}')
```

```
[30] Quality Loss (MSE): 3.666899638953942
```

It can be clearly seen that the quality of compressed image is significantly improved after the increase in clusters; the quality loss reducing from 41.71 to 3.66 and the image is much less blurry at the background.
In conclusion, the more number of clusters, the better of quality of the image and conversely.

Part 2: Principal Component Analysis (50 points)

a. (5 points) Load the dataset and confirm it has loaded correctly.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

```
#a. Load the data
car = pd.read_csv('mtcars.csv')
car_df = pd.DataFrame(car)

car_df['Japan_class'] = np.where(car_df['country'] == 'Japan', 1, 0)
car_df['US_class'] = np.where(car_df['country'] == 'US', 1, 0)
car_df['Europe_class'] = np.where(car_df['country'] == 'Europe', 1, 0)

car_df = car_df.drop(columns = ['country', 'model'])
display(car_df)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	Japan_class	US_class	Europe_class
0	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	1	0	0
1	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	1	0	0
2	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	1	0	0
3	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	0	1	0
4	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2	0	1	0
5	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1	0	1	0
6	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4	0	1	0
7	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2	0	0	1
8	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2	0	0	1
9	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4	0	0	1
10	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4	0	0	1
11	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3	0	0	1
12	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3	0	0	1
13	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3	0	0	1
14	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4	0	1	0
15	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4	0	1	0
16	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4	0	1	0
17	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1	0	0	1
18	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2	1	0	0
19	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1	1	0	0
20	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1	1	0	0

b. (5 points) Standardize all numerical features in the dataset to have a mean of 0 and a standard deviation of 1. This is an essential preprocessing step in many machine learning algorithms and statistical techniques.

```
#b. Standardize all numerical features
scaler = StandardScaler()
standardized_data = scaler.fit_transform(car_df)
```

c. (15 points) Apply PCA to the standardized numerical features to reduce it to the first two principal components. Also, explain the percentage of variance explained by each principal component.

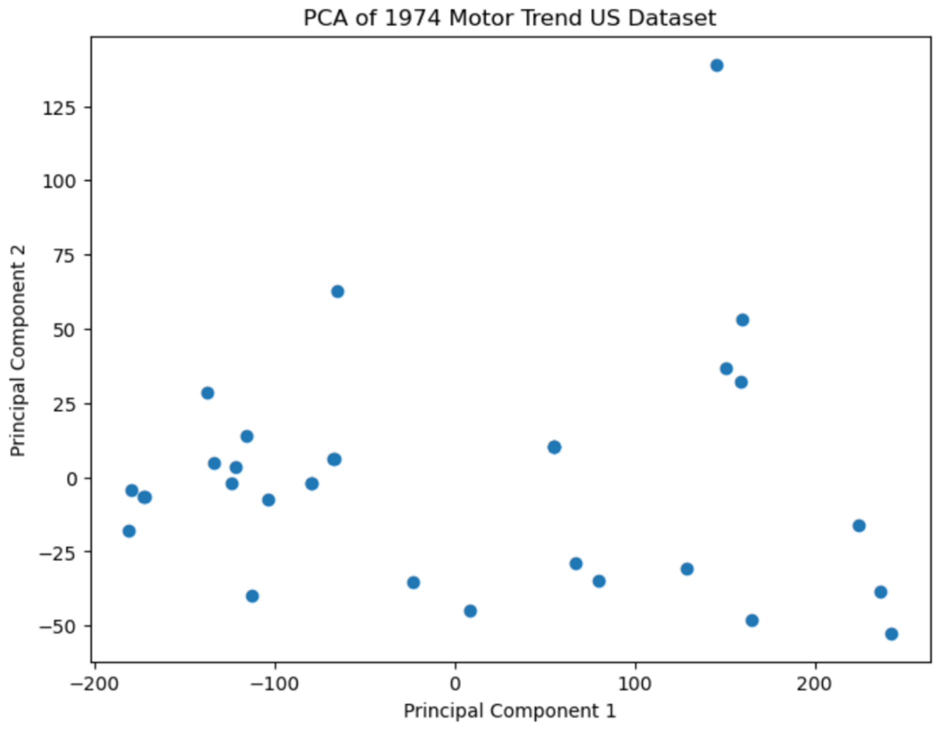
```
#c. Apply PCA
pca = PCA(n_components=2)
car_pca = pca.fit_transform(car_df)

print("Explained Variance:", pca.explained_variance_ratio_)
```

```
Explained Variance: [0.9269794 0.0723694]
```

The first value 92.69% means that the PCA captures approximately 92.69% of the total variance in the data. From this, it can be explained that a large majority of the dataset's variance can be represented along this single axis.
The second value 7.24% means that the second PCA captures approximately 7.24% of the total variance. While this is less than the first value, it explained the rest 7.24% of dataset's variance, which can't be captured by the first PCA.

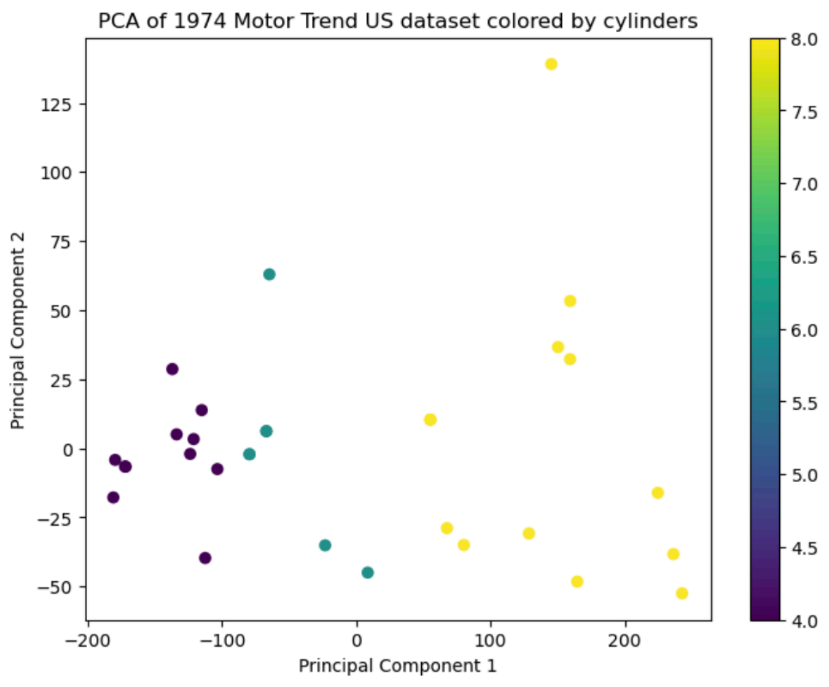
d. (5 points) Create a scatter plot of the data points in the new 2-dimensional space defined by the principal components.

```
#d. Scatter plot
pca_df = pd.DataFrame(data= car_pca, columns=['PC1', 'PC2'])
plt.figure(figsize=(8, 6))
plt.scatter(pca_df['PC1'], pca_df['PC2'])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of 1974 Motor Trend US Dataset')
plt.show()
```



e. (10 points) Colour-code the observations by the number of cylinders (cyl). Discuss the separation of the different cylinder categories in the PCA plot. Which cylinder categories are most clearly separated?

```
#e. Colour-code the observations by the number of cylinders (cyl).

pca_df['cyl'] = car_df['cyl']
plt.figure(figsize=(8, 6))
scatter = plt.scatter(pca_df['PC1'], pca_df['PC2'], c=pca_df['cyl'], cmap='viridis')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of 1974 Motor Trend US dataset colored by cylinders')
plt.colorbar(scatter)
plt.show()
```



4-Cylinder Cars:

This type of car clusters at the lower of PC1 values, expressing that their features are significantly different from the higher-cylinder cars. Additionally, they have a tight cluster, suggesting that they are similar to each other.

6-Cylinder Cars:

These cars are located in the middle graph, indicating that features captured by the principal components are relatively relevant. Even though the cluster is as not tight as 4-cylinders, it still form a cluster suggesting that they are similar to each other.

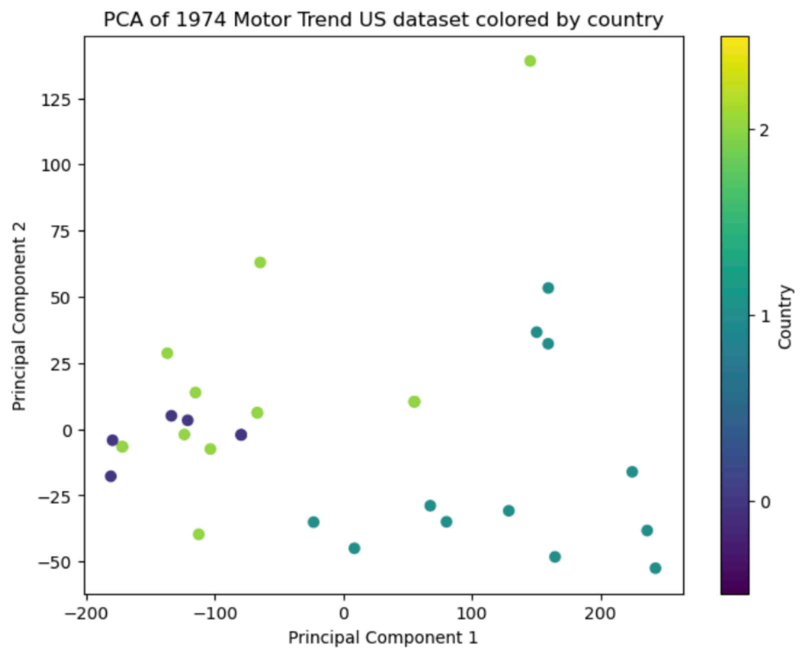
8-Cylinder Cars:

These cars are located at the higher of PC1 values. The separation suggests that these cars have characteristics which are significantly different from the 4-cylinder cars, including higher power, larger displacement, and increased weight.

f. (10 points) Next, colour-code the observations by the country of the models. Discuss and interpret the separation of the cars from different countries in the PCA plot.

```python
#f.Colour-code the observations by the country of the models.
country_labels, country_uniques = pd.factorize(car['country'])
for index, country in enumerate(country_uniques):
    print(f"Country: {country}, Code: {index}")

plt.figure(figsize=(8, 6))
scatter = plt.scatter(pca_df['PC1'], pca_df['PC2'], c=country_labels, cmap='viridis')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of 1974 Motor Trend US dataset colored by country')
colorbar = plt.colorbar(scatter, ticks=range(len(country_uniques)), label='Country')
colorbar.set_ticklabels(country_uniques)
plt.clim(-0.5, len(country_uniques) - 0.5)
plt.show()
```

```
Country: Japan, Code: 0
Country: US, Code: 1
Country: Europe, Code: 2
```



- Japanese cars are mainly located at the lower of PC1, reflecting smaller engine sizes and better fuel efficiency.

- European cars are spread across the middle of the PCA graph, indicating a balance between engine sizes and efficiency.

- US cars also cluster on the right side and some at the middle of the PCA plot, indicating larger enginesizes and higher horsepower.