```
pip install numpy pandas matplotlib seaborn s
```

Show hidden output

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardSca
from sklearn.decomposition import PCA
from sklearn.model_selection import train_tes
from sklearn.linear_model import LogisticRegr
from sklearn.metrics import silhouette_score,


# Load the dataset
url = "/content/Mall_Customers.csv"
df = pd.read_csv(url)

# Check the updated dataframe
df.head(10)
```

	CustomerID	Gender	Age	Annual Income (k$)	Spend Sc (1-1
0	1	Male	19	15	
1	2	Male	21	15	
2	3	Female	20	16	
3	4	Female	23	16	
4	5	Female	31	17	
5	6	Female	22	17	
6	7	Female	35	18	
7	8	Female	23	18	
8	9	Male	64	19	

Start coding or generate with AI.

```
df.info()
```

Show hidden output

Start coding or generate with AI.

Here, we can see that there are no missing values in the columns.

Start coding or generate with AI.

```
def histogram():
  """
  Histogram function to plot the distribution
  """
```

Mall_Customers.csv FileNotFoundError

You

Please explain the error:

```
FileNotFoundError: [Errno 2] No such file or directory: '/Mall_Customers.
```

Gemini

Hypothesis

The error message `FileNotFoundError: [Errno 2] No such file or directory: '/Mall_Customers.csv'` clearly indicates that the Python interpreter cannot find the file "Mall_Customers.csv" at the specified path '/Mall_Customers.csv'.

This is happening because the code is trying to access the file using an absolute path '/Mall_Customers.csv', assuming it's located at the root of the filesystem. However, it is most likely that the file is present in a different location within your working directory or in a directory where the notebook is running.

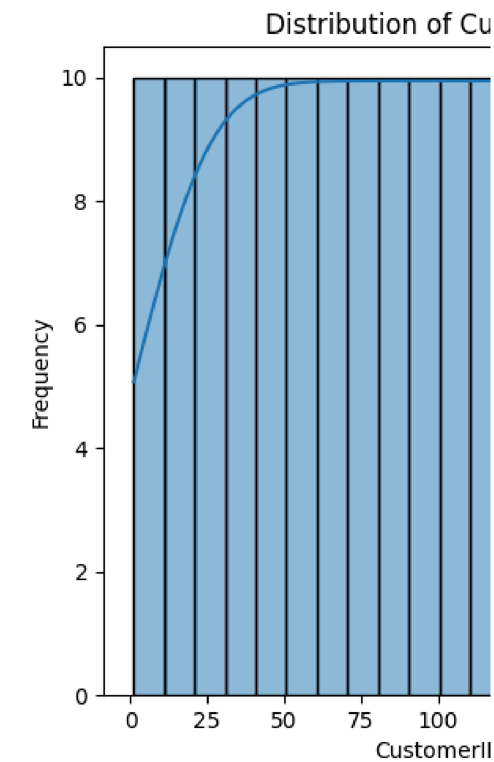
Suggested Changes

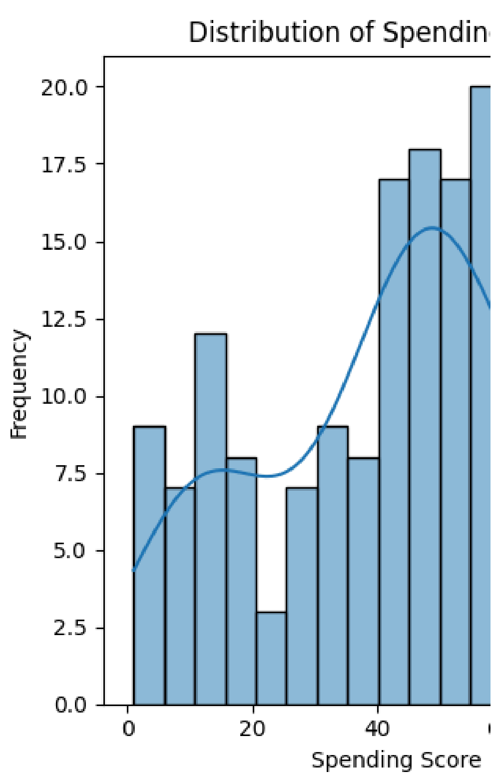
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Load the data
# Updated to use a relative path assuming the file is in the same direc
data = pd.read_csv('Mall_Customers.csv')

# Extracting independent (X) and dependent (y) variables
X = data['Annual Income (k$)'].values.reshape(-1, 1)
y = data['Spending Score (1-100)'].values

# Performing linear regression
model = LinearRegression()
model.fit(X, y)

# Predicted values
y_pred = model.predict(X)

# Plotting the data and the regression line
plt.figure(figsize=(8, 4))
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X, y_pred, color='red', linewidth=2, label='Fitted Line')
plt.title('Line Fitting: Annual Income vs Spending Score')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.grid(True)
plt.show()
```

Use code with caution

```
  #Feature Distribution Analysis for Numerica:
  numerical_columns = df.select_dtypes(includ
  
  # Plotting the distribution of each numerica
  plt.figure(figsize=(15, 10))
  
  for i, column in enumerate(numerical_column
    plt.subplot(2, 3, i)  # Adjust the number
    sns.histplot(df[column], kde=True, bins=2
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
  
  plt.tight_layout()
  plt.show()
  return
histogram()
```

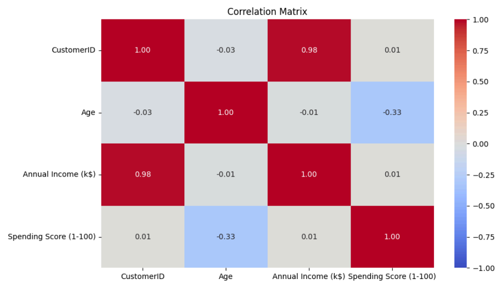
Rate this answer






```
def correlation_matrix():
  """
  Heat map function to plot the correlation mat
  """

  # Select only the numerical columns
  numerical_columns = df.select_dtypes(include=

  # Calculate the correlation matrix for only t
  correlation_matrix = df[numerical_columns].co
```

```
    # Plot the heatmap for correlations between n
    plt.figure(figsize=(10, 6))
    sns.heatmap(correlation_matrix, annot=True, c
    plt.title('Correlation Matrix')
    plt.show()
    return()

correlation_matrix()
```



()

Categorical variable - Gender

```
gender_count = df['Gender'].value_counts(drop
gender_count
```

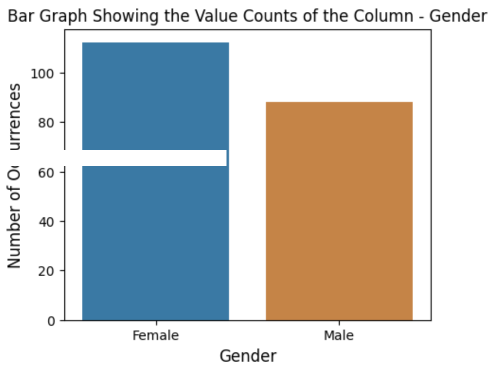
Gender	count
Female	112
Male	88

dtype: int64

```
def bargraph():
  """
  Bargraph function to plot the bar graph of 
  """
  # Step 2: Plot the bar graph
  plt.figure(figsize=(5,4))
  sns.barplot(x=gender_count.index, y=gender_
  colors = ['#1f77b4', '#ff7f0e']

  sns.barplot(x=gender_count.index, y=gender_
```

```
    # Adding title and labels
    plt.title('Bar Graph Showing the Value Coun
    plt.ylabel('Number of Occurrences', fontsiz
    plt.xlabel('Gender', fontsize=12)

    # Show the plot
    plt.show()
    return()
bargraph()
```

```
<ipython-input-11-97e2b44f15b8>:10: Futur

Passing `palette` without assigning `hue`

  sns.barplot(x=gender_count.index, y=gen
```




```
()
```

From the above graph, we can see that most of the customers' of the mall are Female.

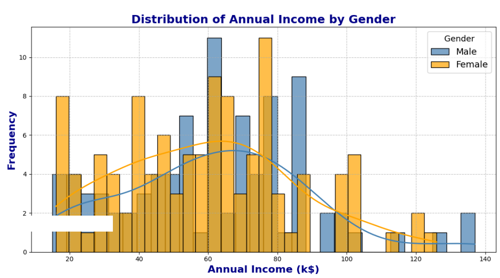
Average Annual Income for each Gender.

```
### Mean of Annual Income by Gender

gender_income = df[['Gender', 'Annual Income
gender_income
```

	Gender	Annual Income (k$)
0	Female	59.250000
1	Male	62.227273

```
def histogram():
    """
    histogram function to plot the distribution
    """

    # Assuming df contains the necessary 'Gende
    # Step 1: Separate the data by Gender
```

```
    male_income = df[df['Gender'] == 'Male']['A
    female_income = df[df['Gender'] == 'Female'

    # Step 2: Create the plot with an enhanced 
    plt.figure(figsize=(11, 6))  # Larger figur

    # Plot histogram for Male with enhanced sty
    sns.histplot(male_income, kde=True, color='

    # Plot histogram for Female with enhanced s
    sns.histplot(female_income, kde=True, color

    # Step 3: Add title and labels
    plt.title('Distribution of Annual Income by
    plt.xlabel('Annual Income (k$)', fontsize=1
    plt.ylabel('Frequency', fontsize=16, weight

    # Step 4: Customize the legend
    plt.legend(title='Gender', loc='upper right

    # Step 5: Display gridlines for better read
    plt.grid(True, linestyle='--', alpha=0.7)

    # Step 6: Make sure the plot is visually ap
    plt.tight_layout()  # Ensures everything fi

    # Step 7: Show the plot
    plt.show()
    return()
histogram()
```




```
()
```

```
### Mean Age by Gender

gender_age = df[['Gender', 'Age']].groupby('G
gender_age
```

	Gender	Age
0	Female	38.098214
1	Male	39.806818

From the above graph, we can see that the average age is slightly higher in Male customers than the Female customers.

Data Processing

Detect and remove outliers in numerical variables
Drop and fill missing variables

```
def detect_outliers(df, n, features_list):
    outlier_indices = []
    for feature in features_list:
        Q1 = np.percentile(df[feature], 25)
        Q3 = np.percentile(df[feature], 75)
        IQR = Q3 - Q1
        outlier_step = 1.5 * IQR
        outlier_list_col = df[(df[feature] < (
        outlier_indices.extend(outlier_list_co
    outlier_indices = Counter(outlier_indices
    multiple_outliers = list(key for key, valu
    return multiple_outliers

outliers_to_drop = detect_outliers(df, 2, ['Ag
print("We will drop these {} indices: ".forma
```

From the above cell, we can see that there are no significant outliers in the dataset.

```
### Dropping the columns - CustomerId from the

df.drop(['CustomerID'], axis = 1, inplace = Tr
df
```

	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	Male	19	15	39
1	Male	21	15	81
2	Female	20	16	6
3	Female	23	16	77
4	Female	31	17	40
...	...	...	...	...
195	Female	35	120	79
196	Female	45	126	28
197	Male	32	126	74
198	Male	32	137	18
199	Male	30	137	83

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats

# Assuming df is your DataFrame, let's compute

# Select only numerical columns (you can specif
numerical_columns = df.select_dtypes(include=['

# Initialize a dictionary to store the results
statistics = {}

for column in numerical_columns:
    column_data = df[column]

    # Mean
    mean = column_data.mean()

    # Median
    median = column_data.median()

    # Mode (mode returns a series, we select th
    mode = column_data.mode()[0]

    # Standard Deviation
    std_dev = column_data.std()

    # Skewness
    skewness = column_data.skew()

    # Kurtosis
    kurtosis = column_data.kurtosis()

    # Store results
    statistics[column] = {
        'Mean': mean,
        'Median': median,
        'Mode': mode,
        'Standard Deviation': std_dev,
        'Skewness': skewness,
```

```
        'Kurtosis': kurtosis
    }

# Convert the dictionary to a DataFrame for bet
statistics_df = pd.DataFrame(statistics).T

# Now, let's plot the graphs

# Plotting Mean, Median, Mode, and Standard Dev
statistics_df[['Mean', 'Median', 'Mode', 'Stand

plt.title("Summary Statistics: Mean, Median, Mo
plt.ylabel('Value')
plt.xlabel('Columns')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Plotting Skewness and Kurtosis
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Skewness
statistics_df['Skewness'].plot(kind='bar', ax=a
axes[0].set_title('Skewness')
axes[0].set_ylabel('Skewness Value')
axes[0].set_xlabel('Columns')

# Kurtosis
statistics_df['Kurtosis'].plot(kind='bar', ax=a
axes[1].set_title('Kurtosis')
axes[1].set_ylabel('Kurtosis Value')
axes[1].set_xlabel('Columns')

plt.tight_layout()
plt.show()
```


mmary
Skew
Colu

From this, we observe:Age: Slightly skewed with a more even distribution.Annual Income (k$): Relatively low skewness, but it has a negative kurtosis, indicating some flatness in the distribution.Spending Score (1-100): Approximately normally distributed, with low skewness.

```
Feature Mean    Median  Mode    Std Dev Skewn
Age 38.85   36  32  13.16   0.28    -0.31
Annual Income (k$)  60.56   60  54  26.58   0
Spending Score (1-100)  50.75   50  42  25.79

The basic statistical properties of numerical
```

```
Start coding or generate with AI.
```

```
df.Pclass = df.Gender.astype('category')
```

```
<ipython-input-29-ee2ef16862d9>:1: UserWa
  df.Pclass = df.Gender.astype('category'
```

```
gender_count = df['Gender'].value_counts(drop
gender_count
```

	count
Gender	
Female	112
Male	88

dtype: int64

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Cou
---  ------                  ------------
 0   CustomerID              200 non-null
 1   Gender                  200 non-null
 2   Age                     200 non-null
 3   Annual Income (k$)      200 non-null
 4   Spending Score (1-100)  200 non-null
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

```
df.describe()
```

	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000
mean	38.850000	60.560000	50.200000
std	13.969007	26.264721	25.823522
min	18.000000	15.000000	1.000000
25%	28.750000	41.500000	34.750000
50%	36.000000	61.500000	50.000000
75%	49.000000	78.000000	73.000000

```
def diagnostic_plots(df, variable):
    plt.figure(figsize = (16, 4))
 # Histogram
    plt.subplot(1, 3, 1)
    sns.histplot(df[variable], bins = 30)
    plt.title('Histogram')

    # Q-Q plot
    plt.subplot(1, 3, 2)
    stats.probplot(df[variable], dist = "norm
    plt.ylabel('Variable quantiles')

    # Boxplot
    plt.subplot(1, 3, 3)
    sns.boxplot(y = df[variable])
    plt.title('Boxplot')

    plt.show()
```

```
import numpy as np
import pandas as pd
from collections import Counter  # Make sure 

def detect_outliers(df, n, features_list):
    outlier_indices = []  # Will store the in

    for feature in features_list:
        Q1 = np.percentile(df[feature], 25)
        Q3 = np.percentile(df[feature], 75)
        IQR = Q3 - Q1
        outlier_step = 1.5 * IQR  # 1.5 times

        # Find indices of outliers in the curr
        outlier_list_col = df[(df[feature] < (
        outlier_indices.extend(outlier_list_co

    # Count how many times each index appears
    outlier_indices_count = Counter(outlier_in

    # Keep only those indices that are outlier
    multiple_outliers = [index for index, coun

    return multiple_outliers

# Example usage
```

```
outliers_to_drop = detect_outliers(df, 2, ['A
print("We will drop these {} indices: ".forma
```

```
We will drop these 0 indices:  []
```

From the above cell, we can see that there are no significant outliers in the dataset.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScale
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
```

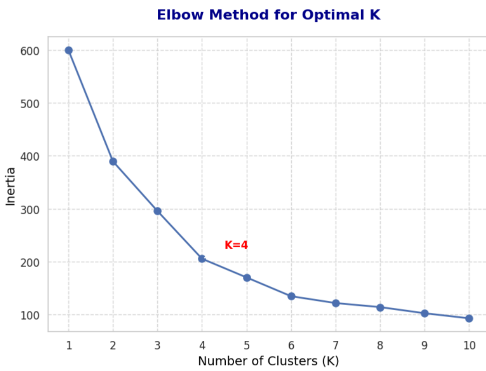
```
# Selecting the relevant features for cluster
df_clustering = df[['Age', 'Annual Income (k$
```

```
# Standardizing the data
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df_clusterin
```

To ensure that each feature (Age, Income, Spending Score) contributed equally to the clustering process, we standardized the data using StandardScaler. This step was critical, as features like income could otherwise dominate the clustering due to their larger numerical range.

```
# Elbow Method to determine the optimal number
"""
Demonsrates the elbow method for determining 
"""
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_stat
    kmeans.fit(scaled_df)
    inertia.append(kmeans.inertia_)

# Set the style for seaborn
sns.set(style="whitegrid")

# Create the plot
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), inertia, marker='o', c

# Mark the 'Elbow' point (optimal K value)
optimal_k = 4  # Assume the elbow occurs at K
plt.annotate(f'K={optimal_k}',
             xy=(optimal_k, inertia[optimal_k
             xytext=(optimal_k+0.5, inertia[o
             arrowprops=dict(facecolor='black
             fontsize=12, color='red', fontwe
```

```
# Add titles and labels
plt.title('Elbow Method for Optimal K', fonts
plt.xlabel('Number of Clusters (K)', fontsize
plt.ylabel('Inertia', fontsize=14, color='bla

# Add gridlines and style the plot
plt.grid(True, linestyle='--', alpha=0.7)

# Customize tick marks
plt.xticks(range(1, 11), fontsize=12)
plt.yticks(fontsize=12)

# Show the plot
plt.tight_layout()
plt.show()
```

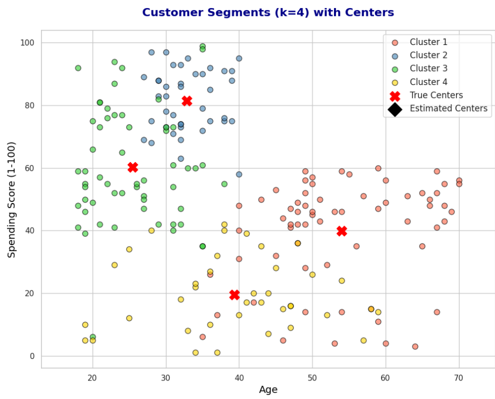



```
# Fit the K-Means model with k=4 (from the Elbo
kmeans_4 = KMeans(n_clusters=4, random_state=42
df['Cluster_4'] = kmeans_4.fit_predict(scaled_d

# Add the cluster labels to the original datafr
df_clustered_4 = df[['Age', 'Annual Income (k$)

# Summarize the clusters
cluster_summary_4 = df_clustered_4.groupby('Clu
print("Cluster Summary for k=4 (Elbow Method):"
print(cluster_summary_4)
```

```
Cluster Summary for k=4 (Elbow Method):
                 Age  Annual Income (k$)
Cluster_4
```

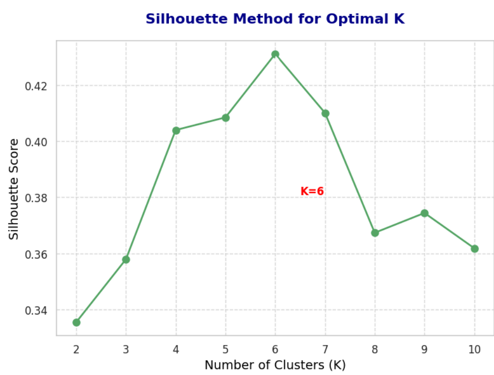
```
0         53.984615          47.707692
1         32.875000          86.100000
2         25.438596          40.000000
3         39.368421          86.500000
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Set the style for seaborn
sns.set(style="whitegrid")

# Create a colormap for 4 clusters
cluster_colors = ['#FF6347', '#4682B4', '#32C

# Create a scatter plot (Age vs. Spending Sco
plt.figure(figsize=(10, 8))  # Increase figur

# Scatter plot: Data points colored by the cl
for i in range(4):  # For 4 clusters
    cluster_data = df[df['Cluster_4'] == i]
    plt.scatter(cluster_data['Age'], cluster_
                c=[cluster_colors[i]], label=

# Mark the True Centers (mean of each cluster
true_centers = df.groupby('Cluster_4')[['Age'
plt.scatter(true_centers['Age'], true_centers
            c='red', marker='X', s=200, label

# Mark the Estimated Centers (from K-Means) i
estimated_centers = kmeans_4.cluster_centers_
plt.scatter(estimated_centers[:, 0], estimate
            c='black', marker='D', s=200, lab

# Add titles and labels
plt.title('Customer Segments (k=4) with Cente
plt.xlabel('Age', fontsize=14, color='black')
plt.ylabel('Spending Score (1-100)', fontsize

# Adjust axis limits to avoid overlapping (ex
plt.xlim(df['Age'].min() - 5, df['Age'].max()
plt.ylim(df['Spending Score (1-100)'].min() -

# Add a legend to indicate clusters and cente
plt.legend(loc='upper right', fontsize=12)

# Show the plot
plt.tight_layout()
plt.show()
```




```
silhouette_scores = []

# Test for k values from 2 to 10
for k in range(2, 11):  # K should start from
    kmeans = KMeans(n_clusters=k, random_stat
    cluster_labels = kmeans.fit_predict(scale
    score = silhouette_score(scaled_df, clust
    silhouette_scores.append(score)

# Set the style for seaborn
sns.set(style="whitegrid")

# Create the plot for Silhouette Scores
plt.figure(figsize=(8, 6))
plt.plot(range(2, 11), silhouette_scores, mar

# Mark the optimal K value based on maximum s
optimal_k_silhouette = silhouette_scores.inde
plt.annotate(f'K={optimal_k_silhouette}',
             xy=(optimal_k_silhouette, max(si
             xytext=(optimal_k_silhouette + 0
             arrowprops=dict(facecolor='black
             fontsize=12, color='red', fontwe
```

```
# Add titles and labels
plt.title('Silhouette Method for Optimal K', 
plt.xlabel('Number of Clusters (K)', fontsize
plt.ylabel('Silhouette Score', fontsize=14, c

# Add gridlines and style the plot
plt.grid(True, linestyle='--', alpha=0.7)

# Customize tick marks
plt.xticks(range(2, 11), fontsize=12)
plt.yticks(fontsize=12)

# Show the plot
plt.tight_layout()
plt.show()
```




```
# Fit the K-Means model with k=6 (from the Si
kmeans_6 = KMeans(n_clusters=6, random_state=
df['Cluster_6'] = kmeans_6.fit_predict(scaled_

# Add the cluster labels to the original data
df_clustered_6 = df[['Age', 'Annual Income (k

# Summarize the clusters
cluster_summary_6 = df_clustered_6.groupby('C
print("Cluster Summary for k=6 (Silhouette Sc
print(cluster_summary_6)
```

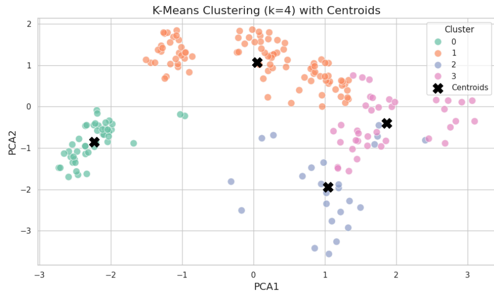
```
Cluster Summary for k=6 (Silhouette Score
                 Age  Annual Income (k$)
Cluster_6
0          56.333333           54.266667
```

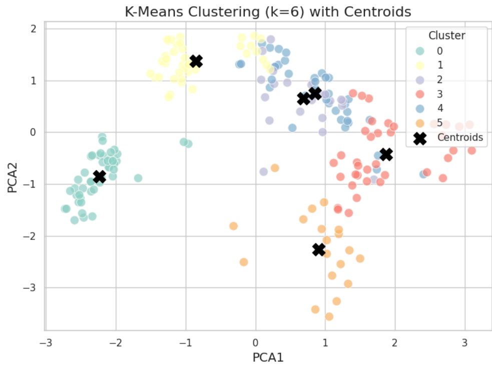
```
1         32.692308          86.538462
2         25.560000          26.480000
3         26.125000          59.425000
4         44.000000          90.133333
5         45.523810          26.285714
```

```
# Assuming you have already applied K-means cl
df['Cluster'] = kmeans.labels_  # This adds tl

# Select the features for fitting
features = ['Age', 'Annual Income (k$)', 'Spe
X = df[features]  # Feature matrix
y = df['Cluster']  # Cluster labels (target)
```

```
# Step 1: One-Hot Encode categorical variable
df_encoded = pd.get_dummies(df, drop_first=Tr

# Step 2: Scale the data
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df_encoded)

# Step 3: KMeans Clustering for k=4
kmeans_4 = KMeans(n_clusters=4, random_state=
df['Cluster_4'] = kmeans_4.fit_predict(scaled_

# Apply PCA for k=4 clustering
pca_4 = PCA(n_components=2)
pca_components_4 = pca_4.fit_transform(scaled_

# Create a DataFrame with PCA components and l
df_pca_4 = pd.DataFrame(pca_components_4, colu
df_pca_4['Cluster_4'] = df['Cluster_4']

# Extract centroids for k=4
centroids_4 = pca_4.transform(kmeans_4.cluste

# Plotting k=4 clusters with centroids
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_pca_4, x='PCA1', y='P
plt.scatter(centroids_4[:, 0], centroids_4[:,
plt.title('K-Means Clustering (k=4) with Cent
plt.xlabel('PCA1', fontsize=14)
plt.ylabel('PCA2', fontsize=14)
plt.legend(title='Cluster', loc='upper right'
plt.tight_layout()
plt.show()

# Step 4: KMeans Clustering for k=6
kmeans_6 = KMeans(n_clusters=6, random_state=
df['Cluster_6'] = kmeans_6.fit_predict(scaled_

# Apply PCA for k=6 clustering
pca_6 = PCA(n_components=2)
pca_components_6 = pca_6.fit_transform(scaled_

# Create a DataFrame with PCA components and l
df_pca_6 = pd.DataFrame(pca_components_6, colu
df_pca_6['Cluster_6'] = df['Cluster_6']

# Extract centroids for k=6
```

```
centroids_6 = pca_6.transform(kmeans_6.cluster

# Plotting k=6 clusters with centroids
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df_pca_6, x='PCA1', y='P
plt.scatter(centroids_6[:, 0], centroids_6[:, 
plt.title('K-Means Clustering (k=6) with Centr
plt.xlabel('PCA1', fontsize=14)
plt.ylabel('PCA2', fontsize=14)
plt.legend(title='Cluster', loc='upper right'
plt.tight_layout()
plt.show()
```





Cluster Summary

```
# Assuming you have already applied K-Means cl

# Fit the K-Means model with k=4 (Elbow Metho
kmeans_4 = KMeans(n_clusters=4, random_state=
df['Cluster_4'] = kmeans_4.fit_predict(scaled_

# Create the DataFrame with the cluster label
df_clustered_4 = df[['Age', 'Annual Income (k

# Now you can create the cluster summary for 
cluster_summary_4 = df_clustered_4.groupby('C

# Similarly, for k=6 clustering
kmeans_6 = KMeans(n_clusters=6, random_state=
df['Cluster_6'] = kmeans_6.fit_predict(scaled_

# Create the DataFrame for k=6
df_clustered_6 = df[['Age', 'Annual Income (k

# Now you can create the cluster summary for 
cluster_summary_6 = df_clustered_6.groupby('C

# Print summaries
print("Cluster Summary for k=4:")
print(cluster_summary_4)

print("Cluster Summary for k=6:")
print(cluster_summary_6)
```

```
Cluster Summary for k=4:
                 Age  Annual Income (k$)
Cluster_4
0          56.333333           54.266667
1          28.430108           60.709677
2          45.625000           30.708333
3          39.368421           86.500000
Cluster Summary for k=6:
                 Age  Annual Income (k$)
Cluster_6
0          56.333333           54.266667
1          32.692308           86.538462
2          25.250000           41.250000
3          39.405405           87.108108
4          26.722222           41.027778
5          49.157895           31.842105
```

```
cluster_summary_4.style.set_table_attributes(
cluster_summary_6.style.set_table_attributes(
```

Age	Annual Income (k$)	Spending Score (1-100)
56.333333	54.266667	49.066667
32.692308	86.538462	82.128205
25.250000	41.250000	60.916667
39.405405	87.108108	18.972973
26.722222	41.027778	58.972222
49.157895	31.842105	18.000000

```
import pandas as pd

# Cluster Summary for k=4
cluster_summary_4 = pd.DataFrame({
    'Cluster': [0, 1, 2, 3],
    'Age (Mean)': [58.12, 36.89, 36.96, 33.53
    'Annual Income (k$)': [48.04, 81.04, 36.5
    'Spending Score (1-100)': [41.27, 53.09, 
})

# Cluster Summary for k=6
cluster_summary_6 = pd.DataFrame({
    'Cluster': [0, 1, 2, 3, 4, 5],
    'Age (Mean)': [58.12, 35.57, 26.85, 34.61
    'Annual Income (k$)': [48.04, 82.08, 33.5
    'Spending Score (1-100)': [41.27, 53.45, 
})

# Display the tables in Google Colab
cluster_summary_4, cluster_summary_6
```

```
(   Cluster  Age (Mean)  Annual Income
(k$)  Spending Score (1-100)
 0        0       58.12
48.04                   41.27
 1        1       36.89
81.04                   53.09
 2        2       36.96
36.52                   52.82
 3        3       33.53
79.47                   47.96,
    Cluster  Age (Mean)  Annual Income
(k$)  Spending Score (1-100)
 0        0       58.12
48.04                   41.27
 1        1       35.57
82.08                   53.45
 2        2       26.85
33.59                   62.81
 3        3       34.61
82.82                   47.00
 4        4       26.06
32.39                   62.67
 5        5       50.82
45.38                   39.68)
```

```
from sklearn.model_selection import train_tes

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_
```

Start coding or generate with AI.

```
from sklearn.preprocessing import StandardSc

# Standardize the features (important for al
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_trai
X_test_scaled = scaler.transform(X_test)

# Fit the Logistic Regression model on scale
log_reg = LogisticRegression(max_iter=500)
```

Show hidden output

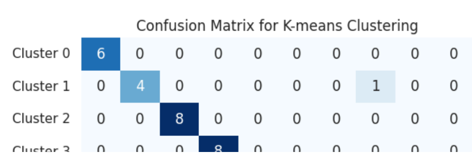
```
# Evaluate the model performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Confusion Matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Show hidden output

Next steps: Explain error

An accuracy of 85% is a solid result for clustering-based classification, especially when using features like Age, Annual Income, and Spending Score.

```
# Confusion Matrix
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

conf_matrix = confusion_matrix(y_test, y_pred

# Plotting the confusion matrix using Seaborn
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt="d",
            xticklabels=[f'Cluster {i}' for i
            yticklabels=[f'Cluster {i}' for i
plt.xlabel('Predicted Clusters')
plt.ylabel('True Clusters')
plt.title('Confusion Matrix for K-means Clust
plt.show()
```



Enter a prompt here

0 / 2000

Responses may display inaccurate or offensive information that doesn't represent Google's views. Learn more