```
#import dependencies
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
#load the dataset
df=pd.read_csv('/content/Mall_Customers.csv')
```

```
#check the first 5 rows
df.head()
```

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
#check the information
df.info()
```

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

```
# Drop CustomerID for analysis
features = df.drop(columns=['CustomerID'])
```

```
# Convert categorical 'Gender' to numeric
features = pd.get_dummies(features, drop_first=True)
```

```
from sklearn.preprocessing import StandardScaler
```

```
# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
```

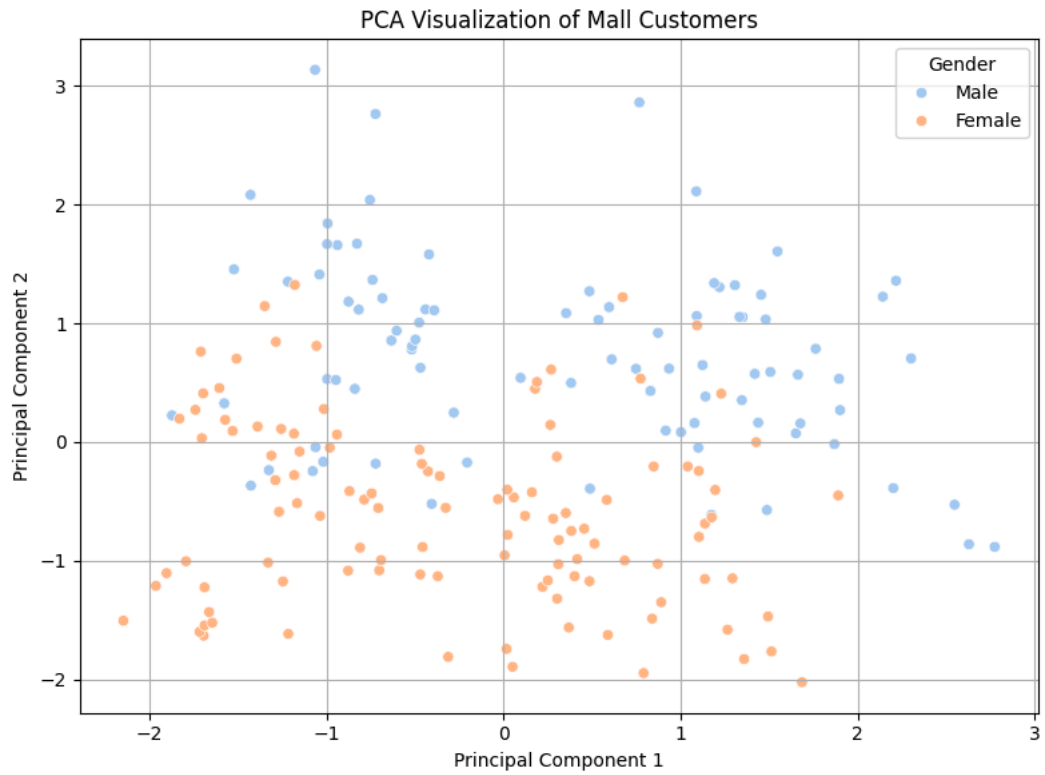
1.Load and visualize dataset (optional PCA for 2D view).

```
from sklearn.decomposition import PCA
```

```
# Apply PCA to reduce to 2 components for visualization
pca = PCA(n_components=2)
pca_result = pca.fit_transform(scaled_features)
```

```
# Create a DataFrame for PCA results
pca_df = pd.DataFrame(data=pca_result, columns=['PC1', 'PC2'])
```

```
# Plot the PCA results
plt.figure(figsize=(8,6))
sns.scatterplot(x='PC1', y='PC2', data=pca_df, hue=df['Gender'], palette='pastel')
plt.title('PCA Visualization of Mall Customers')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Gender')
plt.grid(True)
plt.tight_layout()
plt.show()
```



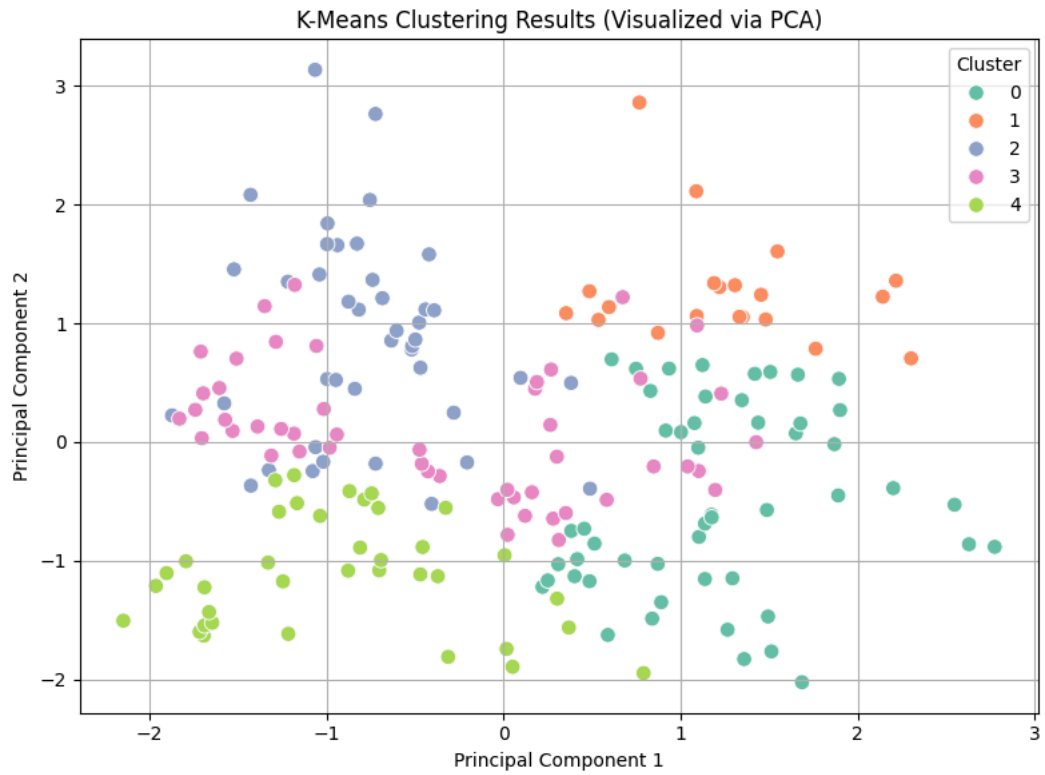
2.Fit K-Means and assign cluster labels

```
from sklearn.cluster import KMeans
```

```
# Apply KMeans with an initial assumption of 5 clusters
kmeans = KMeans(n_clusters=5, random_state=42)
cluster_labels = kmeans.fit_predict(scaled_features)
```

```
# Add cluster labels to the PCA dataframe
pca_df['Cluster'] = cluster_labels
```

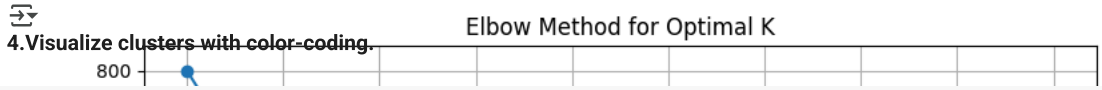
```
# Visualize the clusters in PCA-reduced space
plt.figure(figsize=(8, 6))
sns.scatterplot(x='PC1', y='PC2', data=pca_df, hue='Cluster', palette='Set2', s=70)
plt.title('K-Means Clustering Results (Visualized via PCA)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Cluster')
plt.grid(True)
plt.tight_layout()
plt.show()
```



3.Use the Elbow Method to find optimal K

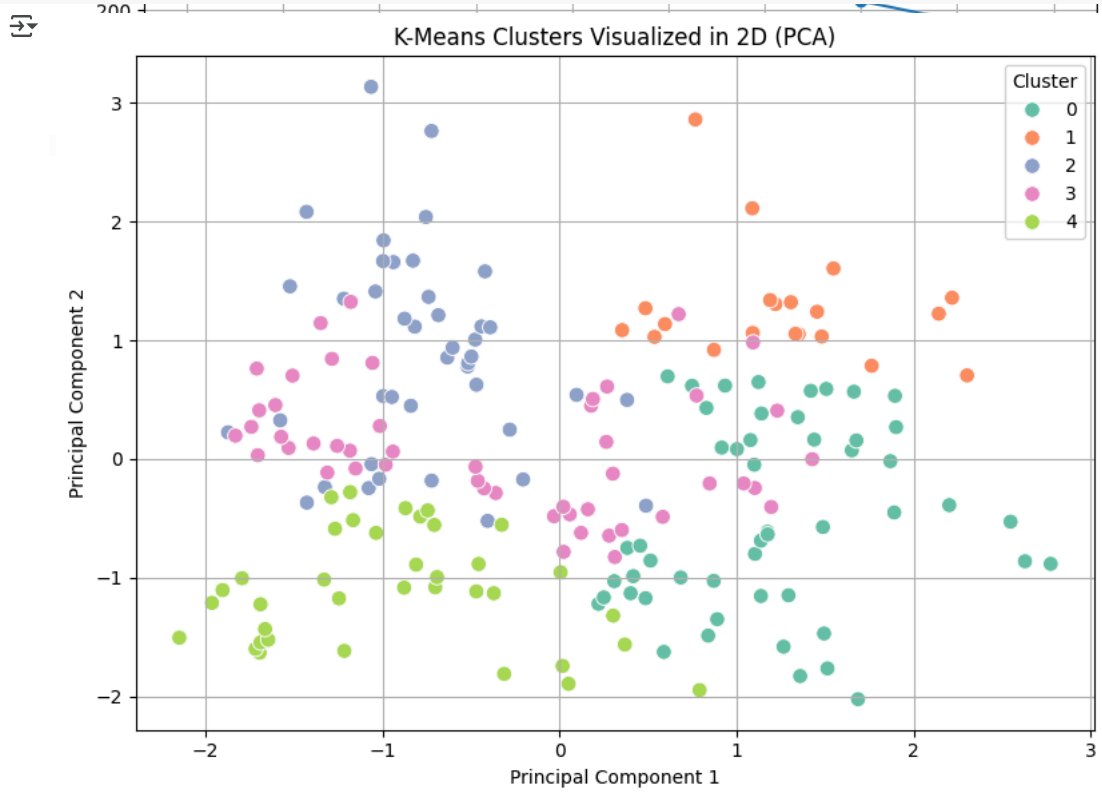
```
# Use the Elbow Method to find optimal number of clusters
inertia = []
K_range = range(1, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_features)
    inertia.append(kmeans.inertia_)
```

```
# Plotting the elbow curve
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, marker='o')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia (Within-Cluster Sum of Squares)')
plt.xticks(K_range)
plt.grid(True)
plt.tight_layout()
plt.show()
```



4.Visualize clusters with color-coding.

```
# Visualize the clusters using color coding
plt.figure(figsize=(8, 6))
sns.scatterplot(
    x='PC1', y='PC2',
    data=pca_df,
    hue='Cluster',
    palette='Set2',
    s=70
)
plt.title('K-Means Clusters Visualized in 2D (PCA)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Cluster')
plt.grid(True)
plt.tight_layout()
plt.show()
```



5.Evaluate clustering using Silhouette Score

```
from sklearn.metrics import silhouette_score
```

```
# Calculate silhouette score
score = silhouette_score(scaled_features, cluster_labels)
```