**✅ Cell 1**

python

CopyEdit

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

* **Problem Step**: Import all required Python libraries.
* **Definition**:
  + pandas as pd: For data manipulation and DataFrame creation.
  + numpy as np: For numerical operations and arrays.
  + matplotlib.pyplot as plt: For plotting basic charts and graphs.
  + seaborn as sns: For advanced statistical visualizations.
* **Why**: These libraries are essential to load data, perform calculations, and visualize results in an analysis pipeline.

**✅ Cell 2**

python

CopyEdit

data = pd.read\_csv('Mall\_Customers.csv')

data.head()

* **Problem Step**: Load dataset and display first few entries.
* **Definition**:
  + pd.read\_csv(): Reads data from a CSV file into a DataFrame.
  + data.head(): Shows the first 5 rows of the data.
* **Why**: To get an initial look at the structure, columns, and sample values in the dataset.

**✅ Cell 3**

python

CopyEdit

data.shape

* **Problem Step**: Get dataset dimensions.
* **Definition**:
  + .shape: Returns a tuple (rows, columns) representing the size of the DataFrame.
* **Why**: Helps to understand the scale of the dataset for further processing.

**✅ Cell 4**

python

CopyEdit

data.info()

* **Problem Step**: Explore data types and non-null values.
* **Definition**:
  + .info(): Displays column names, data types, and null value count.
* **Why**: To check data types and determine if any column needs conversion or contains missing data.

**✅ Cell 5**

python

CopyEdit

data.describe()

* **Problem Step**: Get statistical summary of numeric columns.
* **Definition**:
  + .describe(): Gives count, mean, std deviation, min, max, and quartiles.
* **Why**: Useful to understand distribution, outliers, and central tendency of numeric features.

**✅ Cell 6**

python

CopyEdit

data.isnull().sum()

* **Problem Step**: Check for missing data.
* **Definition**:
  + .isnull() creates a boolean mask where True = missing value.
  + .sum() totals missing values per column.
* **Why**: Identifies which columns may require imputation or removal.

**✅ Cell 7**

python

CopyEdit

data.drop(['CustomerID'], axis=1, inplace=True)

* **Problem Step**: Remove irrelevant column.
* **Definition**:
  + .drop(): Removes specified column.
  + axis=1: Indicates column-wise deletion.
  + inplace=True: Applies the change directly without creating a new copy.
* **Why**: CustomerID is just an identifier and not useful for clustering.

**✅ Cell 8**

python

CopyEdit

sns.pairplot(data)

plt.show()

* **Problem Step**: Visualize pairwise relationships between features.
* **Definition**:
  + sns.pairplot(): Plots scatterplots and histograms for all feature pairs.
* **Why**: Good for detecting patterns, correlations, or clusters in the data.

**✅ Cell 9**

python

CopyEdit

sns.heatmap(data.corr(), annot=True)

plt.show()

* **Problem Step**: Visualize feature correlation.
* **Definition**:
  + .corr(): Calculates Pearson correlation between numeric columns.
  + sns.heatmap(): Plots a color-coded correlation matrix.
  + annot=True: Displays numeric correlation values in the plot.
* **Why**: Helps identify which variables are linearly related.

**✅ Cell 10**

python

CopyEdit

X = data.iloc[:, [2, 3]].values

* **Problem Step**: Select features for clustering.
* **Definition**:
  + .iloc[:, [2, 3]]: Selects columns at positions 2 and 3 (Annual Income and Spending Score).
  + .values: Converts the selection to a NumPy array.
* **Why**: These two features are most relevant for customer segmentation.

**✅ Cell 11**

python

CopyEdit

from sklearn.cluster import KMeans

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state=42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

* **Problem Step**: Use the Elbow Method to find the optimal number of clusters.
* **Definition**:
  + KMeans(): Scikit-learn class for K-means clustering.
  + init='k-means++': Smart initialization to speed up convergence.
  + inertia\_: Sum of squared distances to the nearest cluster center (WCSS).
* **Why**: Helps identify the best k by checking where adding more clusters doesn’t significantly reduce WCSS.

**✅ Cell 12**

python

CopyEdit

plt.plot(range(1, 11), wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

* **Problem Step**: Plot the Elbow Curve.
* **Definition**:
  + plt.plot(): Plots WCSS vs. cluster count.
* **Why**: Visual tool to determine optimal number of clusters based on the 'elbow' point.

**✅ Cell 13**

python

CopyEdit

kmeans = KMeans(n\_clusters=5, init='k-means++', random\_state=42)

y\_kmeans = kmeans.fit\_predict(X)

* **Problem Step**: Fit the KMeans model with 5 clusters.
* **Definition**:
  + fit\_predict(): Fits KMeans and assigns each sample to a cluster.
* **Why**: Segments the customers into 5 distinct groups based on income and spending.

**✅ Cell 14**

python

CopyEdit

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s=100, c='red', label='Cluster 1')

plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s=100, c='blue', label='Cluster 2')

plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s=100, c='green', label='Cluster 3')

plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s=100, c='cyan', label='Cluster 4')

plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s=100, c='magenta', label='Cluster 5')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=300, c='yellow', label='Centroids')

plt.title('Clusters of customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()

* **Problem Step**: Visualize customer clusters and centroids.
* **Definition**:
  + plt.scatter(): Plots points for each cluster and centroid.
  + kmeans.cluster\_centers\_: Returns coordinates of cluster centers.
* **Why**: Shows the grouping of customers visually, which is helpful in business decision-making like targeted marketing.