**✅ 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.

**🔍 1. ROC Curve and AUC Score**

To **evaluate classifier performance**, especially in binary or multiclass settings, we use the **ROC curve** and the **AUC**.

**✅ Code Example**

python

CopyEdit

from sklearn.metrics import roc\_curve, auc

from sklearn.preprocessing import label\_binarize

# Binarize the output for multi-class (One-vs-Rest)

y\_test\_bin = label\_binarize(y\_test, classes=['setosa', 'versicolor', 'virginica'])

y\_pred\_prob = model.predict\_proba(X\_test)

# Plot ROC curve for each class

for i in range(3):

fpr, tpr, \_ = roc\_curve(y\_test\_bin[:, i], y\_pred\_prob[:, i])

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f'Class {i} (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--') # Random guess line

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve - Multiclass')

plt.legend()

plt.grid()

plt.show()

**📘 Explanation:**

* **Problem Step**: Evaluate model performance beyond accuracy.
* **Definition**:
  + roc\_curve(): Calculates true positive rate and false positive rate at different thresholds.
  + auc(): Calculates area under the ROC curve.
  + label\_binarize(): Converts categorical labels into binary vectors (needed for multiclass ROC).
* **Why**: ROC-AUC helps understand the trade-off between sensitivity and specificity across thresholds, and AUC quantifies this.

**🔍 2. Logistic (Sigmoid) Curve**

To **understand logistic regression behavior**, we visualize the **sigmoid function**, which maps input values to probabilities between 0 and 1.

**✅ Code Example**

python

CopyEdit

import numpy as np

def sigmoid(z):

return 1 / (1 + np.exp(-z))

z = np.linspace(-10, 10, 200)

sig = sigmoid(z)

plt.plot(z, sig)

plt.title("Logistic (Sigmoid) Curve")

plt.xlabel("z (Linear Combination of Inputs)")

plt.ylabel("Sigmoid(z)")

plt.grid()

plt.show()

**📘 Explanation:**

* **Problem Step**: Understand how logistic regression predicts probabilities.
* **Definition**:
  + sigmoid(z): Computes the logistic function, which squashes input to (0,1).
* **Why**: Logistic regression uses this function to model probability of class membership, making it interpretable and bounded.

**Summary of Concepts**

| **Metric/Tool** | **Purpose** | **Usefulness in Classification** |
| --- | --- | --- |
| ROC Curve | Plots TPR vs FPR at thresholds | Shows performance visually |
| AUC Score | Area under ROC | Quantifies overall performance |
| Logistic Curve | Models probability from inputs | Interprets classifier outputs |

**1. Logistic Regression**

Logistic Regression is a statistical method used for binary classification, where the outcome variable is categorical and typically takes the form of a binary response (e.g., success/failure, 1/0). It models the relationship between one or more independent variables (features) and a binary dependent variable by using a logistic function.

**The Logistic Function (Sigmoid Function)**

The core idea behind logistic regression is that it predicts the probability of an event occurring, which is between 0 and 1. This is achieved using the **sigmoid function**, defined as:

P(y=1∣X)=11+e−zP(y = 1 | X) = \frac{1}{1 + e^{-z}}P(y=1∣X)=1+e−z1​

Where:

* P(y=1∣X)P(y = 1 | X)P(y=1∣X) is the probability of the positive class (1),
* XXX represents the input features,
* z=w0+w1X1+w2X2+...+wnXnz = w\_0 + w\_1X\_1 + w\_2X\_2 + ... + w\_nX\_nz=w0​+w1​X1​+w2​X2​+...+wn​Xn​ is the linear combination of input features,
* eee is the base of the natural logarithm, and
* w0,w1,...,wnw\_0, w\_1, ..., w\_nw0​,w1​,...,wn​ are the coefficients that need to be learned.

**Decision Boundary**

Logistic regression maps the inputs through a sigmoid function to produce a value between 0 and 1. A threshold of 0.5 is commonly used to classify:

* If the output P(y=1∣X)≥0.5P(y = 1 | X) \geq 0.5P(y=1∣X)≥0.5, classify as 1 (positive class),
* Otherwise, classify as 0 (negative class).

**2. Log-Loss and ROC Curve**

**Log-Loss**

Log-loss evaluates the performance of classification models where the output is a probability value between 0 and 1. For binary classification, it's calculated as:

Log-Loss=−1n∑i=1n[yilog⁡(pi)+(1−yi)log⁡(1−pi)]\text{Log-Loss} = - \frac{1}{n} \sum\_{i=1}^{n} \left[ y\_i \log(p\_i) + (1 - y\_i) \log(1 - p\_i) \right]Log-Loss=−n1​i=1∑n​[yi​log(pi​)+(1−yi​)log(1−pi​)]

Where:

* yiy\_iyi​ is the true class label (0 or 1),
* pip\_ipi​ is the predicted probability of the positive class (1).

A lower log-loss value indicates better model performance.

**Receiver Operating Characteristic (ROC) Curve**

The ROC curve is a graphical representation of the diagnostic ability of a binary classifier as its discrimination threshold is varied. It plots:

* **True Positive Rate (TPR)**, also known as recall or sensitivity: TPR=TPTP+FNTPR = \frac{TP}{TP + FN}TPR=TP+FNTP​,
* **False Positive Rate (FPR)**: FPR=FPFP+TNFPR = \frac{FP}{FP + TN}FPR=FP+TNFP​.

Where:

* TPTPTP is the number of true positives,
* TNTNTN is the number of true negatives,
* FPFPFP is the number of false positives,
* FNFNFN is the number of false negatives.

The **Area Under the Curve (AUC)** is often used as a summary metric. A perfect model would have an AUC of 1, while a random classifier would have an AUC of 0.5.

**Interpretation of ROC Curve**

* The **x-axis** represents the False Positive Rate (FPR),
* The **y-axis** represents the True Positive Rate (TPR).

A classifier with a ROC curve closer to the top-left corner of the plot is considered better, as it has a higher TPR and a lower FPR.

**3. Learning Curves (LC)**

A **learning curve** is a plot that shows the performance of a model over time or with increasing training data. It typically plots:

* **Training error** vs. the number of training examples or iterations,
* **Validation error** vs. the number of training examples or iterations.

Learning curves are useful for understanding the behavior of a model and diagnosing common issues:

* **High bias (underfitting)**: The training error and validation error are both high, indicating the model is too simple.
* **High variance (overfitting)**: The training error is low, but the validation error is high, indicating the model is too complex and overfits the data.
* **Good performance**: Both training and validation errors decrease and converge to low values.

**Types of Learning Curves**

1. **Training Learning Curve**: Tracks how the model's error decreases on the training set as the number of training examples increases.
2. **Validation Learning Curve**: Tracks how the model's error changes on the validation set as the number of training examples increases.
3. **Test Learning Curve**: (less commonly plotted) Tracks how the model's error changes on the test set over time.

**Summary**

* **Logistic Regression** is a classification algorithm used for predicting probabilities, with the output modeled by a sigmoid function.
* **Log-Loss** is a cost function used to evaluate classification models.
* The **ROC Curve** helps to visualize the performance of a classifier, with AUC serving as a key evaluation metric.
* **Learning Curves** help diagnose issues in the model such as underfitting or overfitting by showing how error changes with more data or training iterations.