**-: BASIC :-**

**Step 1: Strengthen Your Python Basics**

Fundamental topics:

1. **Data Types and Variables**

Numbers, Strings, Booleans, Lists, Tuples, Sets, Dictionaries.

1. **Control Flow**

if-else, loops (for, while), comprehensions.

1. **Functions**

Defining functions, arguments, return values.

1. **File Handling**

Reading/writing files, working with CSV and text files.

1. **Error Handling**

Try-except blocks.

**Step 2: Python for Data Analysis**

1. **Learn Libraries for Data Analysis**
   * **NumPy**: For numerical operations and handling arrays.
   * **Pandas**: For data manipulation and analysis.
   * **Matplotlib and Seaborn**: For data visualization.
2. **Topics to Master**
   * Data Cleaning: Handling missing values, duplicates.
   * Data Transformation: Filtering, grouping, merging datasets.
   * Exploratory Data Analysis (EDA): Visualizing data patterns and relationships.
3. **Practice**
   * Work on datasets from **Kaggle** or **UCI Machine Learning Repository**.
   * Perform tasks like summarizing data, finding trends, and visualizing distributions.

**Step 3: Introduction to Machine Learning**

1. **Understand the Basics**

**What is machine learning?**

Machine learning (ML) is a subset of artificial intelligence (AI) that enables computers to learn from and make predictions or decisions based on data without being explicitly programmed.

In other words, ML algorithms analyze data, recognize patterns, and make decisions with minimal human intervention. It’s about building models that can learn from experience (data) and improve over time.

**Example**:

If you have data about house prices (features like square footage, number of bedrooms, etc.) and their actual sale prices, a machine learning model can "learn" the relationship between the features and the price, and then predict the price of a new house based on its features.

**Supervised vs. Unsupervised Learning.**

**Supervised Learning**

In **supervised learning**, the model is trained on labeled data. This means that the algorithm is given a set of input-output pairs. The goal is for the model to learn the relationship between the inputs (features) and outputs (labels/targets) so it can predict the output for new, unseen data.

* **Input:** Labeled data (features + target labels).
* **Output:** The model learns to predict the target label for new data.

**Examples of Supervised Learning:**

1. **Classification:** Predicting a category or class.  
   Example: Classifying emails as spam or not spam based on features like the subject, sender, etc.
2. **Regression:** Predicting a continuous value.  
   Example: Predicting the price of a house based on features like size, location, number of rooms, etc.

**Example:**

Using **Linear Regression** to predict house prices based on various features like square footage, number of bedrooms, and location.

from sklearn.linear\_model import LinearRegression

# Features (X) and target (y)

X = df[['square\_feet', 'num\_bedrooms', 'location']]

y = df['price']

# Train the model

model = LinearRegression()

model.fit(X, y)

# Predict house prices

predictions = model.predict(X\_test)

**Unsupervised Learning**

In **unsupervised learning**, the model is trained on data that doesn't have labeled outputs. The goal is to find hidden patterns or structures in the data. There is no specific target variable to predict; instead, the algorithm looks for similarities, differences, or groupings in the data.

* **Input:** Unlabeled data.
* **Output:** The model discovers patterns, such as clusters or relationships.

**Examples of Unsupervised Learning:**

1. **Clustering:** Grouping similar data points together.  
   Example: Grouping customers into segments based on their purchasing behavior.
2. **Dimensionality Reduction:** Reducing the number of features while preserving the essential information.  
   Example: Reducing the number of variables in a large dataset while retaining most of the original information.

**Example:**

Using **K-Means Clustering** to group customers based on their buying patterns.

from sklearn.cluster import KMeans

# Features (X)

X = df[['annual\_income', 'spending\_score']]

# Apply KMeans clustering

kmeans = KMeans(n\_clusters=3) # Grouping into 3 clusters

kmeans.fit(X)

# Get cluster labels

df['Cluster'] = kmeans.labels\_

**Common algorithms: Linear Regression, Decision Trees, Clustering.**

**a. Linear Regression**

Linear Regression is a **supervised learning** algorithm used for predicting a continuous target variable based on one or more input features. It assumes a linear relationship between the input and output.

**How it works:**

The algorithm finds the line (or hyperplane, in higher dimensions) that best fits the data points by minimizing the difference between the actual data points and the predictions made by the model.

**Use Case:**

Predicting house prices based on features like square footage and number of bedrooms.

from sklearn.linear\_model import LinearRegression

# Features (X) and target (y)

X = df[['square\_feet', 'num\_bedrooms', 'location']]

y = df['price']

# Train the model

model = LinearRegression()

model.fit(X, y)

# Predict house prices

predictions = model.predict(X\_test)

**b. Decision Trees**

Decision Trees are **supervised learning** algorithms used for both **classification** and **regression** tasks. They model decisions based on a tree-like structure of conditions.

**How it works:**

* The data is split into subsets based on feature values.
* The decision tree chooses the feature that best splits the data at each node.
* The process continues recursively until it creates leaf nodes that represent predictions (for classification) or values (for regression).

**Use Case:**

Predicting if a customer will buy a product based on features like age, income, and browsing history.

from sklearn.tree import DecisionTreeClassifier

# Features (X) and target (y)

X = df[['age', 'income', 'browsing\_history']]

y = df['purchase']

# Train the decision tree model

model = DecisionTreeClassifier(random\_state=42)

model.fit(X, y)

# Predict purchase decisions

predictions = model.predict(X\_test)

**c. Clustering (K-Means)**

Clustering is an **unsupervised learning** algorithm used to group similar data points into clusters.

**How it works:**

The K-Means algorithm:

1. Selects k random points as the centroids (initial cluster centers).
2. Assigns each data point to the nearest centroid.
3. Recalculates the centroids as the average of the points assigned to each cluster.
4. Repeats the assignment and centroid recalculation until convergence (i.e., centroids no longer change).

**Use Case:**

Segmenting customers based on their buying behavior into different groups.

from sklearn.cluster import KMeans

# Features (X)

X = df[['annual\_income', 'spending\_score']]

# Apply KMeans clustering

kmeans = KMeans(n\_clusters=3) # Grouping into 3 clusters

kmeans.fit(X)

# Get cluster labels

df['Cluster'] = kmeans.labels\_

**d. Other Common Algorithms**

* **Logistic Regression:** For classification tasks (e.g., predicting whether a customer will buy a product or not).
* **Random Forest:** An ensemble method that uses multiple decision trees to improve prediction accuracy.
* **Support Vector Machines (SVM):** Used for classification tasks, particularly when the data is high-dimensional.
* **Neural Networks:** Powerful models inspired by the human brain, used for tasks like image recognition and natural language processing.

1. **Learn Libraries for ML**
   * + **Scikit-Learn**: For implementing ML models.
     + **TensorFlow/PyTorch** (later): For deep learning.
2. **Topics to Start**
   * + Data Preprocessing: Scaling, encoding, splitting datasets.
     + Building a Simple Model: Train, test, evaluate performance.
     + Model Evaluation: Metrics like accuracy, precision, recall, etc.

**Step 4: Advance to Deep Learning**

1. **Learn Deep Learning Concepts**
   * Neural Networks, activation functions.
   * Optimizers (e.g., Gradient Descent).
   * Loss functions (e.g., Mean Squared Error, Cross-Entropy).
2. **Deep Learning Frameworks**
   * Start with **Keras (TensorFlow)** for simplicity.
   * Use **PyTorch** for more flexibility and advanced use cases.

**Step 5: Projects and Practice**

1. **Real-World Projects**
   * Predictive modeling: Sales predictions, stock price forecasting.
   * Classification: Email spam detector, image classification.
   * Clustering: Customer segmentation, grouping similar products.
2. **Competitions**
   * Participate in challenges on Kaggle to improve problem-solving skills.

**-: MACHINE LEARNING :-**

**Relevant Python Topics for ML**

1. **Object-Oriented Programming (OOP)**  
   Helps organize code better in larger ML projects.
2. **Iterators and Generators**  
   Efficient handling of large datasets.
3. **Working with APIs**  
   For gathering data from online sources.
4. **Regular Expressions**  
   Useful for text preprocessing in Natural Language Processing (NLP).
5. **Multithreading and Multiprocessing**  
   Speeding up tasks during data preparation or model training.
6. **Variables and Data Types**

**Definition**

A variable is a name that refers to a value stored in memory. Python supports various data types, including integers, floats, strings, and booleans.

**Example**

# Integer

x = 10

print("Integer:", x)

# Float

y = 3.14

print("Float:", y)

# String

name = "Machine Learning"

print("String:", name)

# Boolean

is\_active = True

print("Boolean:", is\_active)

**2. Control Flow (if-else)**

**Definition**

Control flow determines the order in which statements are executed based on conditions.

**Example**

# Check if a number is positive, negative, or zero

number = -5

if number > 0:

print("The number is positive.")

elif number < 0:

print("The number is negative.")

else:

print("The number is zero.")

**3. Loops**

**Definition**

Loops are used to repeat a block of code multiple times. Python provides for and while loops.

**Example**

**For Loop:**

# Print all elements in a list

numbers = [1, 2, 3, 4, 5]

for num in numbers:

print(num)

**While Loop:**

# Print numbers from 1 to 5

count = 1

while count <= 5:

print(count)

count += 1

**4. Functions**

**Definition**

A function is a block of reusable code that performs a specific task. It is defined using the def keyword.

**Example**

# Define a function to add two numbers

def add\_numbers(a, b):

return a + b

# Call the function

result = add\_numbers(3, 7)

print("The sum is:", result)

**5. Working with Lists**

**Definition**

A list is an ordered collection of items. It can hold different data types.

**Example**

# Create a list

fruits = ["apple", "banana", "cherry"]

# Access elements

print("First fruit:", fruits[0])

# Add an element

fruits.append("orange")

print("Updated list:", fruits)

# Remove an element

fruits.remove("banana")

print("List after removal:", fruits)

**6. File Handling**

**Definition**

File handling is used to read from or write to files.

**Example**

# Write to a file

with open("example.txt", "w") as file:

file.write("Hello, Python!")

# Read from a file

with open("example.txt", "r") as file:

content = file.read()

print("File content:", content)

**7. Importing Libraries**

**Definition**

Libraries extend Python’s functionality. For example, math for mathematical operations or numpy for numerical computations.

**Example**

import math

# Calculate square root

result = math.sqrt(16)

print("Square root:", result)

**8. NumPy (Numerical Python)**

**Definition**

NumPy is a library used for numerical computations. It introduces the ndarray object, which is more efficient than Python lists for handling large datasets.

**Key Features**

* Arrays and array operations.
* Mathematical functions (e.g., mean, median, standard deviation).
* Random number generation.

**Example 1: Creating Arrays**

import numpy as np

# Create a 1D array

arr = np.array([1, 2, 3, 4, 5])

print("1D Array:", arr)

# Create a 2D array

matrix = np.array([[1, 2], [3, 4], [5, 6]])

print("2D Array:\n", matrix)

**Example 2: Array Operations**

# Perform element-wise operations

arr = np.array([1, 2, 3])

print("Array + 2:", arr + 2)

print("Array \* 3:", arr \* 3)

# Calculate statistics

print("Mean:", np.mean(arr))

print("Sum:", np.sum(arr))

**9. Pandas (Python Data Analysis Library)**

**Definition**

Pandas is used for data manipulation and analysis. It provides two primary structures:

* + **Series**: One-dimensional labeled arrays.
  + **DataFrame**: Two-dimensional labeled tables.

**Example 1: Series**

import pandas as pd

# Create a Series

data = pd.Series([10, 20, 30, 40], index=['A', 'B', 'C', 'D'])

print("Series:\n", data)

# Access elements

print("Value at index 'B':", data['B'])

**Example 2: DataFrames**

# Create a DataFrame

data = {

"Name": ["Alice", "Bob", "Charlie"],

"Age": [25, 30, 35],

"Salary": [50000, 60000, 70000]

}

df = pd.DataFrame(data)

# Display the DataFrame

print("DataFrame:\n", df)

# Access columns

print("Names:\n", df["Name"])

# Filter rows

print("People with Age > 28:\n", df[df["Age"] > 28])

**10. Data Visualization**

Visualization is key to understanding data patterns. Let’s use **Matplotlib** and **Seaborn**.

**Definition**

* + **Matplotlib**: A basic plotting library.
  + **Seaborn**: Built on Matplotlib, provides prettier plots with less code.

**Example: Matplotlib**

import matplotlib.pyplot as plt

# Create a simple plot

x = [1, 2, 3, 4, 5]

y = [10, 20, 25, 30, 35]

plt.plot(x, y, label="Line")

plt.title("Simple Plot")

plt.xlabel("X-axis")

plt.ylabel("Y-axis")

plt.legend()

plt.show()

**Example: Seaborn**

import seaborn as sns

# Create a histogram

data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]

sns.histplot(data, bins=5, kde=True)

plt.title("Histogram")

plt.show()

**11. Real-World Data Analysis Example**

Let’s analyze a dataset using Pandas and visualize it.

**Dataset: Titanic (Survival Information)**

# Load the dataset

df = pd.read\_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")

# View the first few rows

print("First 5 rows:\n", df.head())

# Analyze missing values

print("Missing Values:\n", df.isnull().sum())

# Fill missing values in 'Age' with the median

df['Age'].fillna(df['Age'].median(), inplace=True)

# Analyze survival based on gender

survival\_rate = df.groupby('Sex')['Survived'].mean()

print("Survival Rate by Gender:\n", survival\_rate)

# Visualize survival rates

survival\_rate.plot(kind='bar', color=['blue', 'pink'])

plt.title("Survival Rate by Gender")

plt.ylabel("Survival Rate")

plt.show()

**12. Data Preprocessing**

**Definition**

Data preprocessing involves preparing raw data into a format suitable for analysis. It includes:

* + **Handling Missing Data**
  + **Encoding Categorical Data**
  + **Feature Scaling**
  + **Splitting Data into Training and Testing Sets**

**Step A: Handling Missing Data**

When data contains missing values, they can cause errors in calculations and reduce model performance.

**Example: Filling Missing Values**

import pandas as pd

# Create a DataFrame with missing values

data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],

'Age': [25, None, 30, 35],

'Salary': [50000, 60000, None, 70000]}

df = pd.DataFrame(data)

print("Original DataFrame:\n", df)

# Fill missing values in 'Age' with the mean

df['Age'].fillna(df['Age'].mean(), inplace=True)

# Fill missing values in 'Salary' with a fixed value

df['Salary'].fillna(0, inplace=True)

print("\nDataFrame after handling missing values:\n", df)

**Step B: Encoding Categorical Data**

Machine learning models work with numerical data, so we need to convert categorical (text) data into numerical form.

**Example: Encoding**

from sklearn.preprocessing import LabelEncoder

# Create a DataFrame with categorical data

data = {'Country': ['USA', 'France', 'Germany', 'France'],

'Purchased': ['Yes', 'No', 'Yes', 'No']}

df = pd.DataFrame(data)

print("Original DataFrame:\n", df)

# Encode the 'Country' column

label\_encoder = LabelEncoder()

df['Country'] = label\_encoder.fit\_transform(df['Country'])

# Encode the 'Purchased' column

df['Purchased'] = label\_encoder.fit\_transform(df['Purchased'])

print("\nDataFrame after encoding:\n", df)

**Step C: Feature Scaling**

Feature scaling ensures that all features contribute equally to the model. Methods include **Normalization** (scaling values between 0 and 1) and **Standardization** (scaling values with a mean of 0 and standard deviation of 1).

**Example: Scaling**

from sklearn.preprocessing import StandardScaler

# Create a dataset

data = {'Age': [25, 30, 35, 40],

'Salary': [50000, 60000, 70000, 80000]}

df = pd.DataFrame(data)

print("Original DataFrame:\n", df)

# Apply standard scaling

scaler = StandardScaler()

scaled\_data = scaler.fit\_transform(df)

# Convert scaled data back to a DataFrame

scaled\_df = pd.DataFrame(scaled\_data, columns=['Age', 'Salary'])

print("\nScaled DataFrame:\n", scaled\_df)

**Step D: Splitting Data into Training and Testing Sets**

Splitting the dataset ensures that the model is trained on one part of the data and tested on unseen data.

**Example: Splitting**

from sklearn.model\_selection import train\_test\_split

# Create a dataset

data = {'Feature1': [1, 2, 3, 4, 5],

'Feature2': [10, 20, 30, 40, 50],

'Target': [0, 1, 0, 1, 0]}

df = pd.DataFrame(data)

# Split data into features (X) and target (y)

X = df[['Feature1', 'Feature2']]

y = df['Target']

# Split into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

print("Training Features:\n", X\_train)

print("\nTesting Features:\n", X\_test)

print("\nTraining Target:\n", y\_train)

print("\nTesting Target:\n", y\_test)

**13. Building Your First Machine Learning Model**

**Model: Linear Regression**

Linear regression is used for predicting continuous values (e.g., house prices).

**Steps:**

* + Prepare data.
  + Train the model using a training dataset.
  + Test the model and evaluate its performance.

**Example: Predicting House Prices**

from sklearn.linear\_model import LinearRegression

import numpy as np

# Dataset: Square footage vs. Price

X = np.array([[500], [1000], [1500], [2000], [2500]]) # Features

y = np.array([150000, 300000, 450000, 600000, 750000]) # Target

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Predict on the test set

predictions = model.predict(X\_test)

print("Predicted Prices:", predictions)

print("Actual Prices:", y\_test)

**14. Evaluating Model Performance**

**Definition**

Model evaluation helps measure how well a machine learning model performs on unseen data. Common metrics vary based on the type of problem:

* 1. **Regression Problems** (predicting continuous values):
     + Mean Absolute Error (MAE)
     + Mean Squared Error (MSE)
     + Root Mean Squared Error (RMSE)
  2. **Classification Problems** (predicting categories):
     + Accuracy
     + Precision
     + Recall
     + F1-Score

**Example: Evaluating a Linear Regression Model**

Let’s calculate **MSE** and **RMSE** for a regression model.

from sklearn.metrics import mean\_squared\_error

import numpy as np

# Actual prices and predicted prices

y\_test = np.array([300000, 450000, 600000]) # Actual

predictions = np.array([310000, 460000, 590000]) # Predicted

# Calculate Mean Squared Error (MSE)

mse = mean\_squared\_error(y\_test, predictions)

print("Mean Squared Error:", mse)

# Calculate Root Mean Squared Error (RMSE)

rmse = np.sqrt(mse)

print("Root Mean Squared Error:", rmse)

**Example: Evaluating a Classification Model**

For classification, we use accuracy, precision, recall, and F1-score.

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

# Actual and predicted labels

y\_test = [0, 1, 0, 1, 1] # Actual

predictions = [0, 1, 0, 0, 1] # Predicted

# Calculate metrics

accuracy = accuracy\_score(y\_test, predictions)

precision = precision\_score(y\_test, predictions)

recall = recall\_score(y\_test, predictions)

f1 = f1\_score(y\_test, predictions)

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1-Score:", f1)

**15. Decision Trees**

**Definition**

A Decision Tree is a tree-structured model that splits data into branches based on decision rules. It’s intuitive and works for both regression and classification tasks.

**Steps to Build a Decision Tree Classifier**

* 1. Import the necessary libraries.
  2. Preprocess the data (if needed).
  3. Train the model.
  4. Make predictions.
  5. Evaluate the model.

**Example: Predicting if a Passenger Survived on Titanic**

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

import pandas as pd

# Load Titanic dataset

url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"

df = pd.read\_csv(url)

# Select relevant features

df = df[['Pclass', 'Age', 'Sex', 'Survived']]

# Handle missing values

df['Age'].fillna(df['Age'].median(), inplace=True)

# Encode 'Sex' column

df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})

# Split data into features (X) and target (y)

X = df[['Pclass', 'Age', 'Sex']]

y = df['Survived']

# Split into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the Decision Tree model

model = DecisionTreeClassifier()

model.fit(X\_train, y\_train)

# Predict on the test set

predictions = model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, predictions)

print("Model Accuracy:", accuracy)

**Visualizing the Decision Tree**

You can visualize the decision tree to better understand how it makes decisions.

from sklearn.tree import plot\_tree

import matplotlib.pyplot as plt

# Plot the decision tree

plt.figure(figsize=(10, 8))

plot\_tree(model, feature\_names=['Pclass', 'Age', 'Sex'], class\_names=['Died', 'Survived'], filled=True)

plt.show()

**16. K-Nearest Neighbors (KNN)**

**Definition**

K-Nearest Neighbors is a simple algorithm used for classification and regression. It predicts the class (or value) of a data point based on the majority class (or average value) of its nearest neighbors.

**Steps for KNN**

* + Choose the number of neighbors (**K**).
  + Calculate the distance of the test point from all training points (e.g., using Euclidean distance).
  + Find the **K closest neighbors**.
  + For classification: Predict the majority class. For regression: Predict the average of the values.

**Example: Classifying Iris Flower Types**

We’ll use the famous **Iris dataset**, which classifies flowers into three species based on features like petal length and width.

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris = load\_iris()

X = iris.data # Features: petal/sepal lengths and widths

y = iris.target # Target: flower species (0, 1, 2)

# Split the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the KNN model

knn = KNeighborsClassifier(n\_neighbors=3) # Use 3 neighbors

knn.fit(X\_train, y\_train)

# Predict on the test set

predictions = knn.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, predictions)

print("Model Accuracy:", accuracy)

**17. Confusion Matrix**

**Definition**

A **confusion matrix** is used to evaluate the performance of a classification model by showing the true positives, true negatives, false positives, and false negatives.

**Example: Confusion Matrix with KNN**

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

# Generate the confusion matrix

cm = confusion\_matrix(y\_test, predictions)

print("Confusion Matrix:\n", cm)

# Visualize the confusion matrix

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=iris.target\_names)

disp.plot(cmap='Blues')

plt.show()

**18. Random Forest**

**Definition**

Random Forest is an ensemble learning algorithm that builds multiple decision trees and combines their outputs (via averaging for regression or majority voting for classification) for more accurate and robust predictions.

**Steps for Random Forest**

* + Select random subsets of the training data.
  + Build a decision tree for each subset.
  + Aggregate the predictions of all trees.

**Example: Random Forest for Titanic Survival**

from sklearn.ensemble import RandomForestClassifier

# Load Titanic dataset

url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"

df = pd.read\_csv(url)

# Preprocess the data

df = df[['Pclass', 'Age', 'Sex', 'Survived']]

df['Age'].fillna(df['Age'].median(), inplace=True)

df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})

X = df[['Pclass', 'Age', 'Sex']] # Features

y = df['Survived'] # Target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train Random Forest model

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42) # 100 trees

rf\_model.fit(X\_train, y\_train)

# Predict and evaluate

predictions = rf\_model.predict(X\_test)

accuracy = accuracy\_score(y\_test, predictions)

print("Random Forest Model Accuracy:", accuracy)

**19. Hyperparameter Tuning**

Random Forest and other models perform better when hyperparameters (e.g., the number of trees or max tree depth) are tuned.

**Example: Using GridSearchCV for Random Forest**

from sklearn.model\_selection import GridSearchCV

# Define hyperparameter grid

param\_grid = {

'n\_estimators': [50, 100, 200],

'max\_depth': [None, 10, 20],

'min\_samples\_split': [2, 5],

}

# Perform grid search

grid\_search = GridSearchCV(RandomForestClassifier(random\_state=42), param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)

# Best parameters

print("Best Hyperparameters:", grid\_search.best\_params\_)

# Evaluate the best model

best\_model = grid\_search.best\_estimator\_

predictions = best\_model.predict(X\_test)

accuracy = accuracy\_score(y\_test, predictions)

print("Optimized Model Accuracy:", accuracy)

**-: SQL :-**

**1. Understanding Your Data**

To predict profitability, you need to analyze your sales and purchase data effectively. Here's what each key data column might represent:

**Key Columns (Typical Example):**

* 1. **Product\_ID:** Unique identifier for each product.
  2. **Purchase\_Price:** Cost price of the product.
  3. **Sale\_Price:** Selling price of the product.
  4. **Quantity\_Purchased:** Total purchased quantity.
  5. **Quantity\_Sold:** Total sold quantity.
  6. **Date:** Date of purchase or sale.
  7. **Stock\_Left:** Remaining stock quantity.

**By analyzing this data, you can calculate:**

* 1. **Profit:** Profit=Sale Price−Purchase PriceProfit=Sale Price−Purchase Price
  2. **Profit Margin:** Profit Margin=ProfitPurchase Price×100Profit Margin=Purchase PriceProfit​×100
  3. **Turnover Rate:** Turnover Rate=Quantity SoldTotal StockTurnover Rate=Total StockQuantity Sold​

**Deep Analysis Goals:**

* 1. Identify **high-profit products** based on margin and turnover rate.
  2. Detect **seasonality trends** (e.g., products sold more during holidays).
  3. Predict future sales trends based on historical patterns.

**2. Loading the Data from SQL**

First, let’s load your SQL data into a manageable format like **Pandas DataFrame**.

**SQL Data Loading**

If the SQL file contains table dumps, load it into an SQLite database or directly query a connected database.

**Example: Loading Data from SQL File**

import sqlite3

import pandas as pd

# Step 1: Create SQLite database connection

conn = sqlite3.connect('sales\_data.db')

# Step 2: Read SQL script into the database

with open('your\_file.sql', 'r') as file:

sql\_script = file.read()

conn.executescript(sql\_script)

# Step 3: Query data into a DataFrame

query = """

SELECT Product\_ID, Purchase\_Price, Sale\_Price, Quantity\_Purchased,

Quantity\_Sold, Date, Stock\_Left

FROM sales\_table;

"""

df = pd.read\_sql\_query(query, conn)

# Step 4: Explore the data

print(df.head())

**3. Data Cleaning**

Real-world data is messy. Cleaning ensures high-quality analysis.

**Key Cleaning Steps:**

* + **Handle Missing Values:**  
    Replace missing values in critical columns like Purchase\_Price or Quantity\_Sold with medians or mode.
  + **Remove Duplicates:**  
    Drop duplicate rows to avoid bias.
  + **Convert Data Types:**  
    Ensure numeric columns like Purchase\_Price and Quantity\_Sold are floats/integers.
  + **Parse Dates:**  
    Convert Date to a proper datetime format for time-based analysis.

**Example: Cleaning Missing Values**

# Fill missing values in numerical columns

df['Purchase\_Price'].fillna(df['Purchase\_Price'].median(), inplace=True)

df['Quantity\_Sold'].fillna(0, inplace=True)

# Convert Date to datetime

df['Date'] = pd.to\_datetime(df['Date'])

# Drop duplicate rows

df.drop\_duplicates(inplace=True)

print(df.info())

**4. Exploratory Data Analysis (EDA)**

Now that your data is clean, let’s analyze it.

**Profit and Profit Margin**

Profit and profit margin are critical metrics for deciding product profitability.

# Calculate Profit and Profit Margin

df['Profit'] = (df['Sale\_Price'] - df['Purchase\_Price']) \* df['Quantity\_Sold']

df['Profit\_Margin'] = (df['Profit'] / (df['Purchase\_Price'] \* df['Quantity\_Sold'])) \* 100

# Group by Product and Calculate Total Profit

profit\_by\_product = df.groupby('Product\_ID')['Profit'].sum().reset\_index()

top\_profitable = profit\_by\_product.sort\_values(by='Profit', ascending=False)

print(top\_profitable.head())

**Turnover Rate**

Turnover rate helps identify fast-moving products.

df['Turnover\_Rate'] = df['Quantity\_Sold'] / (df['Quantity\_Sold'] + df['Stock\_Left'])

# Average turnover rate by product

turnover\_by\_product = df.groupby('Product\_ID')['Turnover\_Rate'].mean().reset\_index()

print(turnover\_by\_product.sort\_values(by='Turnover\_Rate', ascending=False).head())

**Seasonality Analysis**

Seasonality identifies trends, like higher sales during specific months.

# Extract month and year for seasonality

df['Month'] = df['Date'].dt.month

df['Year'] = df['Date'].dt.year

# Group sales by Month

seasonal\_sales = df.groupby('Month')['Quantity\_Sold'].sum().reset\_index()

import matplotlib.pyplot as plt

plt.plot(seasonal\_sales['Month'], seasonal\_sales['Quantity\_Sold'])

plt.title('Seasonality of Sales')

plt.xlabel('Month')

plt.ylabel('Total Quantity Sold')

plt.show()

**5. Predictive Modeling**

Predict future trends and profitability using machine learning models.

**Feature Engineering**

Create features for prediction:

* + **Turnover\_Rate:** Quantity SoldTotal StockTotal StockQuantity Sold​
  + **Average Sales:** Average monthly sales.
  + **Profit Margin**
  + **Seasonality Index:** Using Month and Year.

**Train a Machine Learning Model**

We’ll use **Random Forest Regression** to predict profit.

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error

# Features and Target

X = df[['Turnover\_Rate', 'Profit\_Margin', 'Quantity\_Sold']]

y = df['Profit']

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train Random Forest

model = RandomForestRegressor(random\_state=42)

model.fit(X\_train, y\_train)

# Predict and Evaluate

predictions = model.predict(X\_test)

mae = mean\_absolute\_error(y\_test, predictions)

print("Mean Absolute Error:", mae)

**6. Generate Report**

Finally, summarize findings and suggest which products to stock.

**Example: Recommend Products**

# Predict profits for all products

df['Predicted\_Profit'] = model.predict(X)

# Recommend products with high predicted profit

recommendations = df[df['Predicted\_Profit'] > 1000] # Customize threshold

print(recommendations[['Product\_ID', 'Predicted\_Profit']].drop\_duplicates())