**Practical 1**

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**df = pd.read\_csv('C:/Users/lenovo/OneDrive/Desktop/titanic.csv')**

**df.info()**

// This line gives you an overview of the dataset.

**df.shape**

// This tells you how many rows and columns are in the dataset. It returns a tuple like (rows, columns).

**df.isnull()**

// This checks if there are any missing (null) values in the dataset. It returns a DataFrame of the same shape as df, where each cell is True if the value is missing and False if it’s not.

**df.isnull().sum()**

// This line counts how many missing values there are in each column. It helps you understand which columns need to be cleaned up or handled.

**df.describe()**

// This provides a summary of the numeric columns. It shows:

* Count (number of non-null values).
* Mean (average value).
* Standard deviation (how spread out the numbers are).
* Minimum, 25th percentile, 50th percentile (median), 75th percentile, and maximum values.

**df['PassengerId'].describe()**

// This gives the same statistical summary but only for the 'PassengerId' column.

**condition = (df['PassengerId'] == "80")**

**condition.head()**

// This line creates a condition to check if the 'PassengerId' column contains the value "80". This would help you filter out rows where the 'PassengerId' equals "80". However, it's important to note that "80" is a string, which might not be the correct type (PassengerId is likely an integer).

**df[df['PassengerId'] >= 80].head(10)**

// This filters and displays rows where the 'PassengerId' is greater than or equal to 80. The .head(10) part shows the first 10 rows that meet this condition.

**df['FamilySize'] = df['SibSp'] + df['Parch']**

// This line creates a new column FamilySize that tells us how many family members each passenger has on board by combining siblings/spouses and parents/children.

**df['PassengerId'] = df['PassengerId'].astype(int)**

// Convert 'PassengerId' to integer

**df['AgeGroup'] = pd.cut(df['Age'], bins=[0, 12, 18, 35, 60, 100], labels=['Child', 'Teenager', 'Young Adult', 'Adult', 'Senior'])**

// This groups the Age column into different categories (like Child, Teenager, etc.) based on age ranges.

**df.drop(['PassengerId', 'Ticket', 'Name'], axis=1, inplace=True)**

//This line removes the PassengerId, Ticket, and Name columns from the dataset because these features may not provide much value for predicting survival (in the Titanic dataset).

**df['Name'] = df['Name'].str.lower()** # Convert to lowercase

**df['Name'] = df['Name'].str.replace('[^a-zA-Z]', ' ')** # Remove special characters

// These lines convert all names to lowercase and remove any special characters, leaving only alphabetic characters.

**Practical 2**

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

# Step 1: Load the dataset

df = pd.read\_csv('C:/Users/lenovo/OneDrive/Desktop/Book1.csv')

# Step 2: Data Preprocessing (Splitting the dataset)

X = df[['YearsExperience']] # Independent variable (Experience)

y = df['Salary'] # Dependent variable (Salary)

# Split the data 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)

# Step 3: Train the Linear Regression model

model = LinearRegression()

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

# Step 4: Predict salaries using the test data

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

# Step 5: Visualize the results (optional)

plt.scatter(X\_train, y\_train, color='blue', label='Actual Salary (Train)')

plt.plot(X\_train, model.predict(X\_train), color='red', label='Predicted Salary (Model)')

plt.title('Salary vs. Years of Experience (Training set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.legend()

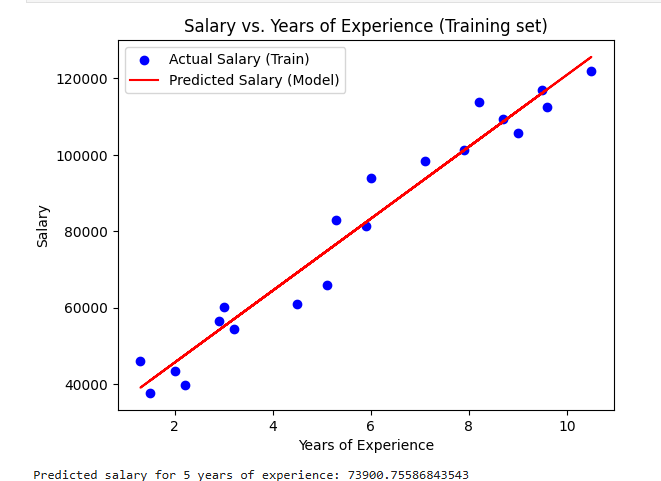
plt.show()

# Step 6: Predict salary for a new experience value (example)

years\_of\_experience = np.array([[5]]) # Predict salary for 5 years of experience

predicted\_salary = model.predict(years\_of\_experience)

print(f'Predicted salary for 5 years of experience: {predicted\_salary[0]}')



**Practical 3**

# Step 1: Importing necessary libraries

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

#train\_test\_split helps in splitting data into training and testing sets.

# LinearRegression is the algorithm used for predicting house prices.

# mean\_squared\_error measures how well the model performs by calculating errors between actual and predicted values.

# Step 2: Loading the dataset

# Assuming you have a CSV file called 'house\_prices.csv'

df = pd.read\_csv('C:/Users/lenovo/OneDrive/Desktop/house\_prices.csv')

# Step 3: Preprocessing - Here, we'll use only numerical data and remove rows with missing values

df = df.dropna()

# Features (X) and Target (y)

X = df[['SquareFeet', 'NumRooms', 'NumBathrooms', 'Age']] # Independent variables

y = df['Price'] # Dependent variable (target)

# Step 4: Split the data 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)

# Step 5: Initialize and train the model

model = LinearRegression()

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

# Step 6: Make predictions on the test set

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

# Step 7: Evaluate the model

# This calculates the Mean Squared Error (MSE), which is a measure of how close the predicted house prices are to the actual prices in the testing data (y\_test).

# A lower MSE means the model is better at making predictions.

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

print(f'Mean Squared Error: {mse}')

# To see the coefficients (how much each feature impacts the price)

# model.coef\_ shows how much each feature (square feet, number of rooms, bathrooms, and age) contributes to the house price.

# model.intercept\_ is a constant term that adjusts the model to make better predictions.

print(f'Coefficients: {model.coef\_}')

print(f'Intercept: {model.intercept\_}')

Output:

Mean Squared Error: 128125000000.00217

Coefficients: [ -1750. 187500. 187500. -100000.]

Intercept: 3412500.0000000373

**Practical 4**

# Step 1: Import necessary libraries

import pandas as pd

import numpy as np

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

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn.datasets import load\_iris

# Step 2: Load the IRIS dataset

# The dataset itself (iris.data) is loaded from the scikit-learn library. you need to do it manually

iris = load\_iris()

df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

df['species'] = iris.target

# Step 3: Features (X) and target (y)

X = df.drop(columns=['species']) # Independent variables (features)

y = df['species'] # Dependent variable (target)

# Step 4: Split the data 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)

# Step 5: Initialize and train the SVM classifier

svm\_model = SVC()

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

# Step 6: Make predictions on the test set

y\_pred = svm\_model.predict(X\_test)

# Step 7: Calculate accuracy score

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

print(f'Accuracy Score: {accuracy}')

# Step 8: Generate confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(f'Confusion Matrix:\n{conf\_matrix}')

**Output:**

Accuracy Score: 1.0

Confusion Matrix:

[[10 0 0]

[ 0 9 0]

[ 0 0 11]]

**Practical 5**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Load the dataset

df = pd.read\_csv('C:/Users/lenovo/OneDrive/Desktop/car\_data.csv')

# Preprocessing

# Convert 'Gender' to numerical values: Male=1, Female=0

df['Gender'] = df['Gender'].map({'Male': 1, 'Female': 0})

# Drop rows with missing values

df = df.dropna()

# Features (X) and Target (y)

X = df[['Gender', 'Age', 'AnnualSalary']] # Independent variables

y = df['Purchased'] # Dependent variable (target)

# Split the data 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)

# Initialize and train the Logistic Regression model

model = LogisticRegression()

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

# Make predictions on the test set

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

# Evaluate the model

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

print(f'Accuracy: {accuracy \* 100:.2f}%')

# Confusion Matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print('Confusion Matrix:')

print(conf\_matrix)

# Visualize the Confusion Matrix

plt.figure(figsize=(6, 4))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,

xticklabels=["Not Purchased", "Purchased"], yticklabels=["Not Purchased", "Purchased"])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

# Predict whether XYZ (given gender, age, and salary) will buy a car

# Example: XYZ's details

gender = 1 # Male

age = 30

AnnualSalary = 50000

# Create a DataFrame for XYZ

xyz\_df = pd.DataFrame({'Gender': [gender], 'Age': [age], 'AnnualSalary': [AnnualSalary]})

# Predict the purchase probability

purchase\_prob = model.predict\_proba(xyz\_df)[0][1] # Probability of purchasing

purchase\_prediction = model.predict(xyz\_df)[0] # Binary prediction (0 or 1)

print(f"Probability of XYZ buying a car: {purchase\_prob:.2f}")

print(f"Prediction for XYZ buying a car: {'Yes' if purchase\_prediction == 1 else 'No'}")

