**Create a model to predict the house price using Multiple Linear Regression.**

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the dataset

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

# Preprocessing: Convert categorical variables to numeric

data['mainroad'] = data['mainroad'].map({'yes': 1, 'no': 0})

data['guestroom'] = data['guestroom'].map({'yes': 1, 'no': 0})

data['basement'] = data['basement'].map({'yes': 1, 'no': 0})

data['hotwaterheating'] = data['hotwaterheating'].map({'yes': 1, 'no': 0})

data['airconditioning'] = data['airconditioning'].map({'yes': 1, 'no': 0})

data['prefarea'] = data['prefarea'].map({'yes': 1, 'no': 0})

# One-hot encoding for 'furnishingstatus'

data = pd.get\_dummies(data, columns=['furnishingstatus'], drop\_first=True)

# Define independent variables (X) and dependent variable (y)

X = data.drop('price', axis=1)  # All columns except 'price'

y = data['price']  # Target variable

# Split the data into training (80%) and testing (20%) 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)

# Make predictions

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

# Evaluate the model

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

r2 = r2\_score(y\_test, y\_pred)

# Output the evaluation metrics

print(f"Mean Squared Error (MSE): {mse}")

print(f"R² Score: {r2}")

# Optional: Display the model's coefficients

coefficients = pd.DataFrame(model.coef\_, X.columns, columns=['Coefficient'])

print(coefficients)

new\_data = pd.DataFrame({

    'area': [8000],  # Area in square feet

    'bedrooms': [4],  # Number of bedrooms

    'bathrooms': [3],  # Number of bathrooms

    'stories': [2],  # Number of stories

    'mainroad': [1],  # 1 for yes, 0 for no

    'guestroom': [0],  # 1 for yes, 0 for no

    'basement': [1],  # 1 for yes, 0 for no

    'hotwaterheating': [0],  # 1 for yes, 0 for no

    'airconditioning': [1],  # 1 for yes, 0 for no

    'parking': [2],  # Number of parking spaces

    'prefarea': [1],  # 1 for yes, 0 for no

    'furnishingstatus\_semi-furnished': [0],  # 1 for semi-furnished

    'furnishingstatus\_unfurnished': [0]  # 1 for unfurnished

})

# Predicting the price for new data

predicted\_price = model.predict(new\_data)

print(f"Predicted price for the new house: ${predicted\_price[0]:,.2f}")

**Display the accuracy score and confusion matrix of IRIS dataset using SVM classifier**

# Step 1: Import necessary libraries

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.svm import SVC

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

from sklearn.datasets import load\_iris

# Step 2: Load the IRIS dataset

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:.2f}')

# Step 8: Generate confusion matrix

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

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

# Step 9: Visualize the confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

            xticklabels=iris.target\_names, yticklabels=iris.target\_names)

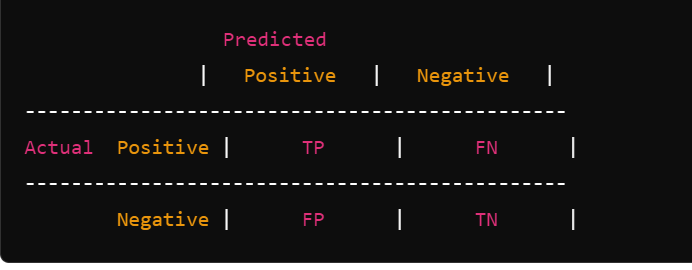
plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

Note: A confusion matrix is a useful tool for evaluating the performance of a classification model. It summarizes how well the model's predictions match the actual target values. Here’s a breakdown of its components:



Where:

* **TP (True Positives)**: The number of instances correctly predicted as positive.
* **TN (True Negatives)**: The number of instances correctly predicted as negative.
* **FP (False Positives)**: The number of instances incorrectly predicted as positive (Type I error).
* **FN (False Negatives)**: The number of instances incorrectly predicted as negative (Type II error).

**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()

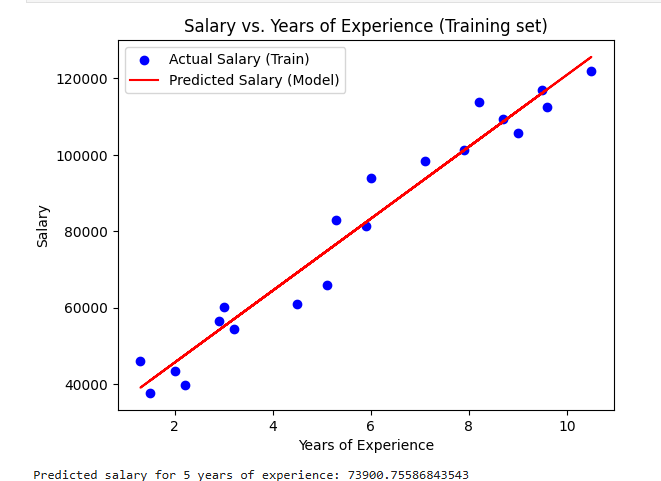
plt.show()

# 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 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]})

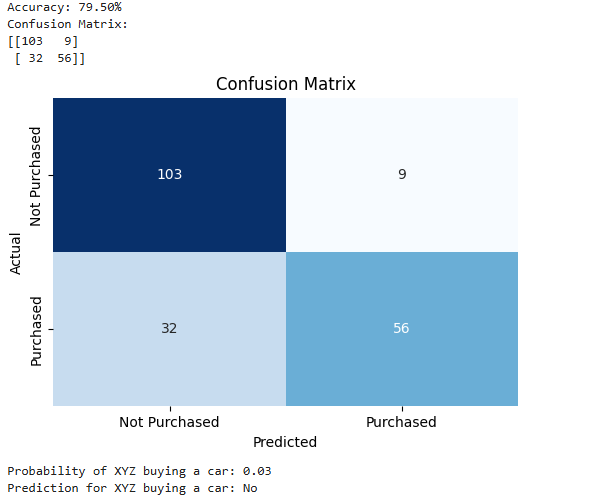
# Predict the purchase probability

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'}")



**Create a mini project on "Handwritten Digit Recognition" using TensorFIow, Keras, SciKit .**

# Import necessary libraries

import numpy as np # For numerical operations

import matplotlib.pyplot as plt # For plotting images

from tensorflow import keras # For building neural network models

from tensorflow.keras import layers # For building layers in the neural network

from sklearn.model\_selection import train\_test\_split # For splitting the dataset

# Load the MNIST dataset from Keras

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.mnist.load\_data()

# Normalize the data to scale pixel values to range [0, 1]

x\_train = x\_train.astype('float32') / 255 # Convert training images to float and scale

x\_test = x\_test.astype('float32') / 255 # Convert test images to float and scale

# Reshape the data to add a channel dimension (28x28 images with 1 color channel)

x\_train = x\_train.reshape((60000, 28, 28, 1)) # 60000 training images

x\_test = x\_test.reshape((10000, 28, 28, 1)) # 10000 test images

# Build the Convolutional Neural Network (CNN) model

model = keras.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)), # Convolutional layer with 32 filters

layers.MaxPooling2D((2, 2)), # Max pooling layer to reduce spatial dimensions

layers.Conv2D(64, (3, 3), activation='relu'), # Second convolutional layer with 64 filters

layers.MaxPooling2D((2, 2)), # Another max pooling layer

layers.Flatten(), # Flatten the output to feed into the dense layer

layers.Dense(64, activation='relu'), # Fully connected layer with 64 neurons

layers.Dense(10, activation='softmax') # Output layer with 10 neurons (for digits 0-9)

])

# Compile the model

model.compile(optimizer='adam', # Adam optimizer for training

loss='sparse\_categoric al\_crossentropy', # Loss function for multi-class classification

metrics=['accuracy']) # Metric to evaluate performance

# Train the model on the training data

model.fit(x\_train, y\_train, epochs=5, batch\_size=64, validation\_split=0.2) # Use 20% of the data for validation

# Evaluate the model on the test data

test\_loss, test\_acc = model.evaluate(x\_test, y\_test) # Get the loss and accuracy on test set

print(f'Test accuracy: {test\_acc}') # Print the test accuracy

# Make predictions on the test data

predictions = model.predict(x\_test) # Predict the classes of the test set

# Display the first test image and its prediction

plt.imshow(x\_test[0].reshape(28, 28), cmap='gray') # Show the first image in grayscale

plt.title(f'Predicted: {np.argmax(predictions[0])}, True: {y\_test[0]}') # Show the predicted and true labels

plt.show() # Display the plot

