**NAME OF THE COURSE : ARTIFICIAL INTELLIGENCE**

**NAME OF THE PROJECT : PREDICITING HOUSE PRICES USING MACHINE LEARNING**

**NAME OF THE STUDENT : R.Sujay**

**DEPARTMENT : BE-ECE**

**PHASE – 2**

Problem statement:

 The problem is to predict house prices using machine learning techniques. The objective is to develop a model that accurately predicts the prices of houses based on a set of features such as location, square footage, number of bedrooms and bathrooms, and other relevant factors. This project involves data preprocessing, feature engineering, model selection, training, and evaluation.

Introduction

In the realm of predictive modeling, XGBoost (Extreme Gradient Boosting) stands out as a powerful and efficient algorithm. It is particularly effective for regression tasks, where the goal is to predict continuous numerical values. In this document, we explore the utilization of XGBoost for regression in Google Colab, a collaborative platform for data science and machine learning.

XGBoost: A Brief Overview

XGBoost is a machine learning algorithm based on the gradient boosting framework, designed to optimize speed and performance. Its flexibility, scalability, and accuracy make it a preferred choice for various data science applications.

Implementation in Google Colab

* Step 1: Data Loading and Preprocessing

The first step involves loading the dataset into Google Colab. The data, typically in Excel format, is uploaded using Colab's file upload functionality. Once loaded, the dataset is split into features (input variables) and the target variable (the value to be predicted). In our example, the dataset is split into training and testing sets to evaluate the model's performance.

* Step 2: XGBoost Model Creation

The XGBoost model is created using the `XGBRegressor` class from the XGBoost library. This class allows for fine-tuning of various hyperparameters to optimize the model's performance. In our case, we specify the objective as 'reg:squarederror' for regression tasks and set the number of estimators to 100.

* Step 3: Model Training and Evaluation

The model is trained using the training dataset. Once trained, it is evaluated using the testing dataset. Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) are calculated to assess the model's accuracy. These metrics provide insights into how well the model performs in predicting numerical values.

* Step 4: Making Predictions

Once the model is trained and evaluated, it can make predictions on new data points. The XGBoost model can provide accurate predictions for continuous variables based on the patterns learned during training.

* Step 5: Visualization of Results

Visualizing the results is crucial to understanding the model's performance. In our example, a scatter plot is generated, comparing actual prices with predicted prices. This visualization provides a clear picture of how well the model predicts numerical values.

Source Code

* Model – 1 : Linear Regression;

import pandas as pd

from sklearn.linear\_model import LinearRegression

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

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

import numpy as np

import matplotlib.pyplot as plt

# Upload 'Own AI data.xlsx' in Google Colab before running this cell

# Load the dataset

dataset = pd.read\_excel('Own AI data.xlsx')

# Separate features and target variable

x = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

# Split the dataset into training and testing sets

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.25, random\_state=0)

# Create and train the Linear Regression model

model = LinearRegression()

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

# Make predictions on the test set

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

# Calculate Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE)

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

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

print("Mean Absolute Error (MAE):", mae)

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

# Make a prediction for a new data point

new\_data = [[78300, 6, 8, 3, 48050]]

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

print("Predicted Price for New Data:", predicted\_result)

# Plotting the predicted vs actual prices

plt.scatter(y\_test, y\_pred, color='blue')

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Actual Prices vs Predicted Prices")

plt.show()

* Model – 2 : RandomForest Regression;

import pandas as pd

from sklearn.ensemble import RandomForestRegressor

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

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

import numpy as np

import matplotlib.pyplot as plt

# Upload 'Own AI data.xlsx' in Google Colab before running this cell

# Load the dataset

dataset = pd.read\_excel('Own AI data.xlsx')

# Separate features and target variable

x = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

# Split the dataset into training and testing sets

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.25, random\_state=0)

# Create and train the Random Forest Regressor model

model = RandomForestRegressor(n\_estimators=100, random\_state=0)

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

# Make predictions on the test set

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

# Calculate Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE)

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

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

print("Mean Absolute Error (MAE):", mae)

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

# Make a prediction for a new data point

new\_data = [[78300, 6, 8, 3, 48050]]

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

print("Predicted Price for New Data:", predicted\_result)

# Plotting the predicted vs actual prices

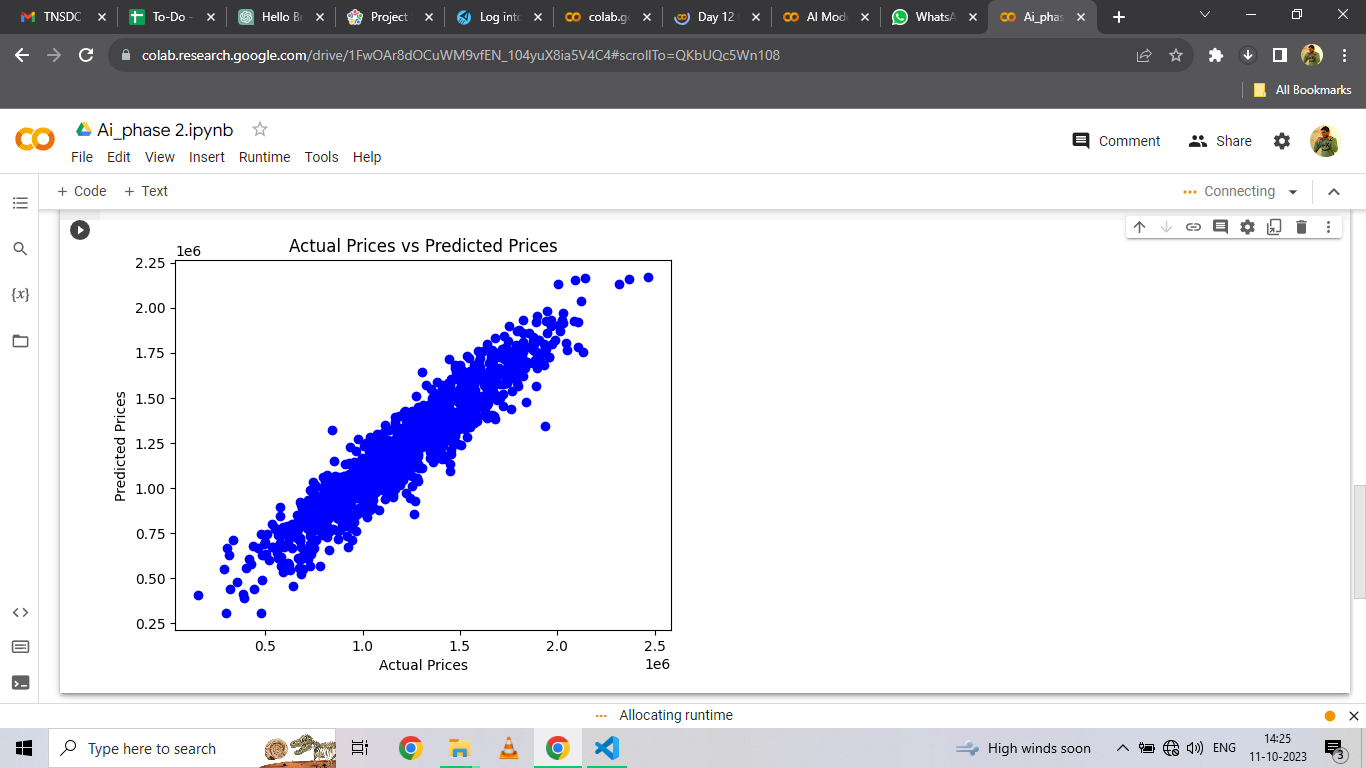
plt.scatter(y\_test, y\_pred, color='blue')

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Actual Prices vs Predicted Prices")

plt.show()



* Model – 3 : XG Bosster Regression;

import pandas as pd

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

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

import numpy as np

import xgboost as xgb

import matplotlib.pyplot as plt

# Upload 'Own AI data.xlsx' in Google Colab before running this cell

# Load the dataset

dataset = pd.read\_excel('Own AI data.xlsx')

# Separate features and target variable

x = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

# Split the dataset into training and testing sets

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.25, random\_state=0)

# Create and train the XGBoost Regressor model

model = xgb.XGBRegressor(objective ='reg:squarederror', n\_estimators=100, random\_state=0)

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

# Make predictions on the test set

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

# Calculate Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE)

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

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

print("Mean Absolute Error (MAE):", mae)

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

# Make a prediction for a new data point

new\_data = [[78300, 6, 8, 3, 48050]]

predicted\_result = model.predict(np.array(new\_data))

print("Predicted Price for New Data:", predicted\_result)

# Plotting the predicted vs actual prices

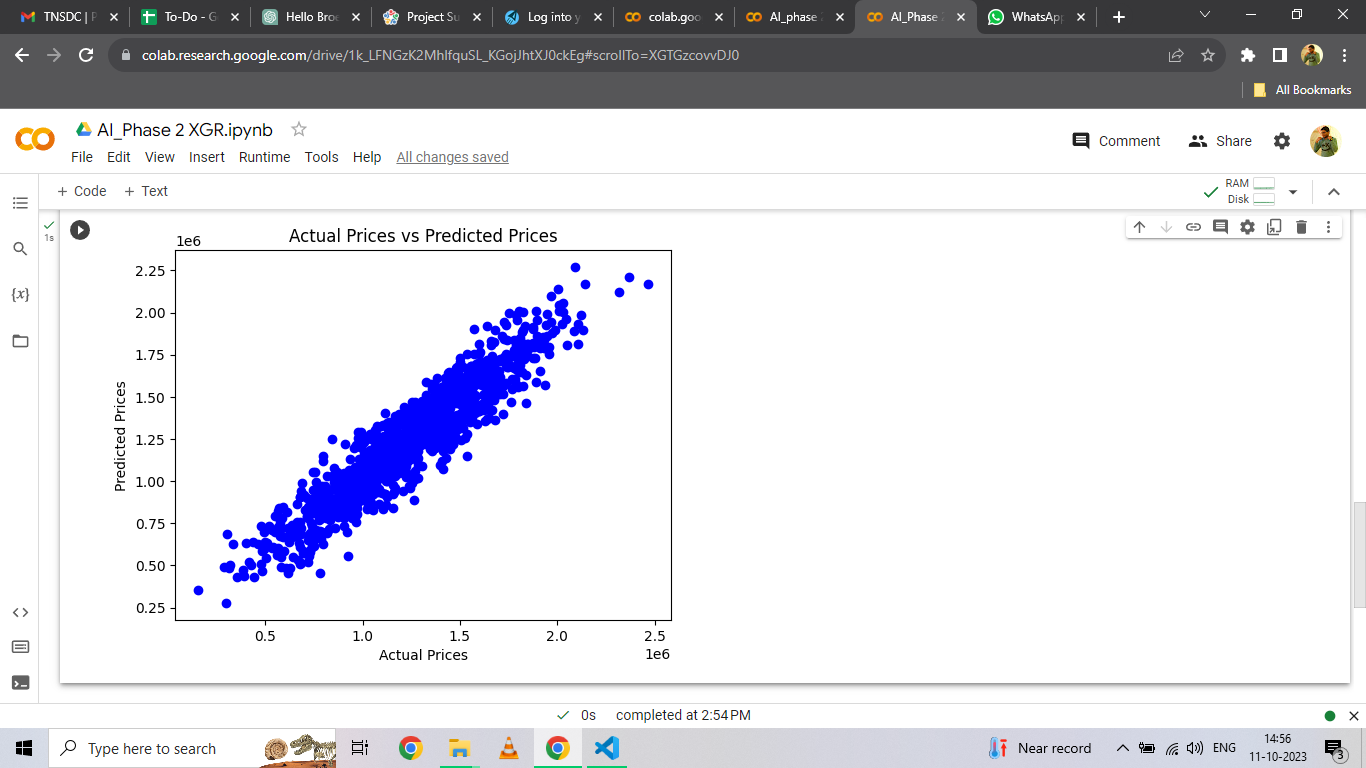
plt.scatter(y\_test, y\_pred, color='blue')

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Actual Prices vs Predicted Prices")

plt.show()



Conclusion

* Implementing XGBoost for regression in Google Colab showcases the power of advanced machine learning techniques in predictive modeling. With its efficiency and accuracy, XGBoost proves to be a valuable tool for data scientists and machine learning practitioners. Utilizing this technique opens up opportunities for accurate predictions in various domains, contributing to data-driven decision-making processes.
* This document provides an overview of how XGBoost can be utilized for regression tasks in Google Colab. Feel free to customize the content further based on your specific use case and audience.