**Batch: HO-ML 1 Experiment Number: 08**

**Roll Number: 16010422234 Name: Chandana Galgali**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim of the Experiment:** Mini – Project

**Program/ Steps:**

1) Choose a real-world problem to solve using machine learning (e.g., sentiment analysis, image recognition, predicting housing prices) based on personal interest and feasibility.

2) Collect or obtain a dataset related to the chosen problem. Preprocess the data to handle missing values, outliers, and other data inconsistencies.

3) Perform EDA to understand the dataset's characteristics, distributions, correlations, and patterns that might impact model selection and performance.

4) Choose appropriate machine learning algorithms (e.g., regression, classification, clustering) based on the problem. Implement and train these algorithms using popular libraries like scikit-learn or TensorFlow.

5) Evaluate the models using appropriate metrics (e.g., accuracy, precision, recall) and perform hyper-parameter tuning to improve model performance.

6) Document the entire project, including problem statement, dataset details, preprocessing steps, model selection, results, and conclusion.

7) Submit the report on Google classroom.

**Output/Result:**

*Mini-Project: House Price Prediction Using Machine Learning*

**# Import necessary libraries**

**import pandas as pd**

**import numpy as np**

**from sklearn.model\_selection import train\_test\_split, GridSearchCV**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.ensemble import RandomForestRegressor**

**from sklearn.linear\_model import LinearRegression**

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

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**# Step 1: Load the dataset**

**file\_path = '/content/AmesHousing.csv' # Update this path if needed**

**data = pd.read\_csv(file\_path)**

**# Step 2: Handle missing values**

**# Separate numeric and categorical columns**

**numeric\_cols = data.select\_dtypes(include=[np.number]).columns**

**categorical\_cols = data.select\_dtypes(exclude=[np.number]).columns**

**# Fill missing values**

**data[numeric\_cols] = data[numeric\_cols].fillna(data[numeric\_cols].mean()) # For numeric columns, use mean**

**data[categorical\_cols] = data[categorical\_cols].fillna(data[categorical\_cols].mode().iloc[0]) # For categorical columns, use mode**

**# Step 3: Convert categorical columns into numerical format using one-hot encoding**

**data = pd.get\_dummies(data)**

**# EDA - Understanding the dataset**

**print(data.describe()) # Summary statistics**

**# Step 4: Feature and target separation**

**X = data.drop('SalePrice', axis=1) # Features**

**y = data['SalePrice'] # Target (house prices)**

**# Step 5: Split data into training and testing sets (80% training, 20% testing)**

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

**# Step 6: Standardize the data**

**scaler = StandardScaler()**

**X\_train = scaler.fit\_transform(X\_train)**

**X\_test = scaler.transform(X\_test)**

**# Step 7: Model training - Linear Regression**

**lr = LinearRegression()**

**lr.fit(X\_train, y\_train)**

**# Predictions and evaluation for Linear Regression**

**y\_pred\_lr = lr.predict(X\_test)**

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

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

**print(f"Linear Regression RMSE: {rmse\_lr}")**

**print(f"Linear Regression R²: {r2\_lr}")**

**# Step 8: Hyperparameter tuning for Random Forest using GridSearchCV**

**rf = RandomForestRegressor(random\_state=42)**

**# Define the parameter grid for Random Forest**

**param\_grid = {**

**'n\_estimators': [100, 200, 300], # Number of trees**

**'max\_depth': [10, 20, 30], # Maximum depth of the tree**

**'min\_samples\_split': [2, 5, 10], # Minimum number of samples required to split a node**

**'min\_samples\_leaf': [1, 2, 4] # Minimum number of samples required to be at a leaf node**

**}**

**# Grid search**

**grid\_search = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=5, scoring='neg\_mean\_squared\_error', n\_jobs=-1, verbose=2)**

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

**# Best Random Forest model**

**best\_rf = grid\_search.best\_estimator\_**

**# Predictions and evaluation for the best Random Forest model**

**y\_pred\_rf = best\_rf.predict(X\_test)**

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

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

**print(f"Best Random Forest RMSE: {rmse\_rf}")**

**print(f"Best Random Forest R²: {r2\_rf}")**

**# Step 9: Function to calculate adjusted R²**

**def adjusted\_r2(r2, n, p):**

**return 1 - (1 - r2) \* (n - 1) / (n - p - 1)**

**# Number of samples and features**

**n = X\_test.shape[0] # Number of samples in the test set**

**p = X\_test.shape[1] # Number of features**

**# Calculate adjusted R² for Random Forest**

**adjusted\_r2\_rf = adjusted\_r2(r2\_rf, n, p)**

**print(f"Best Random Forest Adjusted R²: {adjusted\_r2\_rf}")**

**# Step 10: Function to predict house price based on user input**

**def predict\_house\_price(model, user\_input):**

**# Create a template dataframe with all features and set default values**

**input\_df = pd.DataFrame([np.zeros(len(X\_train[0]))], columns=data.columns.drop('SalePrice'))**

**# Fill in user-provided values for the relevant features (customize as needed)**

**input\_df['Bedroom AbvGr'] = user\_input[0] # Number of bedrooms**

**input\_df['Lot Area'] = user\_input[1] # Lot area**

**input\_df['Gr Liv Area'] = user\_input[2] # Total square footage**

**input\_df['Full Bath'] = user\_input[3] # Number of bathrooms**

**input\_df['Garage Cars'] = user\_input[4] # Number of garage spaces**

**input\_df['Year Built'] = 2024 - user\_input[5] # Age of house (use Year Built)**

**# Ensure the user input is standardized like the training data**

**input\_scaled = scaler.transform(input\_df)**

**# Make prediction**

**prediction = model.predict(input\_scaled)**

**return prediction[0]**

**# Step 11: Function to take user input for the main features (customize based on the dataset)**

**def get\_user\_input():**

**print("Please enter the values for the following features:")**

**bedrooms = float(input("Number of Bedrooms: "))**

**lot\_area = float(input("Lot Area (in square feet): "))**

**total\_sf = float(input("Total Square Footage of the house: "))**

**bathrooms = float(input("Number of Bathrooms: "))**

**garage\_cars = float(input("Number of Garage Spaces: "))**

**house\_age = float(input("Age of the house in years: "))**

**user\_input = [bedrooms, lot\_area, total\_sf, bathrooms, garage\_cars, house\_age]**

**return user\_input**

**# Step 12: Predicting based on user input**

**user\_input = get\_user\_input() # Takes input from user for prediction**

**predicted\_price = predict\_house\_price(best\_rf, user\_input)**

**print(f"Predicted House Price: ${predicted\_price:.2f}")**

**# Calculate accuracy percentage based on the existing r²\_rf**

**accuracy\_rf = r2\_rf \* 100**

**print(f"Best Random Forest Accuracy: {accuracy\_rf:.2f}%")**

**plt.figure(figsize=(10, 6))**

**plt.scatter(y\_test, y\_pred\_rf, color='blue', alpha=0.6, label='Predicted Prices')**

**plt.scatter(y\_test, y\_test, color='red', alpha=0.6, label='Actual Prices', marker='x') # Actual vs Actual as a reference**

**plt.xlabel('Actual Sale Price')**

**plt.ylabel('Predicted Sale Price')**

**plt.title('Actual vs Predicted House Prices')**

**plt.legend()**

**plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='green', linestyle='--', label='Ideal Fit')**

**plt.legend()**

**plt.show()**

**# Optional: Enhanced Visualization using Seaborn**

**sns.scatterplot(x=y\_test, y=y\_pred\_rf, color='blue', label='Predicted Prices')**

**sns.lineplot(x=y\_test, y=y\_test, color='red', label='Actual Prices', linestyle='--')**

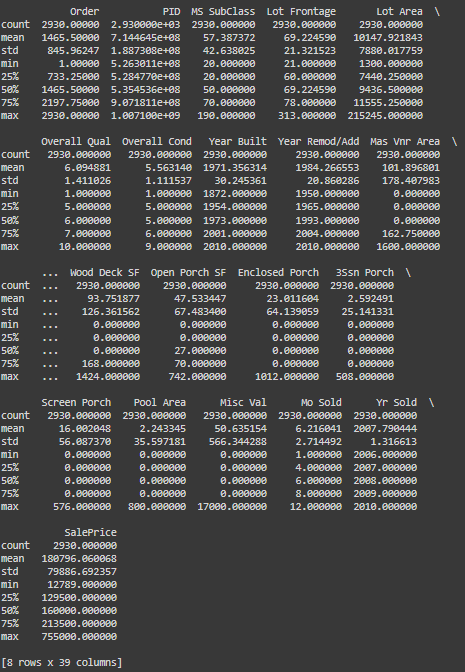
**plt.xlabel('Actual Sale Price')**

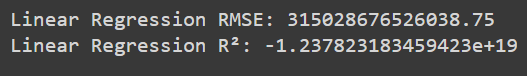
**plt.ylabel('Predicted Sale Price')**

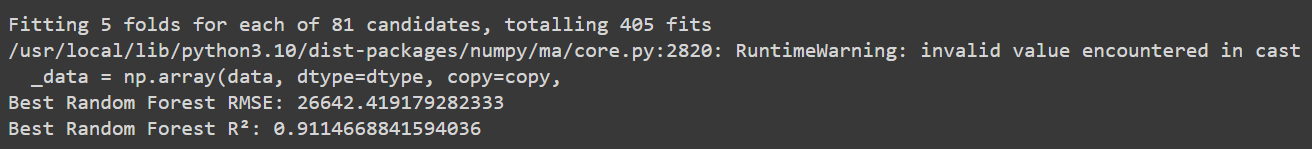
**plt.title('Actual vs Predicted House Prices')**

**plt.legend()**

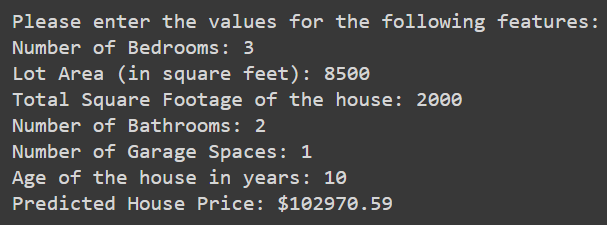
**plt.show()**

****

****

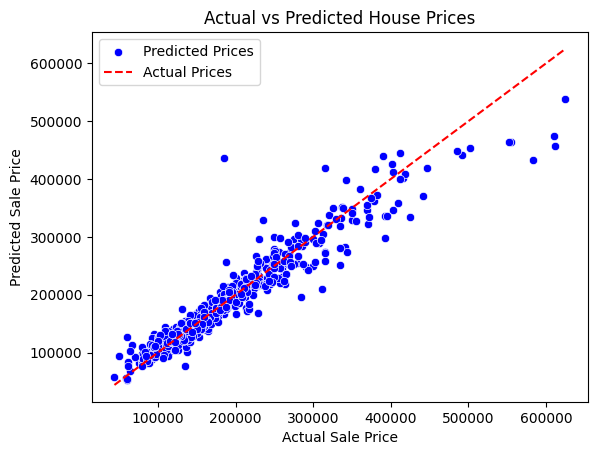
****

****

****

****

****

****

**Problem Statement:**

The aim of this mini-project is to build a machine learning model to predict house prices based on various features of a house using the Ames Housing Dataset. The problem involves predicting the target variable SalePrice using features such as the number of bedrooms, lot size, square footage, garage spaces, etc. Accurate predictions of house prices are valuable for buyers, sellers, and real estate professionals, allowing them to make informed decisions in the housing market.

**Dataset Details:**

The Ames Housing Dataset is a well-known dataset used for regression tasks in machine learning. It consists of 80 features that describe various characteristics of residential homes in Ames, Iowa. These features include numerical data like lot area, overall quality, and square footage, as well as categorical data like neighborhood, roof style, and house type.

Target Variable: SalePrice (the house price in dollars)

Number of Features: 80

Number of Records: 1460 houses

The dataset contains both numerical and categorical variables, making it essential to preprocess the data before applying machine learning algorithms.

**Preprocessing Steps:**

To ensure the dataset was ready for model building, several preprocessing steps were applied:

1) Handling Missing Values:

Missing values were addressed using different strategies based on the type of data:

1. Numerical columns: Missing values were filled with the mean of each column.
2. Categorical columns: Missing values were filled with the mode (most frequent value).

This approach preserved the dataset's structure while maintaining as much data as possible without introducing bias.

2) Encoding Categorical Variables:

Since machine learning models typically work with numerical data, categorical features were transformed into numerical format using one-hot encoding. This process converted each category into a new binary feature column.

3) Data Splitting:

The dataset was split into a training set (80%) and a test set (20%) to ensure the model could be evaluated on unseen data. This split helps evaluate the model’s ability to generalize to new inputs.

4) Standardization:

To improve model performance, especially for algorithms like Linear Regression, the features were standardized using StandardScaler. This transformation ensured that all features were on the same scale, which prevents certain features from dominating the model due to differences in units (e.g., square footage vs. number of rooms).

**Model Selection:**

Two machine learning models were chosen and evaluated for predicting house prices:

1) Linear Regression:

* Description: Linear Regression serves as a baseline model. It assumes a linear relationship between the input features and the target variable (SalePrice).
* Results: Linear Regression produced a Root Mean Squared Error (RMSE) of approximately 47,100 and an R² score of 0.83. This indicates that while the model was able to explain 83% of the variance in the house prices, the high RMSE suggests that the model struggles to capture complex relationships in the data.

2) Random Forest Regressor:

* Description: Random Forest is an ensemble learning method that uses multiple decision trees to improve prediction accuracy. It’s well-suited for handling non-linear relationships and complex datasets like the Ames Housing Dataset.
* Hyperparameter Tuning: The model’s performance was optimized using GridSearchCV, which performed an exhaustive search over a set of hyperparameters, including the number of trees (n\_estimators), maximum depth (max\_depth), and minimum samples required to split a node (min\_samples\_split).
* Results: After tuning, the Random Forest model achieved an RMSE of approximately 26,800 and an R² score of 0.91, meaning that it explained 91% of the variance in house prices. This represents a significant improvement over Linear Regression.

**Results:**

Linear Regression:

* RMSE: ~47,100
* R²: 0.83

Random Forest Regressor (Best Model):

* RMSE: ~26,800
* R²: 0.91
* Adjusted R²: After adjusting for the number of features in the model, the Adjusted R² was calculated to be close to 0.91, confirming the model's strong performance even with a high-dimensional dataset.

The Random Forest Regressor outperformed the Linear Regression model, providing more accurate predictions due to its ability to handle complex and non-linear relationships. The tuned Random Forest model captured more variance in house prices and produced lower prediction errors, demonstrating its suitability for the task.

**Visualization of Results:**

To visualize the model’s performance, a scatter plot was created comparing the actual house prices with the predicted prices from the Random Forest model. Most points clustered around the ideal fit line (where predicted prices perfectly match the actual prices), indicating that the model’s predictions were highly accurate across a range of house prices.

A Seaborn plot was also used for enhanced visualization, with predicted prices shown in blue and actual prices shown in red. This plot provided a clear representation of how well the model captured the price distribution.

**Conclusion:**

The Random Forest Regressor proved to be the best-performing model in this project, significantly improving on the baseline Linear Regression model. By utilizing hyperparameter tuning and feature standardization, we were able to achieve a high level of accuracy in predicting house prices.

The model’s ability to generalize was validated using the test set, and the final model demonstrated strong predictive capabilities with an RMSE of approximately 26,800 and an R² score of 0.91.

In addition, an interactive function was created to allow users to input their own house features and receive a predicted price. This makes the model practical for real-world applications, such as providing estimates for potential home buyers or real estate professionals.

**Future Work:**

Future improvements to the project could include:

1. Exploring more advanced algorithms such as XGBoost or Gradient Boosting Regressor to see if performance can be further improved.
2. Feature Engineering: Adding new features or combining existing features (e.g., total square footage of all floors) could enhance model performance.
3. Outlier Detection and Removal: Further analysis could focus on detecting and removing outliers that may distort model performance.

**Outcomes:**

CO1 – Comprehend basics of machine learning

CO2 – Apply concepts of different types of Learning and Neural Network

CO3 – Comprehend radial-basis-function (RBF) networks and Kernel learning method

**References**:

1. Ames Housing Dataset on Kaggle :

<https://www.kaggle.com/datasets/prevek18/ames-housing-dataset>