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()
```

```
             Order           PID  MS SubClass  Lot Frontage       Lot Area  \
count  2930.00000  2.930000e+03  2930.000000   2930.000000    2930.000000
mean   1465.50000  7.144645e+08    57.387372     69.224590   10147.921843
std     845.96247  1.887308e+08    42.638025     21.321523    7880.017759
min       1.00000  5.263011e+08    20.000000     21.000000    1300.000000
25%     733.25000  5.284770e+08    20.000000     60.000000    7440.250000
50%    1465.50000  5.354536e+08    50.000000     69.224590    9436.500000
75%    2197.75000  9.071811e+08    70.000000     78.000000   11555.250000
max    2930.00000  1.007100e+09   190.000000    313.000000  215245.000000

       Overall Qual  Overall Cond   Year Built  Year Remod/Add  Mas Vnr Area  \
count   2930.000000   2930.000000  2930.000000     2930.000000   2930.000000
mean       6.094881      5.563140  1971.356314     1984.266553    101.896801
std        1.411026      1.111537    30.245361       20.860286    178.407983
min        1.000000      1.000000  1872.000000     1950.000000      0.000000
25%        5.000000      5.000000  1954.000000     1965.000000      0.000000
50%        6.000000      5.000000  1973.000000     1993.000000      0.000000
75%        7.000000      6.000000  2001.000000     2004.000000    162.750000
max       10.000000      9.000000  2010.000000     2010.000000   1600.000000

       ...  Wood Deck SF  Open Porch SF  Enclosed Porch   3Ssn Porch  \
count  ...   2930.000000    2930.000000     2930.000000  2930.000000
mean   ...     93.751877      47.533447       23.011604     2.592491
std    ...    126.361562      67.483400       64.139059    25.141331
min    ...      0.000000       0.000000        0.000000     0.000000
25%    ...      0.000000       0.000000        0.000000     0.000000
50%    ...      0.000000      27.000000        0.000000     0.000000
75%    ...    168.000000      70.000000        0.000000     0.000000
max    ...   1424.000000     742.000000     1012.000000   508.000000

       Screen Porch    Pool Area     Misc Val      Mo Sold      Yr Sold  \
count   2930.000000  2930.000000   2930.000000  2930.000000  2930.000000
mean      16.002048     2.243345     50.635154     6.216041  2007.790444
std       56.087370    35.597181    566.344288     2.714492     1.316613
min        0.000000     0.000000      0.000000     1.000000  2006.000000
25%        0.000000     0.000000      0.000000     4.000000  2007.000000
50%        0.000000     0.000000      0.000000     6.000000  2008.000000
75%        0.000000     0.000000      0.000000     8.000000  2009.000000
max      576.000000   800.000000  17000.000000    12.000000  2010.000000

           SalePrice
count    2930.000000
mean   180796.060068
std     79886.692357
min     12789.000000
25%    129500.000000
50%    160000.000000
75%    213500.000000
max    755000.000000

[8 rows x 39 columns]
```

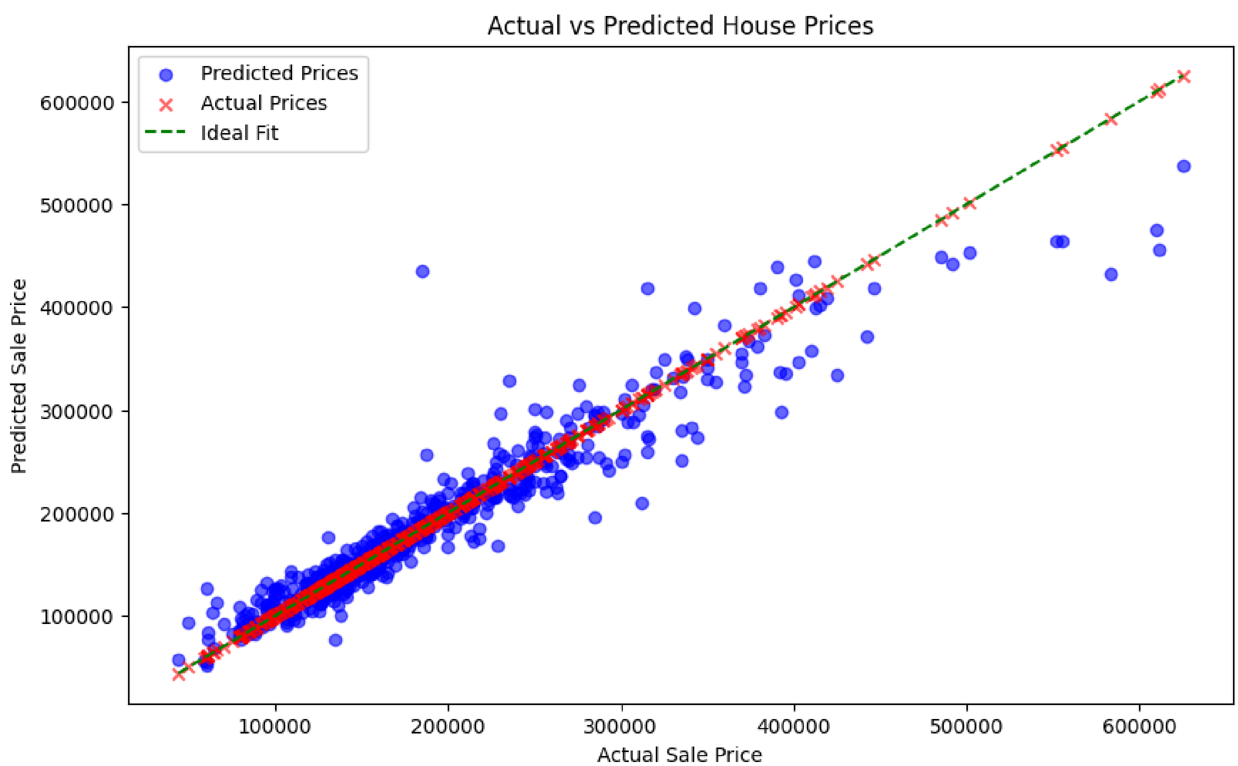
```
Linear Regression RMSE: 315028676526038.75
Linear Regression R²: -1.237823183459423e+19
```

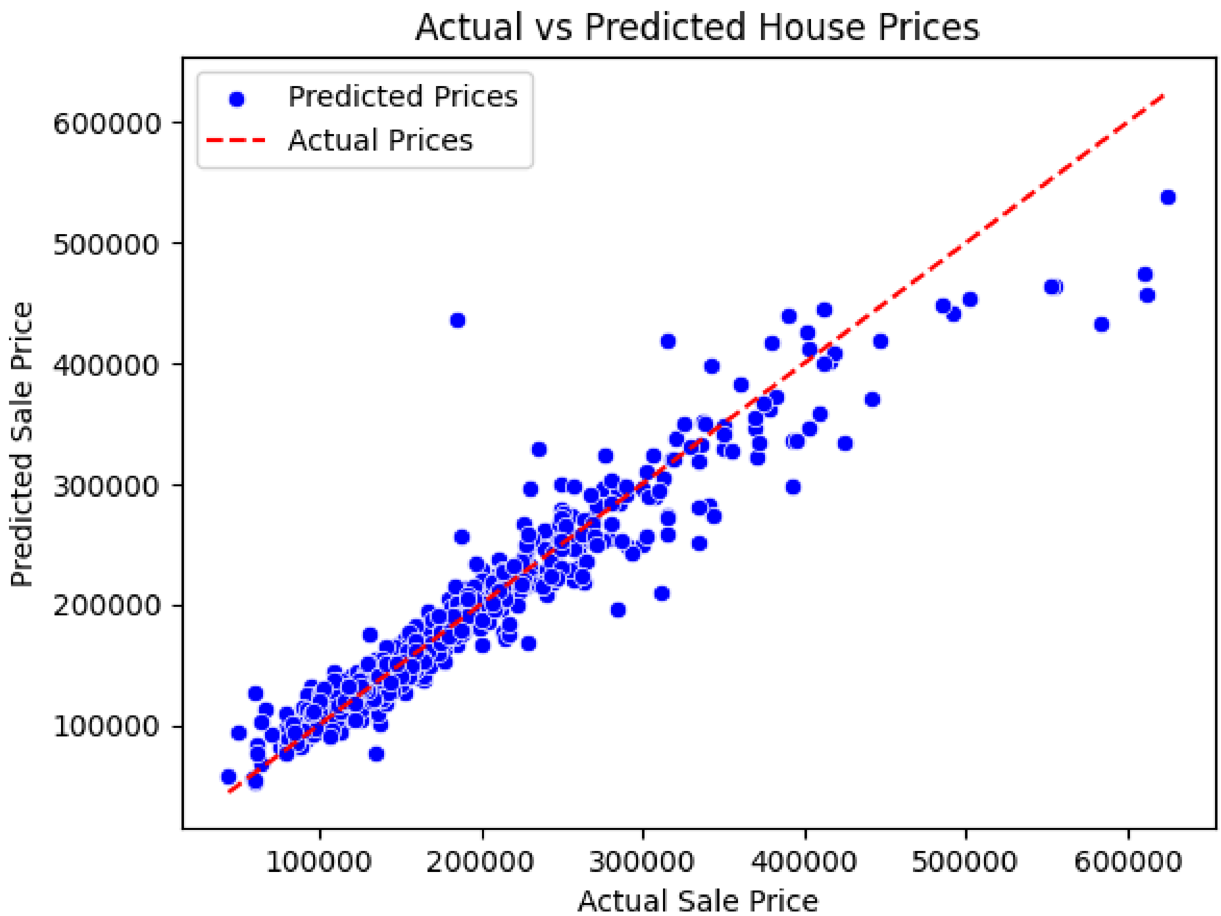
```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
Best Random Forest RMSE: 26642.419179282333
Best Random Forest R²: 0.9114668841594036
```

```
Best Random Forest Adjusted R²: 0.8150290258330396
```

```
Please enter the values for the following features:
Number of Bedrooms: 3
Lot Area (in square feet): 8500
Total Square Footage of the house: 2000
Number of Bathrooms: 2
Number of Garage Spaces: 1
Age of the house in years: 10
Predicted House Price: $102970.59
```

```
Best Random Forest Accuracy: 91.15%
```





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 R^2 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 R^2 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
- R^2: 0.83

Random Forest Regressor (Best Model):

- RMSE: ~26,800
- R^2: 0.91
- Adjusted R^2: After adjusting for the number of features in the model, the Adjusted R^2 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 R^2 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

---