HOUSE PRICE PREDICTION

presented by:

Shatha Alfaris
Raghad thabet
Dima alotaibi
Leen albahli
Sara alhosain

Introduction

This project focuses on applying machine learning techniques to predict house prices based on a dataset provided by the Kaggle competition titled “House Prices - Advanced Regression Techniques.” It serves as a practical application of data analysis, feature selection, and predictive modeling to address a real-world problem. The primary objective is to build an accurate model that can estimate house prices based on various property features.

Description

1. Understanding the Problem:
Our main goal is to predict house prices based on a large set of features.

2. Data Collection:
We used a dataset sourced from Kaggle, which includes a wide range of housing features.

3. Steps Followed:
First, we performed data preprocessing to handle missing values.
then, we worked on feature selection and engineering to enhance model accuracy.
Then, we trained our models and fine-tuned their parameters.
Finally, we evaluated the models using appropriate performance metrics to ensure reliability

Problem Statement

The Problem:
Predicting house prices is a complex challenge due to the wide variety of factors influencing property values, such as location, size, and condition. Real estate stakeholders often lack accurate tools to make data-driven pricing decisions.

Objective:
To develop a machine learning model capable of accurately predicting house prices by analyzing historical data and identifying the most influential property features.

Impact:
- Simplifies pricing decisions for buyers, sellers, and investors.
- Demonstrates the application of data science in solving real-world problems.
- Provides reliable predictions to support fair market value assessments.

Importing libaries

```
import pandas as pd
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.ensemble import RandomForestRegressor
```

pandas (pd) :Library for data manipulation and analysis.

SelectKBest :Selects the top K most important features.

f_regressionComputes: the F-statistic to evaluate the relationship.

RandomForestRegressor:A machine learning algorithm that predict continuous values for regression tasks.

```
dataset_df = pd.read_csv("/kaggle/input/house-prices-advanced-regression-techniques/train.csv")
dataset_df.head(3)

dataset_df = dataset_df.drop('Id', axis=1)
dataset_df.head(3)

dataset_df.info()
```

- Reads a CSV file into a pandas DataFrame.
- Removes the Id column from the dataset as it'snot useful for analysis.

```
list(set(dataset_df.dtypes.tolist()))
```

- dataset_df.dtypes:Retrieves the data types of all columns in the DataFrame .
- .tolist():Converts the data types into a Python list.
- set(): Removes duplicate data types from the list.
- list(): converts the resulting set back into a list for display.

Drop null columns

```
#drop null cols
columns_to_drop = df_num.columns[df_num.count() != 1460]

df_num.drop(columns_to_drop, axis=1, inplace=True)
df_num.info()
```

1. Identify Columns with Null Values

2. Drop Columns with Null Values

3. View Updated Dataset Info

```
df_num = dataset_df.select_dtypes(incl
ude = ['float64', 'int64'])
df_num.head()
```

- In this step we ensures that the data will take only numeical values

```
#drop null cols
columns_to_drop = df_num.columns[df_nu
m.count() != 1460]

df_num.drop(columns_to_drop, axis=1, i
nplace=True)
df_num.info()
```

- Ensures the DataFrame contains only fully complete columns.

```
###############
def split_dataset(dataset, test_ratio=
0.30, random_seed=42):
    # Set the random seed for reproduci
bility
    np.random.seed(random_seed)

    # Generate random indices for the t
est set based on the test ratio
    test_indices = np.random.rand(len(
dataset)) < test_ratio

    # Split the dataset into training a
nd testing based on the random indices
    train_ds_pd = dataset[~test_indice
s]  # Select training set (non-test ind
ices)
    valid_ds_pd = dataset[test_indice
s]   # Select validation set (test indi
ces)

    return train_ds_pd, valid_ds_pd
```

This function split_dataset splits a dataset into training and validation.

Define the Non-Null features

```
#non-null features
train_ds_pd_non, valid_ds_pd_non = split_dataset(df_num)
print("{} examples in training, {} examples in testing.".format(
    len(train_ds_pd_non), len(valid_ds_pd_non)))
```

- split df_num into train and valdiation

Preparing data

```
# Prepare data
X = train_ds_pd_non.drop('SalePrice', axis=1)  # Features
y = train_ds_pd_non['SalePrice']  # Target variable
```

- **X:** Contains all features by dropping the target variable SalePrice from the DataFrame.
- **y:** Contains the target variable SalePrice for prediction.

```
# Drop redundant features based on correlation
# Compute the correlation matrix
correlation_matrix = X.corr().abs()
# Set a threshold for high correlation
threshold = 0.9
to_drop = set()
```

- **Compute Correlation:** Creates a correlation matrix to assess feature relationships.
- **Threshold:** Sets a threshold (0.9) for high correlation.

```python
# Loop over the upper triangle of the correlation matrix
for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > threshold:
            colname = correlation_matrix.columns[i]
            to_drop.add(colname)

# Drop the highly correlated features
X = X.drop(to_drop, axis=1)



print(f"Dropped redundant features (correlation > {threshold}): {to_drop}")
```

- **Loop:** Identifies and collects features that are highly correlated.
- **Drop Features:** Removes the redundant features from X.
- **Output:** Prints the names of the dropped features.

Feature selection

```
selector = SelectKBest(f_regression, k=15)  # Choose top 1
5 features
```

- Creates a SelectKBest object called selector, using the f_regression method as the feature selection criterion.
- Sets k=15 to select the top 15 most important features.

```
X_filtered = selector.fit_transform(X, y)
correlation_features = X.columns[selector.get_support(indi
ces=True)]
```

- Applies the fit_transform method to the feature matrix X and the target variable y.

```
# Tree-Based Feature Importance
model = RandomForestRegressor()
model.fit(X_filtered, y)
```

```
# Get feature importances
importances = model.feature_importances_
feature_names = X.columns
```

We let a RandomForestRegressor model to calculate the importance of features.
This importance reflects how much each feature affects the model's predictions

```
plt.figure(figsize=(10, 6))  # Set figure size
```

- **Creates a figure:** Sets the size of the plot to 10 inches wide and 6 inches tall for better visibility.

```
plt.barh(sorted_features[:15], sorted_importances[:15], color
='skyblue')  # Create horizontal bar chart
```

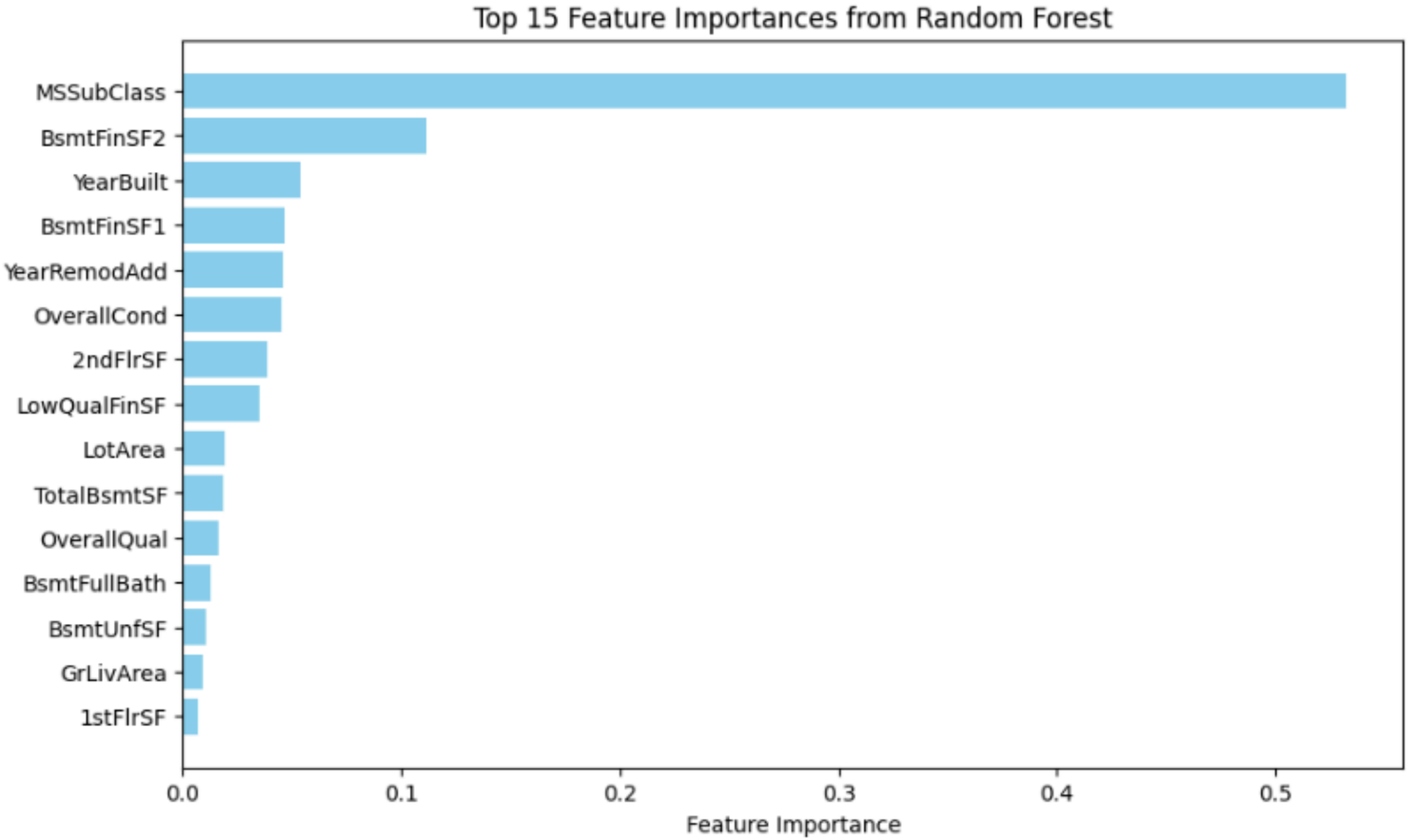
- **Horizontal bar chart**: Plots the top 15 features and their importance scores. Bars are colored sky blue.

- **X-axis label**: Indicates that the x-axis represents the importance scores of the features.

```
plt.gca().invert_yaxis()  # Highest importance at the top
plt.show()  # Display the plot
```

- **Invert y-axis:** Reverses the order of the y-axis so the most important feature appears at the top.

- **Display the plot:** Renders the chart for visualization.


Top 15 Feature Importances from Random Forest
MSSubClass
BsmtFinSF2
YearBuilt
BsmtFinSF1
YearRemodAdd
OverallCond
2ndFlrSF
LowQualFinSF
LotArea
TotalBsmtSF
OverallQual
BsmtFullBath
BsmtUnfSF
GrLivArea
1stFlrSF
0.0
0.1
0.2
0.3
0.4
0.5
Feature Importance

Training the Model

Training a Random Forest Regressor

```
model = RandomForestRegressor(random_state=42)
model.fit(X_filtered, y)
```

A Random Forest Regressor is trained using the filtered input features (X_filtered) and the target variable (y, e.g., SalePrice)

Loading the Test Dataset

```
test = pd.read_csv("/kaggle/input/house-prices-advanced-regression-techniques/t
```

The test dataset is loaded using pandas and its structure is inspected (e.g., column names, data types, and missing values)

Selecting Relevant Features

```
test = test[correlation_features]
```

Only the important features (stored in correlation_features) are retained from the test dataset.

Handling Missing Values

```
test.fillna(test.mean(), inplace=True)
test.info()
```

Missing values in the test dataset are replaced with the mean value of each column to ensure the data is usable for prediction

Validating the Model

Preparing Validation Data

```
X_valid = valid_ds_pd_non.drop('SalePrice', axis=1).drop(to_drop, axis=1)
X_valid_filtered = selector.transform(X_valid)
y_valid = valid_ds_pd_non['SalePrice']
```

The validation dataset is prepared by:

- Removing the target column (SalePrice) and unnecessary columns (to_drop).
- Filtering the relevant features using a selector (selector).
- Separating the target variable (y_valid).

Making Predictions and Evaluating the Model

```
y_pred = model.predict(X_valid_filtered)
mse = mean_squared_error(y_valid, y_pred)
print(f"Mean Squared Error on validation set: {mse:.2f}")
```

Predictions are made using the validation data, and the Mean Squared Error (MSE) is calculated to measure the model's accuracy. A lower MSE indicates better performance.

Preprocessing the Test Data

```
test_filtered = test[correlation_features]
```

The test dataset is filtered to include only the relevant features (correlation_features)

Making Predictions

```
new_predictions = model.predict(test_filtered)

# Print the predictions
print("",new_predictions)
```

The trained model predicts the target values (e.g., SalePrice) for the test dataset.

Creating a DataFrame for Predictions

```
predicted_df = pd.DataFrame({'SalePrice': new_predictions})
```

The predictions are stored in a new DataFrame, which can later be saved as a CSV file or used for further analysis.

Results Sample

	Id	SalePrice
0	1461	120890.49
1	1462	147970.30
2	1463	176647.27
3	1464	178687.00
4	1465	201990.64
5	1466	184278.75
6	1467	176031.15
7	1468	175395.92
8	1469	184410.50
9	1470	121281.00

Conclusion

In this project, we built a model to predict house prices using data that includes 79 features about homes. By using machine learning techniques, we were able to understand how different factors, like the size of the house and its location, affect the price.

Our model provides predictions, which can help people make better decisions when buying or selling homes. In the future, we can improve the model by adding more data or trying other advanced methods to make it even better.

Thank You for listening!