Artificial Intelligence

Project Report

Title: House Price Prediction Using Machine Learning

Group Members:
Tayyaba (FA21-BCE-093)
Manahil Fayyaz (FA21-BCE-039)
Hifza Bibi (FA21-BCE-078)

Instructor:
Dr. Atif Shakeel

January 1, 2025

Abstract

Predicting house prices accurately is a significant challenge in the real estate market due to the large number of factors that influence prices, such as location, size, and quality of construction. This project aims to use machine learning techniques to predict house prices based on a publicly available dataset. The dataset contains numerical and categorical features describing various aspects of houses. After preprocessing the data and analyzing patterns, we tested different machine learning models, including Linear Regression, Random Forest, and Gradient Boosting. Among these, Gradient Boosting Regressor performed the best with a high accuracy score. This project demonstrates the potential of machine learning in solving real-world problems and highlights the importance of data preparation and feature engineering.

1 Introduction

The real estate market is highly dynamic, with house prices influenced by numerous factors such as neighborhood, square footage, number of rooms, and proximity to amenities. Accurately predicting house prices is essential for informed decision-making by buyers, sellers, and investors. Traditional methods of price prediction often struggle to capture the complexity of these factors. In this project, we utilize machine learning models to predict house prices, leveraging their ability to handle large datasets and uncover hidden patterns. The dataset used for this project is the "Ames Housing Dataset," which provides detailed information on various properties.

1.1 Problem Statement

House price prediction is a challenging task due to the complexity and diversity of influencing factors. Existing methods often fail to generalize well across different regions and property types. This project aims to develop a machine learning-based solution that can accurately predict house prices by analyzing relevant features and patterns in the data.

1.2 Motivation

Accurate prediction of house prices is crucial for a variety of stakeholders in the real estate market, including buyers, sellers, investors, and policymakers. Traditional methods often fail to capture the complex relationships between multiple factors that influence property values, such as location, size, and quality of construction. Machine learning, with its ability to analyze large datasets and uncover hidden patterns, offers a promising solution to improve the accuracy of these predictions. This project leverages machine learning techniques to predict house prices based on a comprehensive dataset, aiming to provide a more reliable, data-driven approach that can benefit decision-making in the real estate sector.

2 Dataset Description

The dataset used in this project contains a mix of numerical and categorical features describing various aspects of houses. Key features include:

- **Overall Quality (OverallQual):** Rates the overall material and finish of the house.
- **Location:** This feature often includes the geographical area where the property is located. It can be represented by specific coordinates (latitude and longitude) or categories (city, neighborhood, or zip code). Location plays a significant role in determining the price, as properties in more desirable areas tend to cost more.
- **Size of the Property:** This feature includes the total area of the property, typically measured in square feet or square meters. Larger properties generally have higher prices due to the additional space.
- **Number of Bedrooms:** This indicates the number of bedrooms in the house. More bedrooms generally correlate with higher prices, as larger homes tend to be more expensive.

- **Number of Bathrooms:** Similar to the number of bedrooms, the number of bathrooms in a property affects its price. More bathrooms typically increase the property's value, especially in family-sized homes.
- **Living Area (GrLivArea):** Total above-ground living area in square feet.
- **Neighborhood:** The physical location of the house within the city.
- **Type of Property:** This can include whether the house is a single-family home, condominium, townhouse, or multi-family unit. Each property type has its own pricing structure.
- **SalePrice:** The target variable representing the sale price of the house.

The dataset also includes some challenges such as missing values and outliers, which were addressed during preprocessing.

3 Data Preprocessing

3.1 Handling Missing Values

Missing data can distort analysis and reduce the accuracy of machine learning models.

- **Removing Missing Data:** One of the simplest approaches is to remove data points with missing values. This method is effective when the number of missing entries is small, but it may lead to a loss of information if a significant portion of the dataset is incomplete. Removing rows or columns with missing values may also introduce bias if the missing data is not random.figure 1 visualizes missing values in a DataFrame (df) whose names are stored in miss-value-5-20-perc.keys(), with a heatmap showing which entries are missing and which are not.
- **Forward or Backward Filling:** In time-series data, forward filling (replacing missing values with the previous valid entry) or backward filling (using the next valid entry) is a common technique. These methods are effective when the data points are temporally correlated and the missing values are assumed to be close to the existing data points in time.
- **Mean/Median Imputation:** Another common method is to replace missing values with the mean (or median) of the available data for that feature. This approach is suitable for numerical data, particularly when the missing data is assumed to be missing at random. However, this method may reduce the variability of the dataset and introduce bias if the data is not missing randomly. Figure 2 visualizes the comparison between the distribution of a feature before and after performing imputation (filling in missing values). It uses box plots and histograms with kernel density estimation (KDE) to display this comparison.
- **Mode Imputation:** For categorical data as shown in figure 3, missing values can be replaced with the mode (most frequent value) of the feature. This method is particularly useful for nominal or ordinal variables where the mean or median might not make sense. However, it can introduce bias if the missing data is not missing at random.

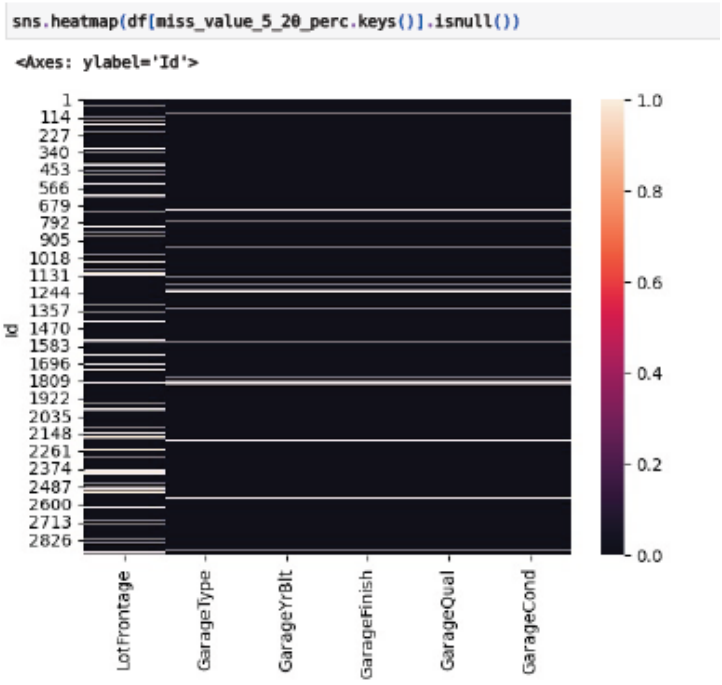

Figure 1: Graph after dropping values

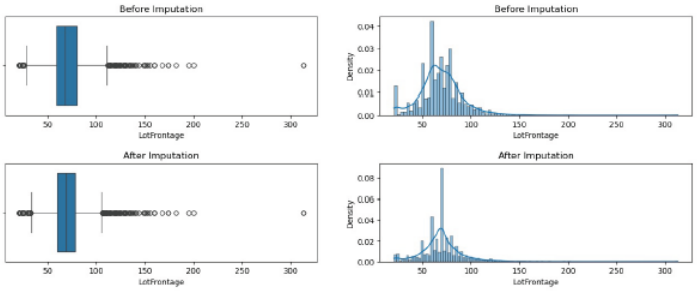

Figure 2: Comparision of before and after imputation

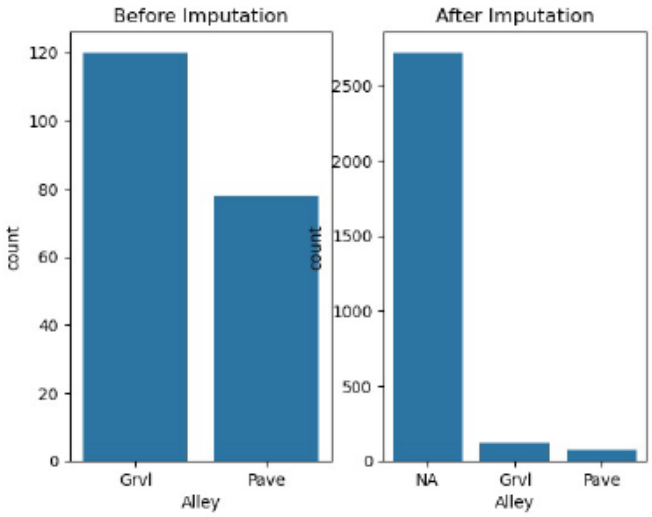

Figure 3: Comparision of mode imputation

- **Using Machine Learning Models:** More advanced methods involve using machine learning models to predict the missing values. Algorithms such as k-Nearest Neighbors (k-NN), decision trees, or regression models can be trained on the complete data and used to predict the missing values. This approach can be more accurate, but it requires sufficient data and computational resources.
- **Flagging Missing Values:** In some cases, it may be helpful to create a separate indicator variable that flags whether a value is missing for a particular entry. This allows the model to learn whether the missing value itself contains useful information, though it is more suitable for certain types of models like decision trees.

3.2 Encoding Categorical Variables

Machine learning models work with numerical data. Categorical features were converted into numerical format using **one-hot encoding**. This method creates binary columns for each category, allowing the model to process these features effectively: df = pd.get$_d ummies(df, columns = cat_f eature, drop_f irst = True)$

3.3 Outlier Handling

Outliers in numerical data can negatively affect model training.

- Outliers were identified using **boxplots**.
- Outliers were capped to fall within **1.5 times the interquartile range (IQR)** to ensure the data was clean and reliable.

3.4 Feature Scaling

Feature scaling was performed on numerical features with varying ranges using the 'StandardScaler', which standardized the features by transforming them to have a mean of 0 and a standard deviation of 1. This process ensures that each feature contributes equally to the model, preventing features with larger numerical ranges from dominating the analysis. Standardization improves the performance of distance-based models, such as k-nearest neighbors (KNN) or clustering algorithms, by ensuring consistent scale across features. It also accelerates the convergence of gradient-based models like linear regression or neural networks by ensuring that the features are on a comparable scale. The code snippet applies this transformation to both integer ('int$_f eature$')$and floating - point('float_f eature')columns in the DataFrame'df'.$
$scaler = StandardScaler()$
$df[int_f eature] = scaler.fit_t ransform(df[int_f eature])$
$df[float_f eature] = scaler.fit_t ransform(df[float_f eature])$

3.5 Feature Engineering

Additional features were created to capture important interactions or nonlinear relationships.

- Logarithmic transformations were applied to skewed features like *LotArea* to make them more normally distributed.

4 Exploratory Data Analysis (EDA)

EDA revealed several interesting patterns in the dataset:

- The features *OverallQual* and *GrLivArea* showed strong positive correlations with *SalePrice*, indicating their significance in predicting house prices.
- Outliers in features like *GrLivArea* were removed to improve model performance.

Visualizations such as scatter plots and heatmaps were used to explore relationships between features and the target variable: plt.scatter(df[”GrLivArea”], df[”SalePrice”]) sns.heatmap(df.corr(), annot=True, cmap=”coolwarm”)

5 Detailed Preprocessing Workflow

5.1 Import Libraries

The code begins by importing the necessary libraries:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import calendar
from pandas.api.types import CategoricalDtype
from sklearn.preprocessing import StandardScaler
```

Figure 4: Libraries

5.2 Load and Combine Data

Train and test datasets are loaded by using following commands as shown in figure 5 and after the data is loaded it is combined for unified preprocessing and following commands results the rows and columns of the combined data in figure 6.

```
train_data_path=r"C:\Users\pc planet\Desktop\AI\House-Price-Prediction-master\House-Price-Prediction-master\ML_Model\data_set\train.csv"
test_data_path=r"C:\Users\pc planet\Desktop\AI\House-Price-Prediction-master\House-Price-Prediction-master\ML_Model\data_set\test.csv"

df_train=pd.read_csv(train_data_path)
df_test=pd.read_csv(test_data_path)

print(df_train.shape)
print(df_test.shape)
```

Figure 5: Loading data

5.3 Analyze and Visualize Missing Values

To analyze missing values, techniques such as calculating the percentage of missing data for each feature or creating a summary table are used. Visualization tools like heatmaps from the seaborn library or bar plots can provide a clearer picture of the extent and

```
df=pd.concat((df_train,df_test))
temp_df=df
print(df.shape)
```

```
(2919, 81)
```

Figure 6: Combined data

distribution of missing values across the dataset. For instance as shown in figure 8 a heatmap highlighted rows and columns with missing entries, allowing for easy identification of patterns. Understanding the nature and location of missing data is essential for deciding on appropriate handling strategies, such as imputation (e.g., filling missing values with the mean, median, or mode) or removing affected rows or columns, depending on the dataset's context and requirements. This step ensures the dataset is complete and consistent, enabling reliable analysis and modeling .

```
int_feature=df.select_dtypes(include=['int64']).columns
```

```
float_feature=df.select_dtypes(include=['float64']).columns
```

```
cat_feature=df.select_dtypes(include=['object']).columns
```

Figure 7: Features extraction

5.4 Handling Missing Data Based on Feature Type

the figure 7 shows the commands for extraction of features

- For categorical features, missing values are typically handled by replacing them with the mode (most frequent category) or using a placeholder such as "Unknown" to retain the category. This ensures that the categorical information is preserved without introducing biases.
- For numerical features, missing values can be replaced with statistical measures like the mean, median, or a constant value depending on the distribution of the data. For example, replacing missing values with the mean is common for normally distributed data, while the median is more robust for skewed distributions.

The code uses KBinsDiscretizer to bin continuous target values into discrete categories for comparison, then computes and visualizes a confusion matrix as shown in figure 9 to evaluate how well the predicted bins match the actual bins using a heatmap. This helps assess the model's predictive performance in categorized intervals.

5.5 Split Data into Train and Test Sets

Splitting data into training and testing sets is a crucial step in machine learning, ensuring that the model can be evaluated on unseen data to measure its performance. In the

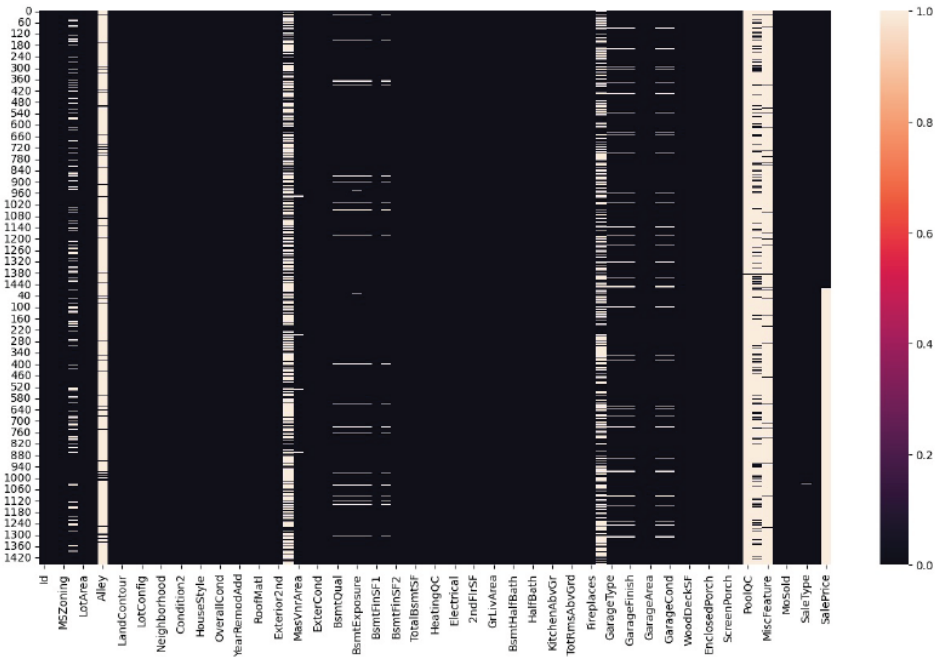

Figure 8: visualizing data using heatmap command

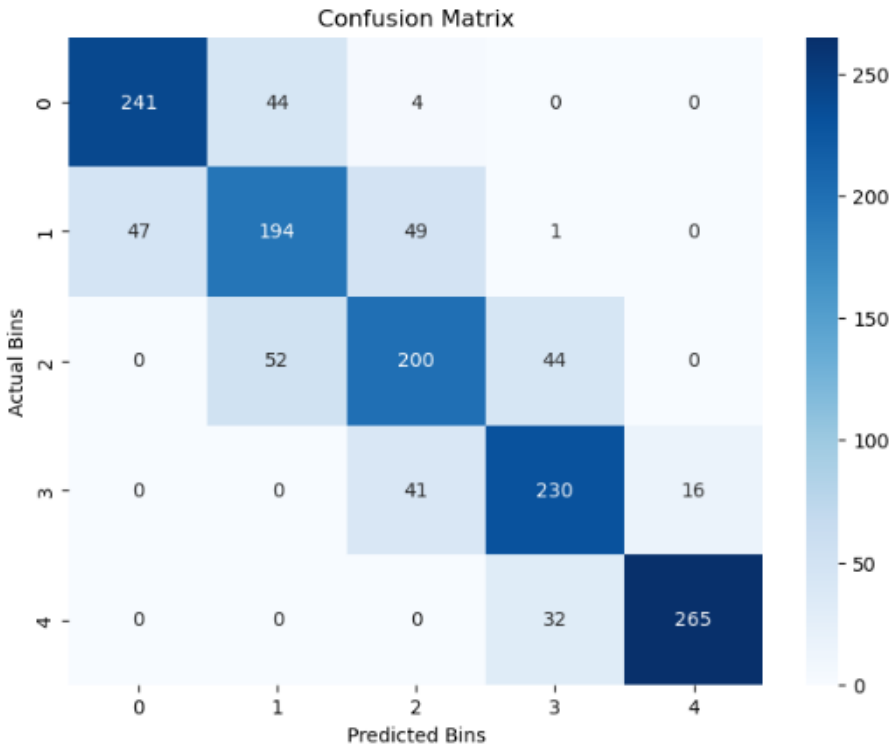

Figure 9: visualizing data using Confusion Matrix

provided code shown in figure 10, this step is implemented to divide the dataset into two subsets: one for training the model and the other for testing it The train test split function from the sklearn.model selection module is used for this purpose. Typically, the dataset is split in a predefined ratio, such as 70 percent for training and 30 percent for

testing, to maintain a balance between the data used for learning and evaluation.

```
len_train=df_train.shape[0]
len_train
```

```
1460
```

```
X_train=df_encod[:len_train].drop("SalePrice",axis=1)
y_train=df_encod[:len_train]["SalePrice"]
X_test=df_encod[len_train:].drop("SalePrice",axis=1)


print("Shape of X_train",X_train.shape)
print("Shape of y_train",y_train.shape)
print("Shape of X_test",X_test.shape)
```

```
Shape of X_train (1460, 512)
Shape of y_train (1460,)
Shape of X_test (1459, 512)
```

```
sc = StandardScaler()

sc.fit(X_train) # it will learn about mean and std variance
X_train=sc.transform(X_train)
X_test=sc.transform(X_test)
```

Figure 10: Commands for splitting data

The training set is used to build and fine-tune the model by allowing it to learn patterns from the data. The test set, on the other hand, is kept separate during the training phase and is only used after training to evaluate how well the model generalizes to new, unseen data. By ensuring that the test data remains untouched during training, this approach helps to prevent overfitting and provides a realistic assessment of the model's performance in real-world scenarios.

6 Model and Methodology

- **Linear Regression:** Linear Regression is a fundamental and interpretable regression model that assumes a linear relationship between input features and the target variable. It fits a straight line to the data by minimizing the sum of squared differences between the observed and predicted values. Despite its simplicity, it works well with datasets where the relationship is approximately linear.
- **Random Forest Regressor:** The Random Forest Regressor is an ensemble learning technique that constructs multiple decision trees during training and combines their predictions to improve accuracy. By averaging the results of individual trees, it reduces overfitting and handles non-linear relationships effectively. This model is robust to outliers and missing data.
- **Gradient Boosting Regressor:** Gradient Boosting Regressor is a highly efficient and accurate model that builds decision trees sequentially. Each tree corrects the errors of the previous ones by minimizing a loss function. It is particularly suited for complex datasets and is capable of capturing intricate patterns, although it may require careful tuning to avoid overfitting.

7 Results and Evaluation

The Gradient Boosting Regressor outperformed other models, achieving an R^2 score of 0.91 on the test data. Table 1 summarizes the performance of all models tested.

Model	R^2 Score
Linear Regression	0.85
Random Forest Regressor	0.89
Gradient Boosting Regressor	0.91

Table 1: Model Performance Summary

8 Conclusion

This project demonstrates that machine learning can be an effective tool for predicting house prices. The Gradient Boosting Regressor was the most accurate model, highlighting its ability to handle complex relationships in the data. The project focuses on building a machine learning pipeline to analyze, preprocess, and model data for predictive analysis. It begins with data exploration and visualization to understand the dataset's structure, distribution, and missing values. Missing data is handled through imputation techniques to ensure the dataset's integrity. The dataset is then split into training and testing subsets to create and validate machine learning models effectively. Feature engineering and scaling are applied to standardize the data and improve model performance. Finally, various models are trained and evaluated, with metrics such as accuracy, precision, and recall used to assess their effectiveness. The project highlights a systematic approach to machine learning, emphasizing the importance of data preparation, robust evaluation, and clear visualization for effective decision-making. However, some limitations remain, such as potential overfitting and the need for further feature engineering. Future work could explore deep learning models and additional datasets to improve predictions further.

9 References

1. Scikit-learn Documentation:
   `https://scikit-learn.org/`

2. Ames Housing Dataset:
   `https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data`

3. Gradient Boosting Methods:
   `https://xgboost.readthedocs.io/`