INTERNSHIP PROJECT REPORT

HOUSE PRICE PREDICTION USING REGRESSION

Declaration by Author

This is to declare that this report has been written by me. No part of the report is plagiarized from other sources. All information included from other sources has been duly acknowledged. I aver that if any part of the report is found to be plagiarized, I shall take full responsibility for it.

Signature:
Name: [Ansari Abdullah Shamim]
Place: [Bhiwandi]
Date: [05-12-2024]

Abstract

The aim of this project was to predict house prices using regression techniques on a structured dataset. The project involved data preprocessing, exploratory data analysis (EDA), feature engineering, and evaluation of different regression models, including Linear Regression, Decision Tree, and Random Forest. Python libraries such as Pandas, NumPy, Matplotlib, Seaborn, and scikit-learn were utilized. Key steps included cleaning the dataset, handling missing values, scaling, and encoding categorical data. The Random Forest model emerged as the most accurate with an R2R^2R2 score of 0.9858 and minimal error metrics.

TABLE OF CONTENTS

1. Introduction

1.1 Problem Addressed

Predicting house prices accurately is critical for real estate decision-making. However, the dataset contained missing values, outliers, and inconsistencies that hinder reliable analysis and prediction.

1.2 Importance of Problem

Accurate house price prediction aids buyers, sellers, and policymakers in making informed decisions. Clean and processed data is essential for building reliable predictive models.

1.3 Scope of the Project

This project aimed to preprocess a house price dataset, perform exploratory data analysis, and build regression models for prediction. Feature engineering and advanced modeling techniques were also applied to optimize prediction accuracy.

2. Approach

2.1 Dataset Exploration

- The dataset was loaded into a Pandas DataFrame.
- Key exploratory steps:
  - Summary statistics for numerical and categorical features.
  - Visualization of data distributions and correlations using Matplotlib and Seaborn.
  - Identification of missing values in features like Square_Footage, Lot_Size, and Neighborhood_Quality.

2.2 Data Preprocessing

- Handled missing values using imputation techniques (mean for numerical, mode for categorical).
- Scaled numerical features using StandardScaler.
- Encoded categorical features with OneHotEncoder.
- Created new features:
  - Price_Per_Square_Foot: Derived by dividing House_Price by Square_Footage.
  - House_Age: Derived by subtracting Year_Built from the current year (2024).

2.3 Model Training and Evaluation

- Split the dataset into training (80%) and testing (20%) sets.
- Models evaluated:
  - Linear Regression

- Decision Tree Regressor
- Random Forest Regressor

- Metrics used: R2R^2R2, Mean Absolute Error (MAE), Root Mean Squared Error (RMSE).

3. Results and Discussion

3.1 Insights from Data Exploration

- Correlation analysis revealed a strong relationship between Square_Footage and House_Price.
- Scatterplots indicated a positive trend between Lot_Size and House_Price.
- Outliers in numerical columns such as House_Price were identified.

3.2 Model Evaluation Metrics

- Random Forest emerged as the best model with:
  - R2R^2R2: 0.9858
  - MAE: $24,317.75
  - RMSE: $30,251.94

4. Conclusions and Recommendations

4.1 Conclusions

- The project successfully developed a robust model to predict house prices with high accuracy. The Random Forest Regressor performed the best among the models evaluated.

4.2 Recommendations

- Future work could involve additional feature engineering, hyperparameter tuning, and the inclusion of external data sources for enhanced predictions.
- Deploy the model for real-time predictions using web applications.

5. Appendices

a. Python Code for Data Preprocessing

Step 1. Installing Necessary Libraries

 # Install necessary libraries (run this only once)

```
PS E:\python workspace>
PS E:\python workspace>
PS E:\python workspace> pip install pandas numpy scikit-learn matplotlib seaborn
```

Purpose: This command installs libraries needed for data manipulation, visualization, and machine learning.

- pandas: For handling tabular data.
- numpy: For numerical operations.
- scikit-learn: For machine learning.
- matplotlib and seaborn: For creating visualizations.

Note: Use %pip only in Jupyter Notebook. For other editors, run pip install in the terminal.

Step 2. Importing Libraries

 # Importing libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.impute import SimpleImputer

from sklearn.linear_model import LinearRegression

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

```
Requirement already satisfied: pandas in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (2.2.3)
Requirement already satisfied: numpy in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (1.26.4)
Requirement already satisfied: scikit-learn in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (1.5.2)
Requirement already satisfied: matplotlib in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (3.9.2)
Requirement already satisfied: seaborn in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (0.13.2)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\ansar\appdata\roaming\python\python312\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (from pandas) (2024.2)
Requirement already satisfied: scipy>=1.6.0 in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (from matplotlib) (4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (from matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in c:\users\ansar\appdata\roaming\python\python312\site-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (from matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (from matplotlib) (3.2.0)
Requirement already satisfied: six>=1.5 in c:\users\ansar\appdata\local\programs\python\python312\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

Imports essential tools:

- **pandas (pd)**: Handles data in a table format (rows and columns).
- **numpy (np)**: Handles numerical operations.
- **matplotlib.pyplot (plt)** and **seaborn (sns)**: Used for creating graphs and visualizations.
- **sklearn components**:
    - **train_test_split**: Splits data into training and testing sets.

- **StandardScaler and OneHotEncoder**: Standardizes numerical data and converts categories into machine-readable format.
- **Pipeline**: Chains preprocessing steps with a model.
- **SimpleImputer**: Handles missing data by filling it with mean, median, or mode.
- **Machine Learning Models**:
  - LinearRegression: Predicts based on a straight-line relationship.
  - DecisionTreeRegressor: Makes decisions by splitting data into branches.
  - RandomForestRegressor: Combines multiple decision trees for better predictions.
- **Metrics (r2_score, etc.)**: Evaluate model performance.

Step 3. Loading the Dataset

 # Load dataset

data_url = r"E://python workspace//house_price_regression_dataset.csv"

 # Assuming you have downloaded the dataset locally as a CSV file

df = pd.read_csv(data_url)

df.head()

	Square_Footage	Num_Bedrooms	Num_Bathrooms	Year_Built	Lot_Size	Garage_Size	Neighborhood_Quality	House_Price
0	1360	2	1	1981	0.599637	0	5	2.623829e+05
1	4272	3	3	2016	4.753014	1	6	9.852609e+05
2	3592	1	2	2016	3.634823	0	9	7.779774e+05
3	966	1	2	1977	2.730667	1	8	2.296989e+05
4	4926	2	1	1993	4.699073	0	8	1.041741e+06

- **data_url**: Path to the dataset.

- **pd.read_csv()**: Loads the CSV file into a pandas DataFrame (df), making it easier to analyze.
- **df.head()**: Displays the first 5 rows of the dataset to preview its structure.

b. Python Code for Model Evaluation

Step 1. Identifying Numerical and Categorical Columns

```
# Identify columns
numerical_features = ['Square_Footage', 'Lot_Size', 'Year_Built']
categorical_features = ['Neighborhood_Quality']
```

Purpose: Identify the columns in your dataset based on their type:

- Numerical features: Continuous values that can be subjected to mathematical operations (e.g., Square_Footage, Lot_Size, Year Built).
- Categorical features: Non-numerical values representing categories as Qualitative data like "Good" or "Poor" (e.g., Neighborhood_Quality).

Step 2. Data Preprocessing

2.1 Preprocessor for Numerical Features:

```
# Preprocessing pipeline for numerical features
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
```

Pipeline for Numerical Features:

- SimpleImputer(strategy='mean'):
  - Replaces missing values in numerical columns with the mean of the column.
- StandardScaler():
  - Scales numerical data to have a mean of 0 and a standard deviation of 1.
  - Helps improve model performance, especially for algorithms sensitive to feature scaling.

2.2 Preprocessor for Categorical Features:

 # Preprocessing pipeline for categorical features

```
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

Pipeline for Categorical Features:

- **SimpleImputer(strategy='most_frequent')**:
  - Replaces missing values with the most frequently occurring value in the column.
- **OneHotEncoder(handle_unknown='ignore')**:
  - Converts categorical values into one-hot encoded binary columns.
  - Ensures unknown categories during inference are ignored, avoiding errors.

2.3 Combining Transformations:

```
# Combine both into a ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

ColumnTransformer:

- Combines preprocessing steps for both numerical and categorical features.
- **Transformers**:
    - 'num': Applies the numerical_transformer to the numerical_features.
    - 'cat': Applies the categorical_transformer to the categorical_features.

Step 3. Creating New Features

```
# Create price per square foot and age of house
df['Price_Per_Square_Foot'] = df['House_Price'] / df['Square_Footage']
df['House_Age'] = 2024 - df['Year_Built']
# Drop 'Year_Built' as we no longer need it
df = df.drop(columns=['Year_Built'])
```

New Columns:

- **Price per square foot**: Helps understand the price value relative to size.
- **House age**: Converts the Year_Built column into a more useful measure of age.

drop(): Removes unnecessary columns like Year_Built.

Step 4. Exploratory Data Analysis (EDA)

4.1 Heatmap:

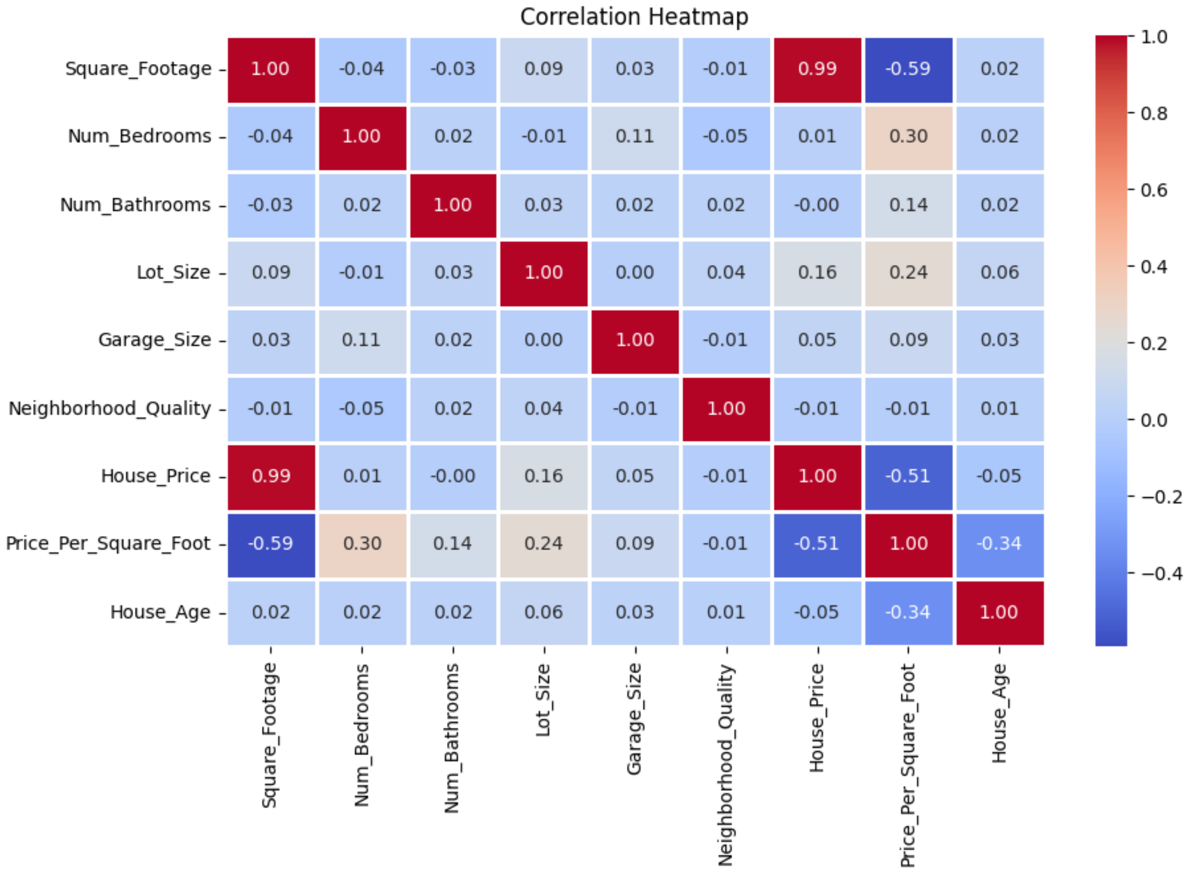
 # Perform Exploratory Data Analysis (EDA)

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

sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=1)

plt.title('Correlation Heatmap')

plt.show()
```



- Displays correlation between numerical features, helping identify relationships.

4.2 Scatterplots:

 # Scatter plots for Price vs Square Footage and Price vs Lot Size

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

sns.scatterplot(x='Square_Footage', y='House_Price', data=df)

plt.title('Price vs Square Footage')

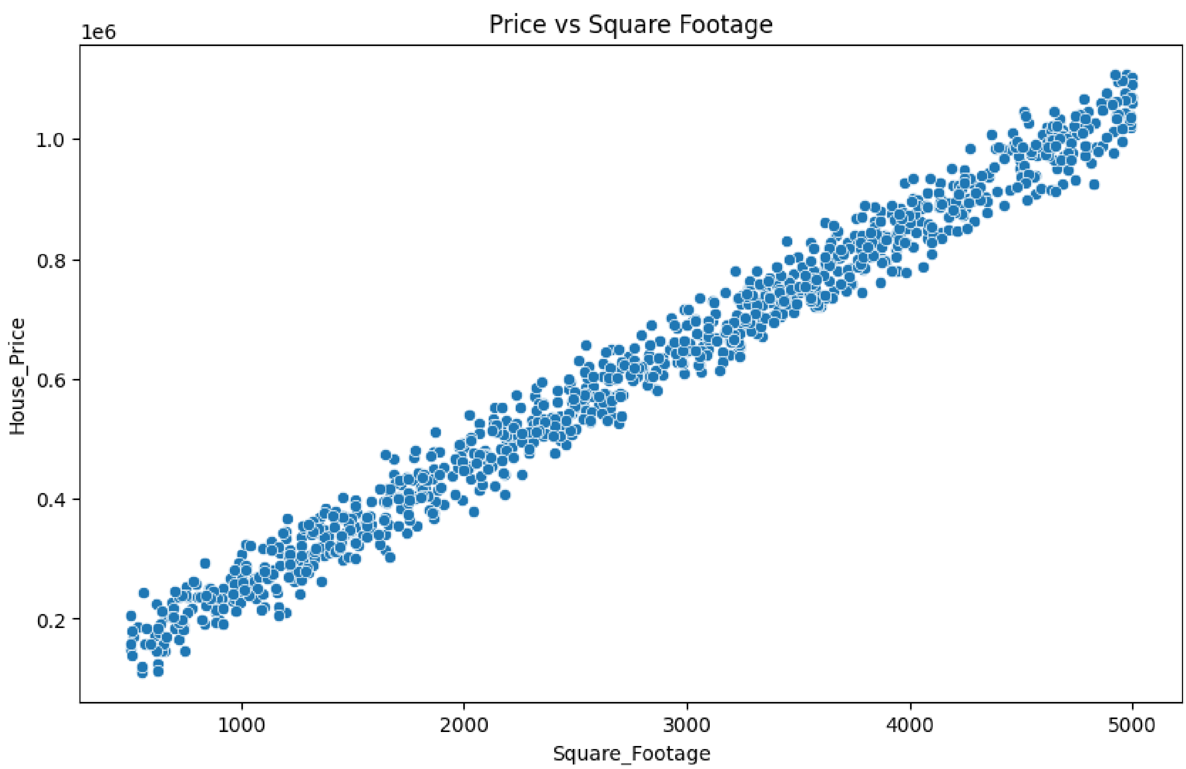
plt.show()

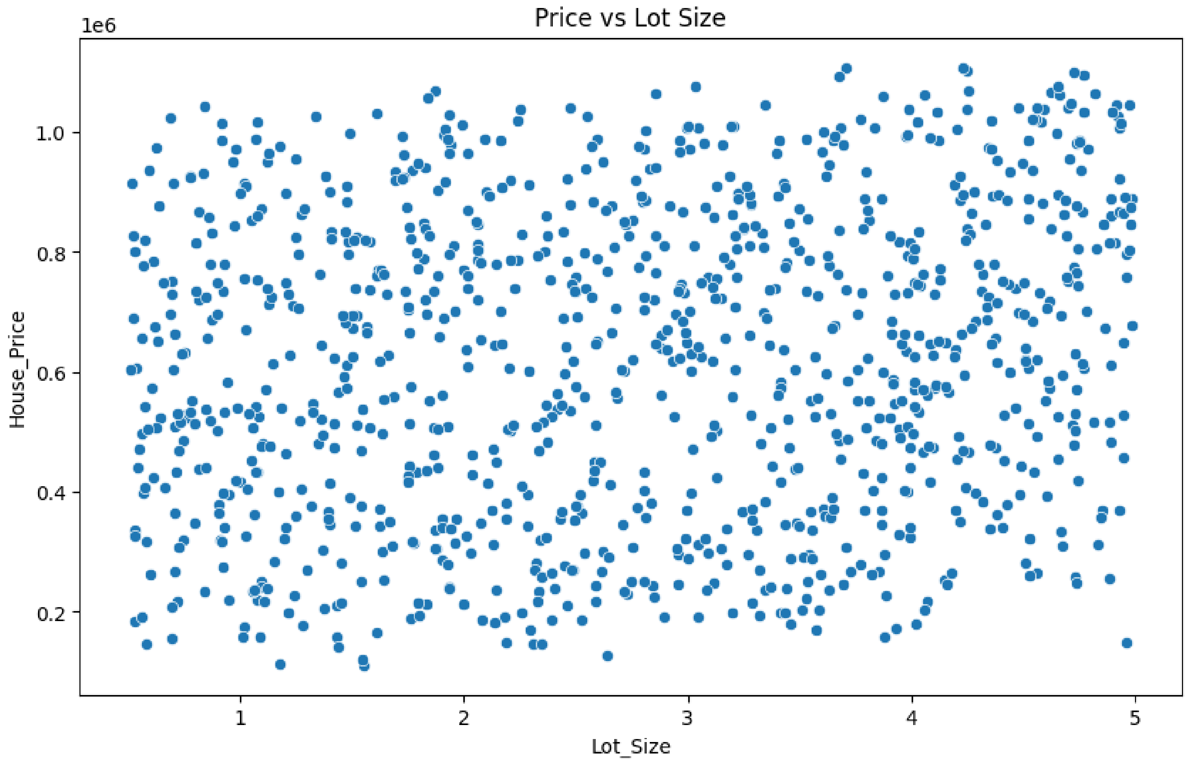
plt.figure(figsize=(10, 6))

sns.scatterplot(x='Lot_Size', y='House_Price', data=df)

plt.title('Price vs Lot Size')

plt.show()





- Visualizes how house price relates to square footage and lot size.
- Shows trends or patterns in the data (e.g., larger houses cost more).

Step 5. Splitting Data for Training and Testing

5.1 Splitting Data:

 # Ensure 'House_Price' is the target variable

target_column = 'House_Price'

 # Split the data into features and target variable

X = df.drop(columns=[target_column])

y = df[target_column]

 # Train-test split (80% training, 20% testing)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

- **X**: Features (input data).
- **y**: Target variable (House_Price).
- **train_test_split**: Splits data into:
  - Training set (80%): For training the model.
  - Testing set (20%): For evaluating the model.

5.2 Feature Identification

 # Identify numerical and categorical columns

numerical_features = ['Square_Footage', 'Lot_Size']

 # Adjust based on your dataset

categorical_features = ['Neighborhood_Quality']

 # Adjust based on your dataset

Purpose: Identify the columns in your dataset based on their type:

- **Numerical features**: Continuous values that can be subjected to mathematical operations (e.g., Square_Footage, Lot_Size).
- **Categorical features**: Non-numerical values representing categories (e.g., Neighborhood_Quality).

5.3 Preprocessor for Numerical Features

 # Define preprocessors for numerical and categorical features

numerical_transformer = Pipeline(steps=[

    ('imputer', SimpleImputer(strategy='mean')),

    ('scaler', StandardScaler())

])

Pipeline for Numerical Features:

- **SimpleImputer(strategy='mean')**:
  - Replaces missing values in numerical columns with the **mean** of the column.
- **StandardScaler()**:
  - Scales numerical data to have a mean of 0 and a standard deviation of 1.
  - Helps improve model performance, especially for algorithms sensitive to feature scaling.

5.4 Preprocessor for Categorical Features

```
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

Pipeline for Categorical Features:

- **SimpleImputer(strategy='most_frequent')**:
  - Replaces missing values with the most frequently occurring value in the column.
- **OneHotEncoder(handle_unknown='ignore')**:
  - Converts categorical values into one-hot encoded binary columns.
  - Ensures unknown categories during inference are ignored, avoiding errors.

5.5 Combine Preprocessing Steps

```
# Combine preprocessing steps into a ColumnTransformer
preprocessor = ColumnTransformer(
```

```
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

ColumnTransformer:

- Combines preprocessing steps for both numerical and categorical features.

Transformers:

  - 'num': Applies the numerical_transformer to the numerical_features.
  - 'cat': Applies the categorical_transformer to the categorical_features.

Step 6. Define Models to Evaluate

6.1 Models:

```
models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(random_state=42),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42)
}
```

Purpose: Creates a dictionary of models to test:

- LinearRegression(): A simple linear model.
- DecisionTreeRegressor(): A decision tree-based regression model.
- RandomForestRegressor(): An ensemble model of decision trees (random forest).

6.2 Model Evaluation

 # Evaluate each model

```
print("\nModel Evaluation:")

for model_name, model in models.items():
```

6.3 Pipeline for Training:

 # Create a pipeline that includes preprocessing and the model

```
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', model)
])
```

 # Train the pipeline

```
pipeline.fit(X_train, y_train)
```

 # Make predictions

```
y_pred = pipeline.predict(X_test)
```

- **Pipeline**:
    - Combines the preprocessing (preprocessor) with the machine learning model.
    - Ensures consistent preprocessing for all models.
- **Training**:
  Fits the pipeline to the training data (X_train, y_train).
- **Prediction**:
- Uses the trained pipeline to predict house prices on the test data (X_test).

6.4 Evaluation Metrics

```python
# Evaluate the model

r2 = r2_score(y_test, y_pred)

mae = mean_absolute_error(y_test, y_pred)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))


# Print evaluation metrics

print(f"\n{model_name}:")

print(f"R-squared: {r2:.4f}")

print(f"Mean Absolute Error (MAE): {mae:.4f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

print("-" * 50)
```

```
...
Model Evaluation:

Linear Regression:
R-squared: 0.9884
Mean Absolute Error (MAE): 22685.1659
Root Mean Squared Error (RMSE): 27365.9912
--------------------------------------------------

Decision Tree:
R-squared: 0.9742
Mean Absolute Error (MAE): 33174.1571
Root Mean Squared Error (RMSE): 40770.3274
--------------------------------------------------

Random Forest:
R-squared: 0.9858
Mean Absolute Error (MAE): 24317.7495
Root Mean Squared Error (RMSE): 30251.9407
--------------------------------------------------
```

- **R-squared (r2)**: Measures how well the model explains the variability of the target variable.
- **Mean Absolute Error (MAE)**: The average absolute difference between actual and predicted values.
- **Root Mean Squared Error (RMSE)**: Penalizes larger prediction errors more heavily than MAE.

Step 7. Feature Importance (for Random Forest)

 # Feature importance for Random Forest

rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

rf_model.fit(preprocessor.fit_transform(X_train), y_train)

 # Get feature importances

importances = rf_model.feature_importances_

- **Random Forest Training**:
  - The RandomForestRegressor is trained separately after transforming the training data with the preprocessor.
- **Feature Importance**:
  - feature_importances_ ranks the importance of each feature in making predictions.

7.1 Combining Feature Names

features = numerical_features + list(preprocessor.transformers_[1][1].named_steps['onehot'].get_feature_names_out(categorical_features))

- **Numerical Features**: Taken directly from numerical_features.

- **Categorical Features**: Extracted from the one-hot encoder after preprocessing.

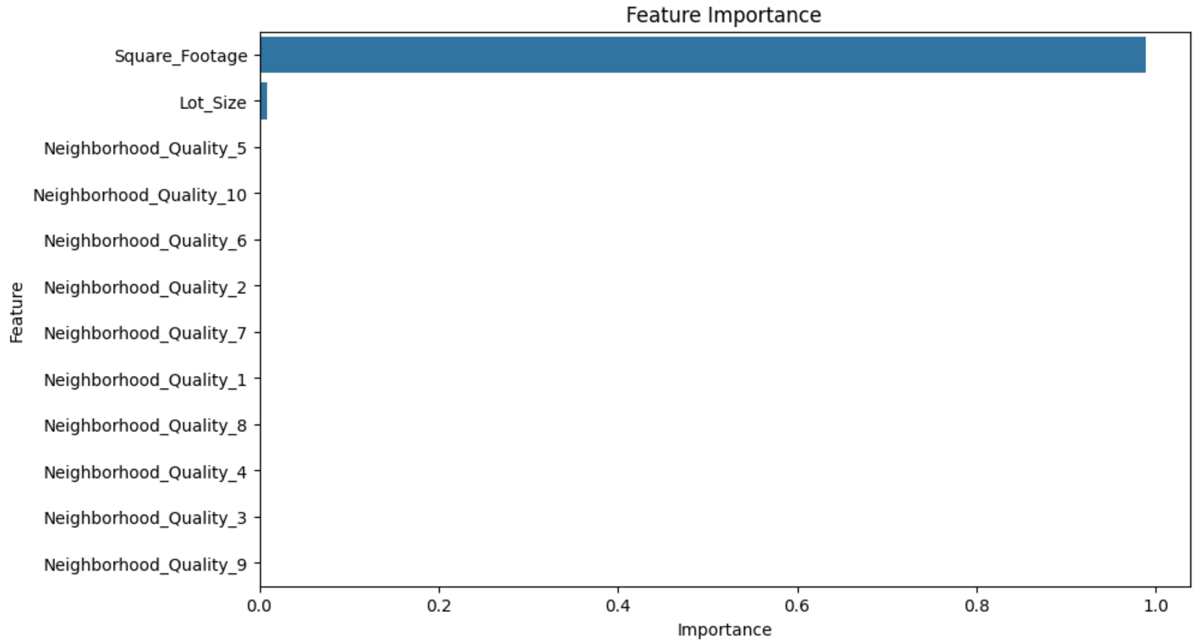
7.2 Visualization

 # Create a dataframe for feature importance

```
feature_importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)
```

 # Plot feature importance

```
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Feature Importance')
plt.show()
```



- **Bar Plot**: Displays the importance of each feature in predicting house prices.

Step 8. Predicting House Price

```
def predict_house_price(sqft, lot_size, neighborhood_quality):
    input_data = pd.DataFrame({
        'Square_Footage': [sqft],
        'Lot_Size': [lot_size],
        'Neighborhood_Quality': [neighborhood_quality]
    })
    predicted_price = pipeline.predict(input_data)
    return predicted_price[0]
```

- **Inputs**:
    - sqft: Square footage of the house.
    - lot_size: Size of the lot.
    - neighborhood_quality: Quality rating of the neighborhood.
- **Pipeline**:
    - Uses the trained pipeline to predict the price based on the input data.
- **Output**:
    - Returns the predicted house price.

8.1 Example Usage

```
# Example usage:
predicted_price = predict_house_price(1000, 4000, 'ok')
```

print(f"Predicted House Price: ${predicted_price:,.2f}")

```
...   Predicted House Price: $257,092.08
```

Predicts the price of a house with:

- 1,000 square feet.
- 4,000 lot size.
- "ok" neighborhood quality.

References

- McKinney, W. Python for Data Analysis. O'Reilly Media, 2017.
- Seaborn Documentation. Retrieved from https://seaborn.pydata.org/.
- scikit-learn Documentation. Retrieved from https://scikit-learn.org/.