AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

23AIE232M PYTHON FOR AI
END SEMESTER REVIEW REPORT

Marks

Faculty In-charge

An AI

Predicting Home Prices with Local Factors (This project has been done as part of the course "Python for AI" (23AIE232M))
Dated:

By:

Manasa J	-	CB.EN.U4AEE23029
Subashini S	-	CB.EN.U4AEE23045
Tanya K	-	CB.EN.U4AEE23055
V Dheeraj	-	CB.EN.U4AEE23060

CONTENTS

Abstract:

This report presents an AI model for predicting house prices using machine learning regression techniques. By analyzing historical real estate data, the model identifies key factors such as area, number of bedrooms, number of stories, parking availability, and other property features to estimate market values. Performance evaluation is based on kurtosis, standard deviation, and mean, providing insights into price distribution and variability. Results show improved predictions over traditional methods, making it a valuable tool for real estate professionals and buyers. The report also discusses challenges like data preprocessing and feature selection, along with potential enhancements for greater accuracy.

Introduction:

-Background and motivation

This project is motivated by the need to leverage data-driven approaches to predict house prices using key features. By utilizing machine learning techniques, we aim to improve the efficiency of property valuation, benefiting buyers, sellers, and real estate professionals alike.

-Problem statement and objective:

Create a system that predicts house prices based on features like location, size, number of bedrooms, and age of the house.

-Scope:

This project developed a predictive model that scales effectively with larger datasets, enhancing real-world applicability. While linear regression served as the foundation for this particular project, potential improvements could explore advanced machine learning techniques for greater accuracy.

Literature review:

Summary of existing work related to the project:

1)Zillow Zestimate (Zillow, Inc.)

Zillow, one of the largest online real estate marketplaces, developed the Zestimate tool, which uses machine learning algorithms to predict house prices.

Features Used: Property characteristics (size, number of bedrooms/bathrooms)

2)Hedonic Pricing Models (Traditional Economic Models)

These models estimate house prices based on the principle that property value is derived from the sum of its individual attributes (e.g., location, size, amenities).

3)How our project differs:

Uses a large-scale dataset with thousands of house details for more accurate predictions.

Implements multiple machine learning models and compares their performance.

Aims for a scalable solution that can be applied across different real estate markets.

Technologies and Libraries used:

1)Python Basics

- Variables: Various features of houses such as Price, SquareFootage, Bedrooms, etc., are stored in a pandas DataFrame.

- Control Structures: for loop is used to iterate through different train-test split ratios and evaluate the model.

- Collections: The dataset is stored as a pandas DataFrame, and a dictionary (mse_results) is used to store the mean squared error results.

2)Database Integration

- SQL connectivity: Importing an SQL database to python.

- CRUD Operations: The dataset manipulation mimics CRUD (Create, Read, Update, and Delete) operations—data is created (DataFrame initialization), read (.corr(), .var(), .std()), updated (feature selection), and deleted (dropping columns before model training).

3)Libraries Used

- NumPy: Used for numerical computations (implicitly through pandas and Scikit-learn operations).

- Pandas: Used to handle and manipulate the dataset (DataFrame creation, feature selection, and statistical analysis).

- Matplotlib: Used to visualize data through correlation heatmaps and boxplots.

- Seaborn: Used for enhanced visualization (complements Matplotlib for heatmaps and boxplots).

- Scikit-learn: Used for machine learning tasks:
  - train_test_split for splitting data into training and testing sets.
  - LinearRegression for building and training the model.
  - mean_squared_error for evaluating model performance.

Methodology:

1)Data collection:

Data Source (CSV, SQL database, Web scraping, API, etc.)

Data Cleaning & Preprocessing (Handling missing values, transformations, etc.)

2)Exploratory Data Analysis (EDA):

Initial insights from data using Pandas & Seaborn

Visualization (Histograms, Boxplots, Scatterplots)

3)Feature Engineering:

Feature selection, transformation, and scaling using NumPy & Pandas

4)Model Building:

Machine Learning models using Scikit-learn

Training and Evaluation Metrics

Completed tasks

1) Successfully imported the dataset.

2) Detected and removed outliers using boxplots.

3) Calculated statistical measures, including kurtosis, for data analysis.

4) Generated heatmaps based on statistical insights.

5) Identified key features with the greatest impact on house prices.

6) Trained the predictive model using linear regression.

7) Evaluated model performance using statistical metrics.

8) Ensured the model scales effectively with larger datasets for real-world applicability.

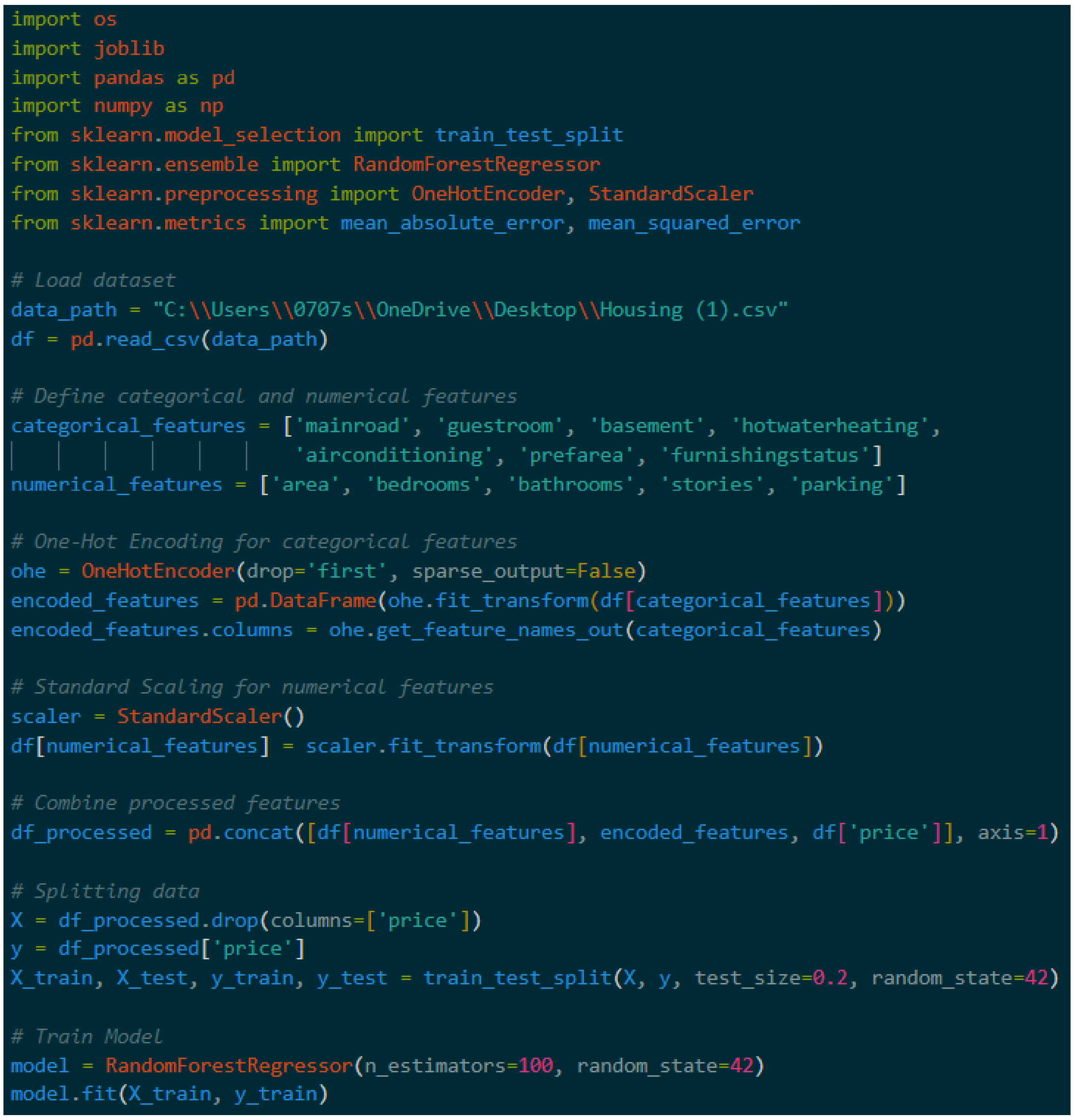
9) Created a structured GitHub repository.

10) Followed PEP8 coding standards for maintainability and readability.

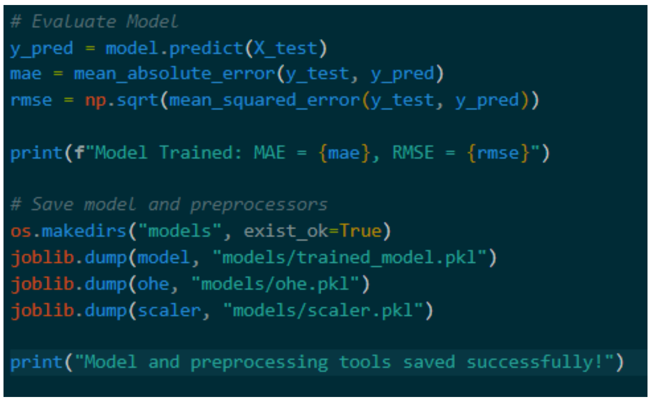
Result:

Our final code:

Code from model_train.py:

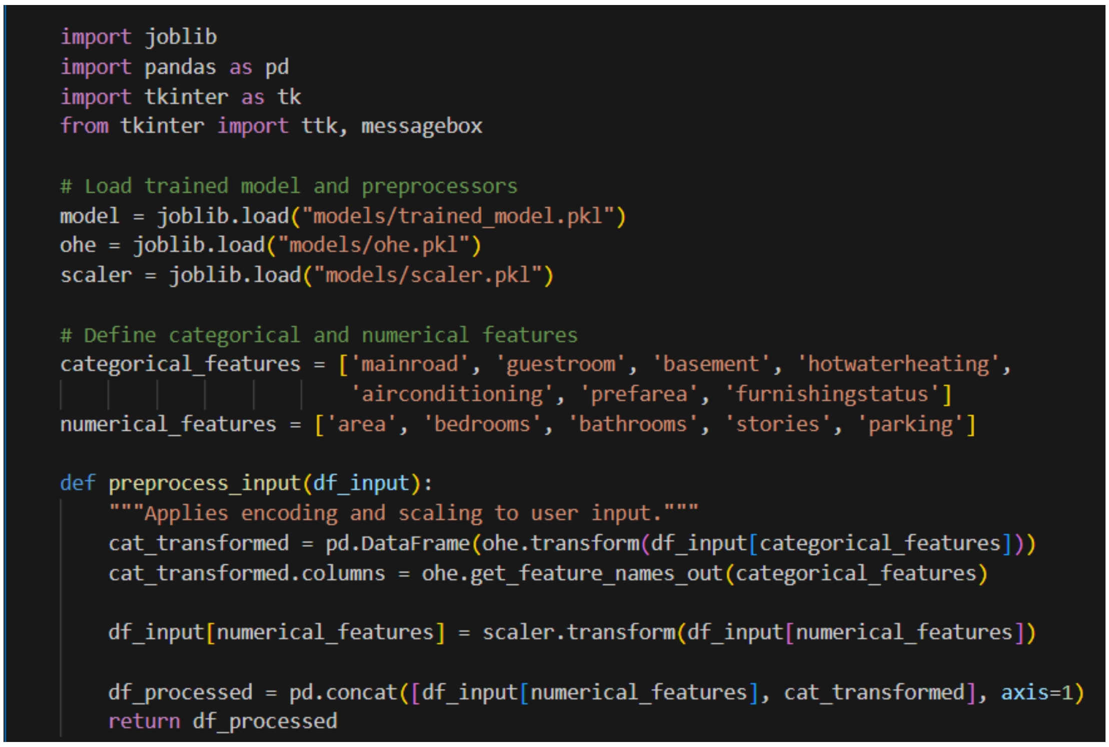


```
import os
import joblib
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Load dataset
data_path = "C:\\Users\\0707s\\OneDrive\\Desktop\\Housing (1).csv"
df = pd.read_csv(data_path)

# Define categorical and numerical features
categorical_features = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',
                        'airconditioning', 'prefarea', 'furnishingstatus']
numerical_features = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']

# One-Hot Encoding for categorical features
ohe = OneHotEncoder(drop='first', sparse_output=False)
encoded_features = pd.DataFrame(ohe.fit_transform(df[categorical_features]))
encoded_features.columns = ohe.get_feature_names_out(categorical_features)

# Standard Scaling for numerical features
scaler = StandardScaler()
df[numerical_features] = scaler.fit_transform(df[numerical_features])

# Combine processed features
df_processed = pd.concat([df[numerical_features], encoded_features, df['price']], axis=1)

# Splitting data
X = df_processed.drop(columns=['price'])
y = df_processed['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

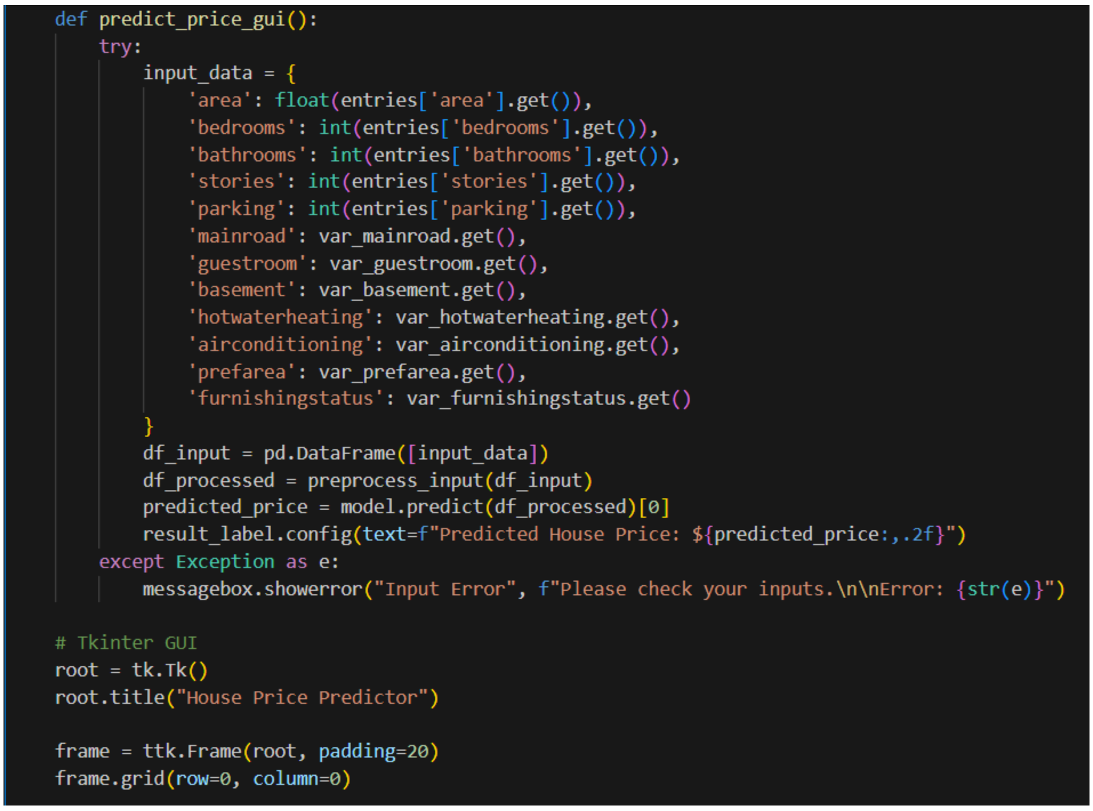


```
# Evaluate Model
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"Model Trained: MAE = {mae}, RMSE = {rmse}")

# Save model and preprocessors
os.makedirs("models", exist_ok=True)
joblib.dump(model, "models/trained_model.pkl")
joblib.dump(ohe, "models/ohe.pkl")
joblib.dump(scaler, "models/scaler.pkl")

print("Model and preprocessing tools saved successfully!")
```

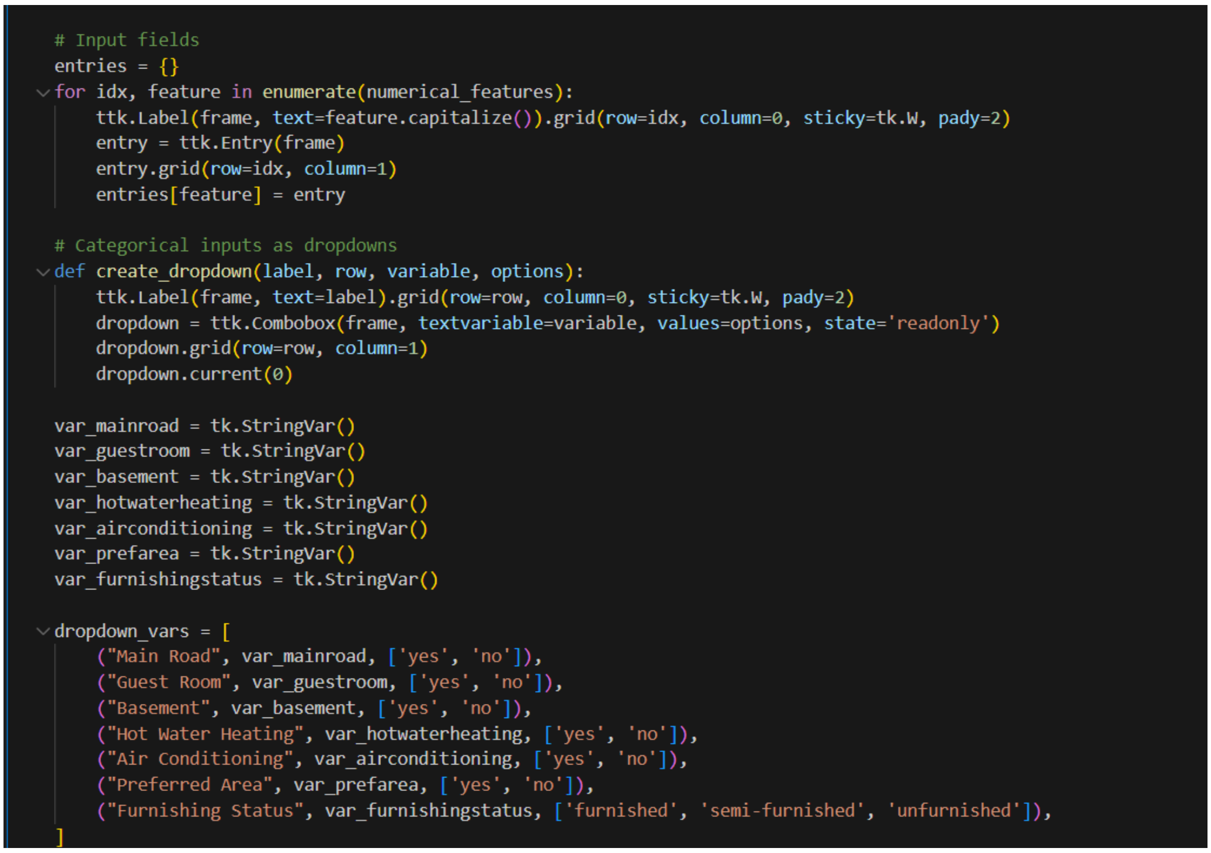
Code from predict_ui.py:



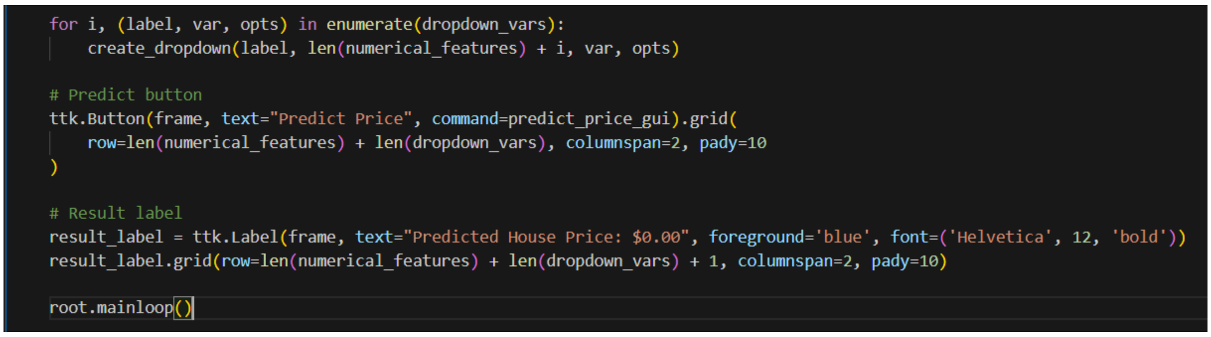
```python
import joblib
import pandas as pd
import tkinter as tk
from tkinter import ttk, messagebox

# Load trained model and preprocessors
model = joblib.load("models/trained_model.pkl")
ohe = joblib.load("models/ohe.pkl")
scaler = joblib.load("models/scaler.pkl")

# Define categorical and numerical features
categorical_features = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',
                        'airconditioning', 'prefarea', 'furnishingstatus']
numerical_features = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']

def preprocess_input(df_input):
    """Applies encoding and scaling to user input."""
    cat_transformed = pd.DataFrame(ohe.transform(df_input[categorical_features]))
    cat_transformed.columns = ohe.get_feature_names_out(categorical_features)

    df_input[numerical_features] = scaler.transform(df_input[numerical_features])

    df_processed = pd.concat([df_input[numerical_features], cat_transformed], axis=1)
    return df_processed
```



```python
def predict_price_gui():
    try:
        input_data = {
            'area': float(entries['area'].get()),
            'bedrooms': int(entries['bedrooms'].get()),
            'bathrooms': int(entries['bathrooms'].get()),
            'stories': int(entries['stories'].get()),
            'parking': int(entries['parking'].get()),
            'mainroad': var_mainroad.get(),
            'guestroom': var_guestroom.get(),
            'basement': var_basement.get(),
            'hotwaterheating': var_hotwaterheating.get(),
            'airconditioning': var_airconditioning.get(),
            'prefarea': var_prefarea.get(),
            'furnishingstatus': var_furnishingstatus.get()
        }
        df_input = pd.DataFrame([input_data])
        df_processed = preprocess_input(df_input)
        predicted_price = model.predict(df_processed)[0]
        result_label.config(text=f"Predicted House Price: ${predicted_price:,.2f}")
    except Exception as e:
        messagebox.showerror("Input Error", f"Please check your inputs.\n\nError: {str(e)}")

# Tkinter GUI
root = tk.Tk()
root.title("House Price Predictor")

frame = ttk.Frame(root, padding=20)
frame.grid(row=0, column=0)
```



```python
# Input fields
entries = {}
for idx, feature in enumerate(numerical_features):
    ttk.Label(frame, text=feature.capitalize()).grid(row=idx, column=0, sticky=tk.W, pady=2)
    entry = ttk.Entry(frame)
    entry.grid(row=idx, column=1)
    entries[feature] = entry

# Categorical inputs as dropdowns
def create_dropdown(label, row, variable, options):
    ttk.Label(frame, text=label).grid(row=row, column=0, sticky=tk.W, pady=2)
    dropdown = ttk.Combobox(frame, textvariable=variable, values=options, state='readonly')
    dropdown.grid(row=row, column=1)
    dropdown.current(0)

var_mainroad = tk.StringVar()
var_guestroom = tk.StringVar()
var_basement = tk.StringVar()
var_hotwaterheating = tk.StringVar()
var_airconditioning = tk.StringVar()
var_prefarea = tk.StringVar()
var_furnishingstatus = tk.StringVar()

dropdown_vars = [
    ("Main Road", var_mainroad, ['yes', 'no']),
    ("Guest Room", var_guestroom, ['yes', 'no']),
    ("Basement", var_basement, ['yes', 'no']),
    ("Hot Water Heating", var_hotwaterheating, ['yes', 'no']),
    ("Air Conditioning", var_airconditioning, ['yes', 'no']),
    ("Preferred Area", var_prefarea, ['yes', 'no']),
    ("Furnishing Status", var_furnishingstatus, ['furnished', 'semi-furnished', 'unfurnished']),
]
```



```python
for i, (label, var, opts) in enumerate(dropdown_vars):
    create_dropdown(label, len(numerical_features) + i, var, opts)

# Predict button
ttk.Button(frame, text="Predict Price", command=predict_price_gui).grid(
    row=len(numerical_features) + len(dropdown_vars), columnspan=2, pady=10
)

# Result label
result_label = ttk.Label(frame, text="Predicted House Price: $0.00", foreground='blue', font=('Helvetica', 12, 'bold'))
result_label.grid(row=len(numerical_features) + len(dropdown_vars) + 1, columnspan=2, pady=10)

root.mainloop()
```

1. Module and Model Loading

- The script begins by importing essential libraries: joblib, pandas, and tkinter, among others.
- It loads the pre-trained model using joblib.load("models/trained_model.pkl").
- Additionally, it loads the associated OneHotEncoder and StandardScaler from disk to preprocess new input data.

2. Feature Categories

- The features are split into two types:
  - **Numerical features**: 'area', 'bedrooms', 'bathrooms', 'stories', 'parking'
  - **Categorical features**: 'mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea', 'furnishingstatus'

---

3. Preprocessing Function

- A function preprocess_input() is defined to handle input transformation:
    - Uses ohe.transform() to convert categorical variables into one-hot encoded format.
    - Uses scaler.transform() to standardize numerical features.
    - Combines the processed features using pd.concat() for final prediction.

---

4. Prediction Function

- predict_price_gui() is triggered when the user clicks the prediction button.
    - Reads user inputs from the form fields.
    - Converts them into a DataFrame.
    - Passes the DataFrame to preprocess_input() for transformation.
    - Predicts the house price using model.predict() and updates the GUI label with the result.
    - If inputs are invalid, it shows an error message using messagebox.showerror().

---

5. GUI Layout with Tkinter

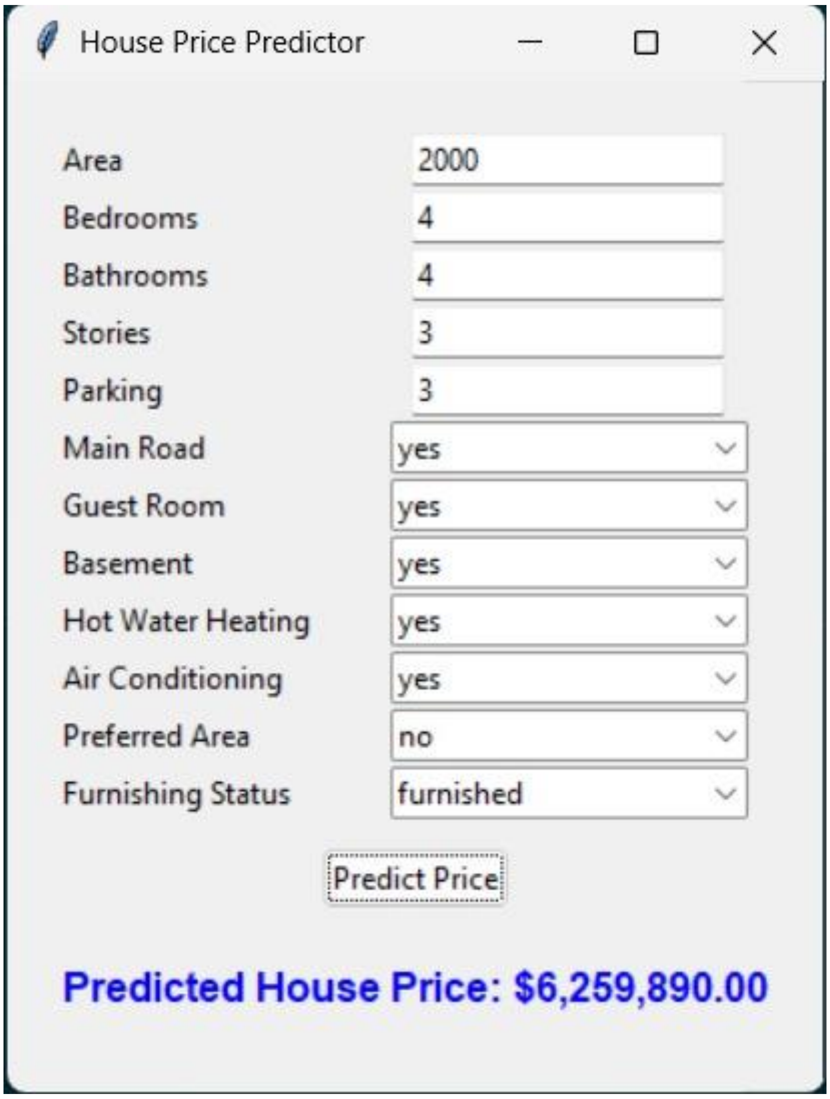
- The interface is created using tk.Tk() and ttk.Frame().
- Input fields are generated for each numerical feature using ttk.Label and ttk.Entry.
- Dropdowns for categorical features are implemented using ttk.Combobox, with predefined options such as:
    - 'yes' / 'no' for binary features
    - 'furnished', 'semi-furnished', 'unfurnished' for furnishingstatus
- A **"Predict Price"** button is created with ttk.Button() to trigger the prediction.
- The result is displayed below using ttk.Label, styled in blue and bold.

---

6. Launching the App

- The final line root.mainloop() starts the Tkinter event loop, keeping the window active for user interaction.

Output:



Area	2000
Bedrooms	4
Bathrooms	4
Stories	3
Parking	3
Main Road	yes
Guest Room	yes
Basement	yes
Hot Water Heating	yes
Air Conditioning	yes
Preferred Area	no
Furnishing Status	furnished

Predicted House Price: $6,259,890.00

Challenges Faced and Risks Involved

Challenges Faced:

Technical Issues:

- Hardware Limitations: Due to the sheer size of the dataset, computing statistical measures was resource-intensive, leading to frequent device resets.
- Computational Resources: Processing large datasets and training models required sufficient computing power, which was sometimes a limitation.

Data Limitations:

- Redundant and Inconsistent Data: The dataset contained redundant and inconsistent data, which required thorough preprocessing using boxplots.

- Skewed Distributions and Missing Data: Certain features had skewed distributions and missing values, affecting model performance and requiring imputation or transformation.
- Feature Selection Complexity: Identifying the most impactful features for price prediction required careful statistical analysis.

Model Performance and Scalability:

- Ensuring Accuracy: The linear regression model needed to be optimized to provide accurate predictions across different datasets.
- Scalability Challenges: The model needed to be adaptable to handle larger datasets efficiently without performance degradation.

Time Constraints:

- Feature Selection and Model Tuning: Extensive experimentation is needed to optimize the model, but project deadlines limited the time available.
- Data Preprocessing Delays: Exploratory Data Analysis (EDA) and data cleaning took significant time, delaying model training and optimization.

Code and Repository Management:

- Code Maintainability: Enforcing PEP8 coding standards and structuring the repository for long-term usability and collaboration.
- Repository Organization: Ensuring a well-structured GitHub repository for easy access and reproducibility.

Risks Involved:

- Overfitting or Underfitting: The model may not generalize well to new data, impacting prediction accuracy.
- Bias in Data: If the dataset lacks diversity, predictions may be skewed or unreliable.
- Market Dynamics: Real estate prices fluctuate due to economic changes, policy shifts, and market trends, which the model may not fully capture.
- Data Privacy Concerns: Handling real estate data requires caution to ensure compliance with data protection regulations.
- Dependency on Historical Data: The model relies on past trends, which may not always reflect future price movements accurately.

Conclusion:

This project successfully developed a predictive AI model for house price estimation, leveraging statistical analysis and machine learning techniques. Through rigorous data preprocessing, feature selection, and model training, the model was optimized to scale effectively with larger datasets, enhancing its real-world applicability. The use of statistical measures such as kurtosis, standard deviation, and mean provided valuable insights into price distribution and variability.

Despite challenges such as hardware limitations, data inconsistencies, and time constraints, the structured approach to data cleaning, exploratory analysis, and model evaluation ensured reliable performance. While linear regression served as the foundation for this project, potential improvements could explore advanced machine learning algorithms for greater accuracy and robustness.

By following best coding practices and organizing a structured GitHub repository, the project is well-documented and maintainable for future enhancements. Overall, this work highlights the potential of AI-driven house price prediction as a valuable tool for real estate professionals, investors, and buyers seeking data-driven insights.

Datasets:

House pricing dataset collected from publicly available real estate data sources.

Tools Used:

Python 3.x, Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn.

SQL database management systems (MySQL/PostgreSQL) for potential large-scale data handling.

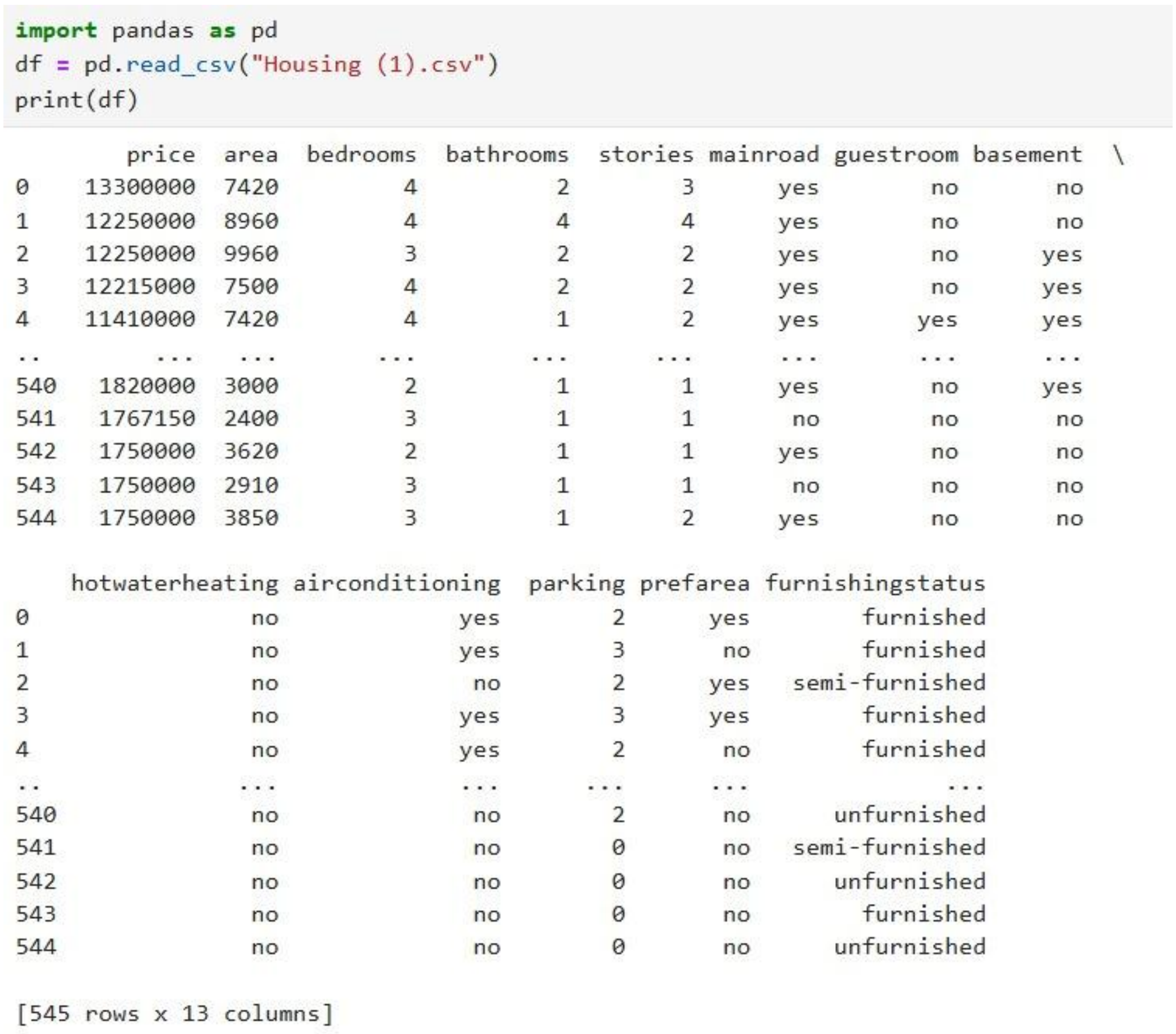
References:

https://www.kaggle.com/datasets/yasserh/housing-prices-dataset/data

Appendix:

Part one: The Data Set

We have imported, read and printed a dataset from sql:



```python
import pandas as pd
df = pd.read_csv("Housing (1).csv")
print(df)
```

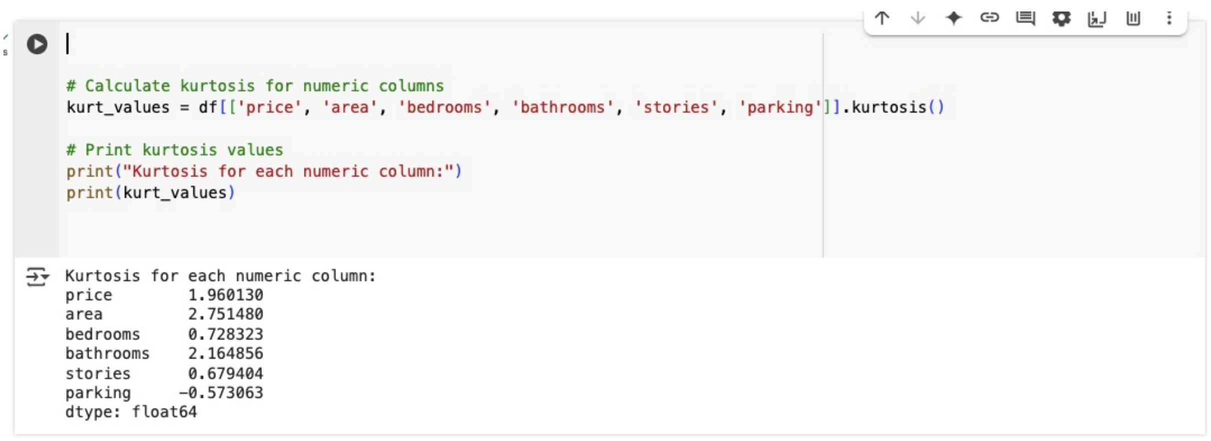
```
        price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0    13300000  7420         4          2        3      yes        no       no
1    12250000  8960         4          4        4      yes        no       no
2    12250000  9960         3          2        2      yes        no      yes
3    12215000  7500         4          2        2      yes        no      yes
4    11410000  7420         4          1        2      yes       yes      yes
..        ...   ...       ...        ...      ...      ...       ...      ...
540   1820000  3000         2          1        1      yes        no      yes
541   1767150  2400         3          1        1       no        no       no
542   1750000  3620         2          1        1      yes        no       no
543   1750000  2910         3          1        1       no        no       no
544   1750000  3850         3          1        2      yes        no       no

    hotwaterheating airconditioning  parking prefarea furnishingstatus
0                no             yes        2      yes        furnished
1                no             yes        3       no        furnished
2                no              no        2      yes   semi-furnished
3                no             yes        3      yes        furnished
4                no             yes        2       no        furnished
..              ...             ...      ...      ...              ...
540              no              no        2       no      unfurnished
541              no              no        0       no   semi-furnished
542              no              no        0       no      unfurnished
543              no              no        0       no        furnished
544              no              no        0       no      unfurnished

[545 rows x 13 columns]
```

Part two: Data Analysis

The first thing we did with this data is to calculate kurtosis, Variance and Standard deviation for each and every feature. In order to do this, we took the help of python.

Code and output:

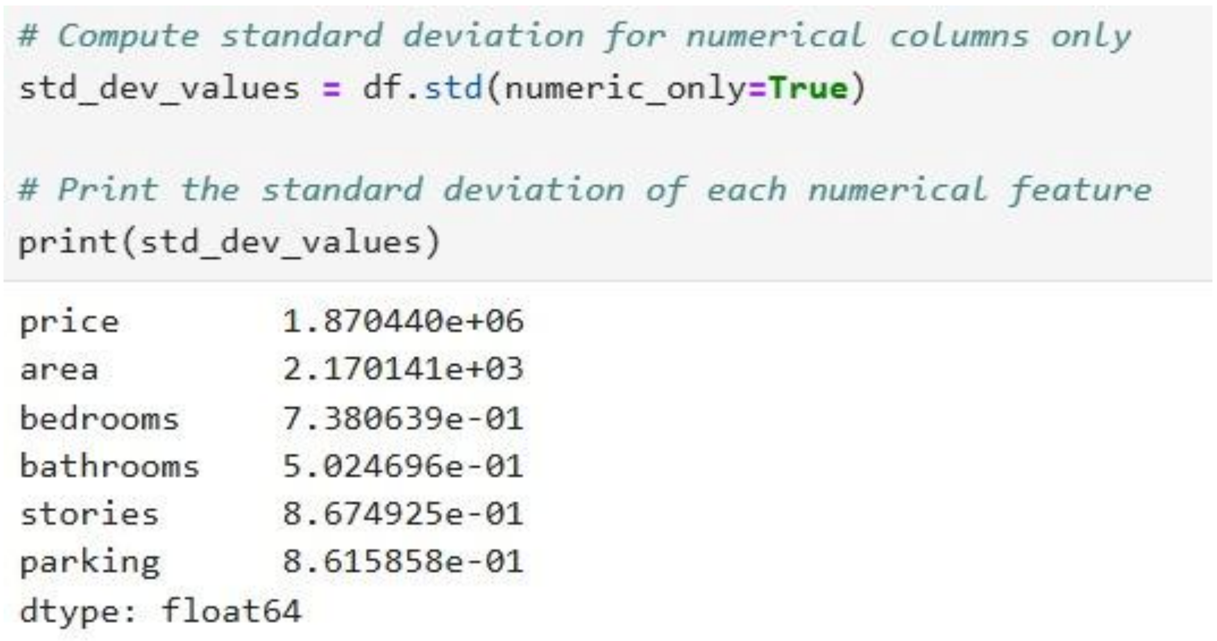
Kurtosis:



```
# Calculate kurtosis for numeric columns
kurt_values = df[['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']].kurtosis()

# Print kurtosis values
print("Kurtosis for each numeric column:")
print(kurt_values)
```

```
Kurtosis for each numeric column:
price        1.960130
area         2.751480
bedrooms     0.728323
bathrooms    2.164856
stories      0.679404
parking     -0.573063
dtype: float64
```

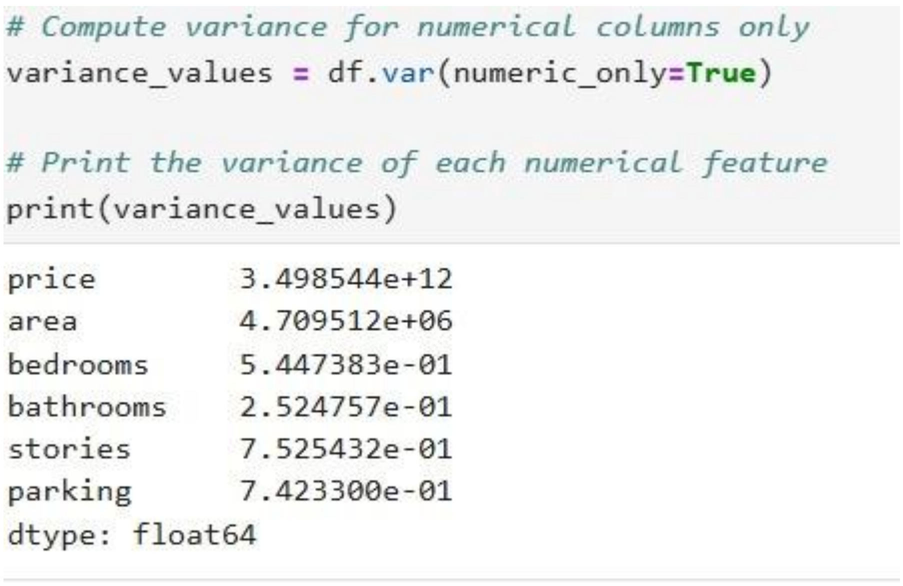
Standard deviation:



```
# Compute standard deviation for numerical columns only
std_dev_values = df.std(numeric_only=True)

# Print the standard deviation of each numerical feature
print(std_dev_values)
```

```
price        1.870440e+06
area         2.170141e+03
bedrooms     7.380639e-01
bathrooms    5.024696e-01
stories      8.674925e-01
parking      8.615858e-01
dtype: float64
```

Variance:



```
# Compute variance for numerical columns only
variance_values = df.var(numeric_only=True)

# Print the variance of each numerical feature
print(variance_values)
```

```
price        3.498544e+12
area         4.709512e+06
bedrooms     5.447383e-01
bathrooms    2.524757e-01
stories      7.525432e-01
parking      7.423300e-01
dtype: float64
```

Analysis based on standard deviation and variance:

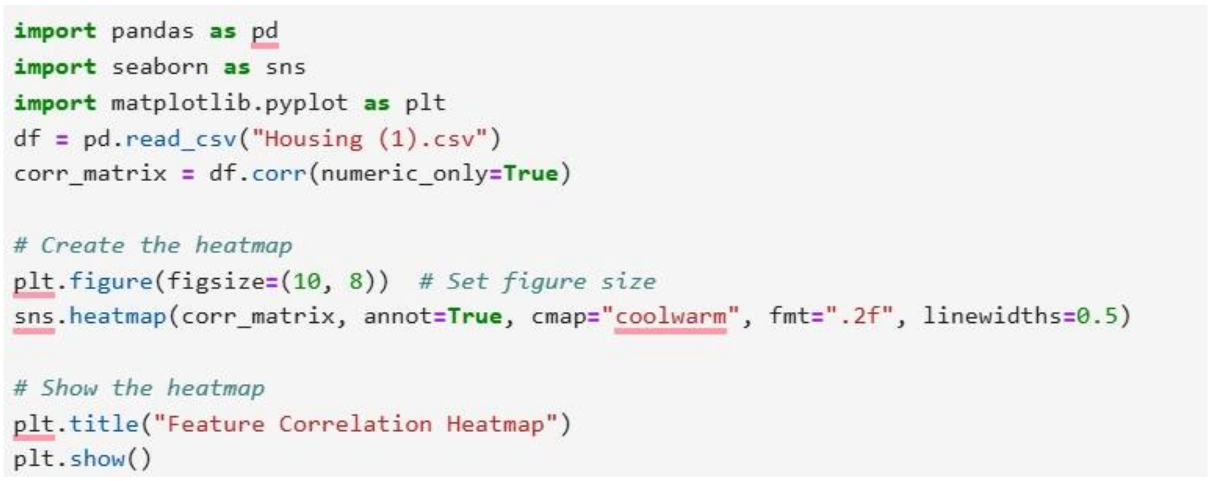
1. A high variance and standard deviation for Price indicate that house prices vary widely in the dataset.

2. Features like Bedrooms and Bathrooms likely have low variance and standard deviation, indicating that their values are more consistent.

Analysis based on Kurtosis:

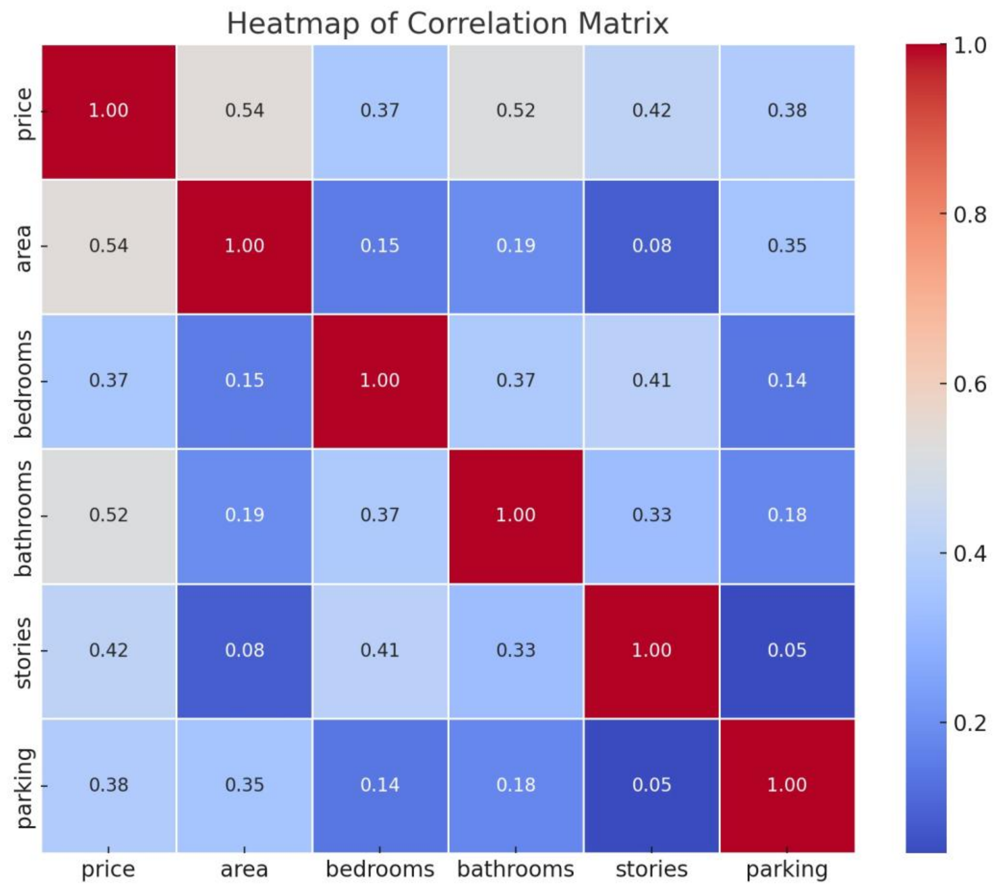
1. If Price has positive kurtosis, this suggests that there may be a few extreme outliers in the pricing data (e.g., very expensive or very cheap houses).

Code to plot a heatmap:



```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv("Housing (1).csv")
corr_matrix = df.corr(numeric_only=True)

# Create the heatmap
plt.figure(figsize=(10, 8))  # Set figure size
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

# Show the heatmap
plt.title("Feature Correlation Heatmap")
plt.show()
```

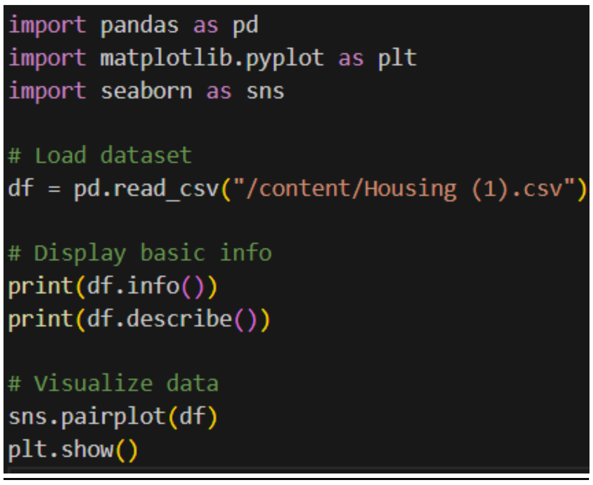
Heatmap:


Heatmap of Correlation Matrix
price
area
bedrooms
bathrooms
stories
parking
1.00
0.54
0.37
0.52
0.42
0.38
0.54
1.00
0.15
0.19
0.08
0.35
0.37
0.15
1.00
0.37
0.41
0.14
0.52
0.19
0.37
1.00
0.33
0.18
0.42
0.08
0.41
0.33
1.00
0.05
0.38
0.35
0.14
0.18
0.05
1.00
price
area
bedrooms
bathrooms
stories
parking
1.0
0.8
0.6
0.4
0.2

Part three: Working with the dataset

Codes, explainations and outputs:

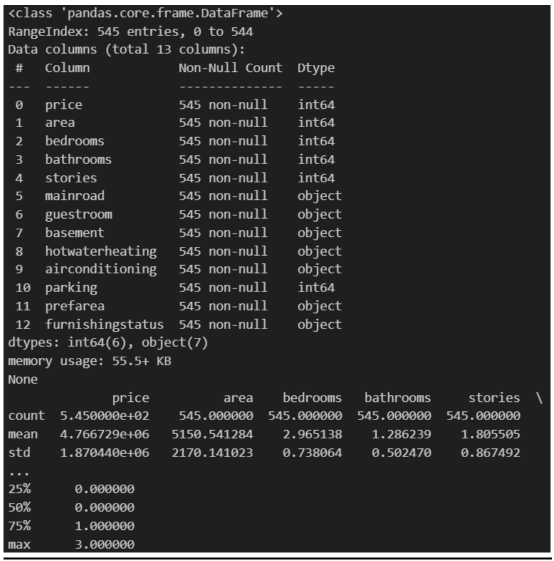
Code:



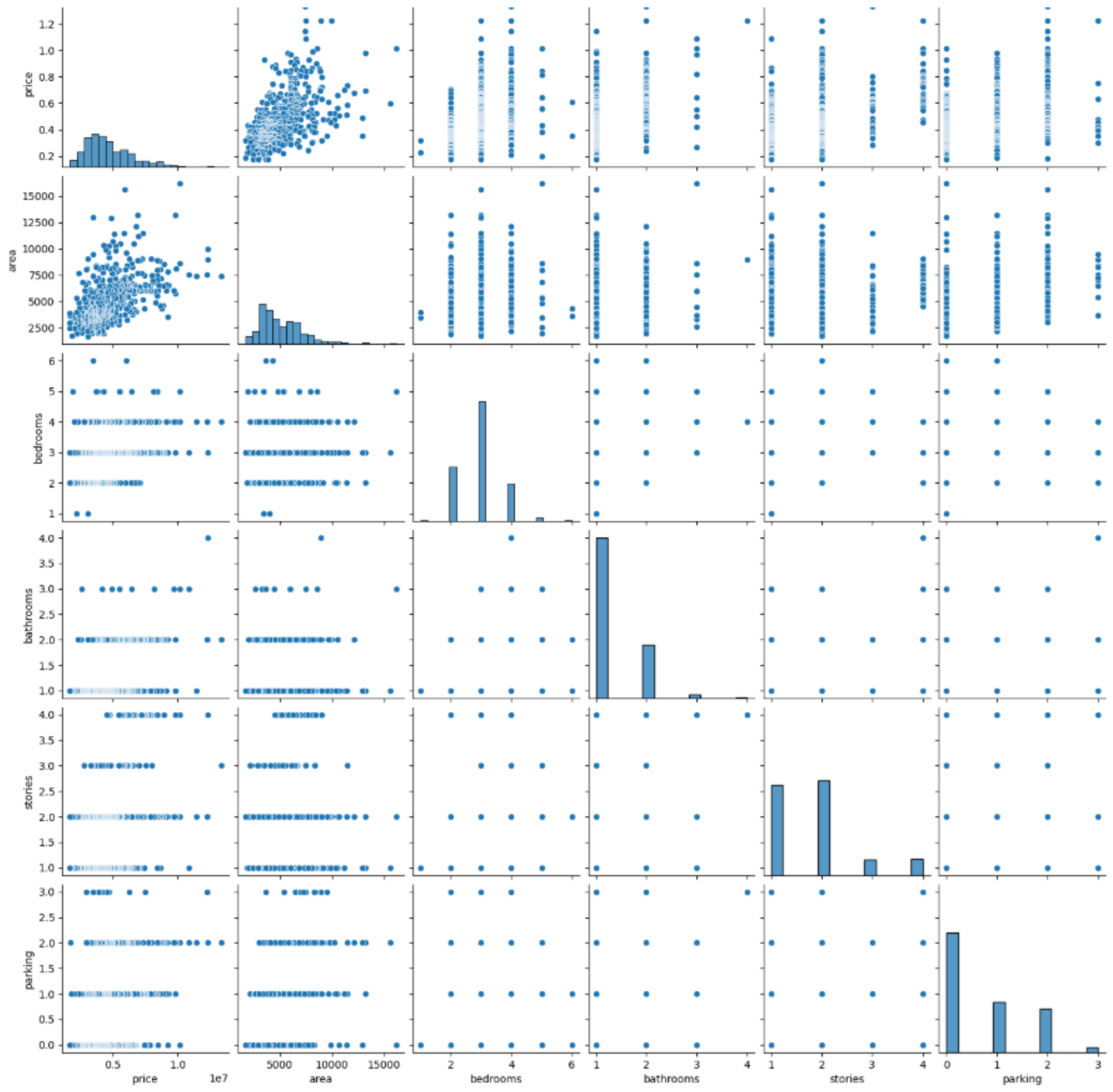
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
df = pd.read_csv("/content/Housing (1).csv")

# Display basic info
print(df.info())
print(df.describe())

# Visualize data
sns.pairplot(df)
plt.show()
```

This code represents the first step in analyzing the housing dataset. It loads the data using Pandas and performs an initial exploratory data analysis. Basic information about the dataset, including column types and summary statistics, is displayed to understand its structure. Finally, a Seaborn `pairplot` is generated to visualize relationships between numerical variables, helping to identify trends, correlations, and potential outliers.

Output:



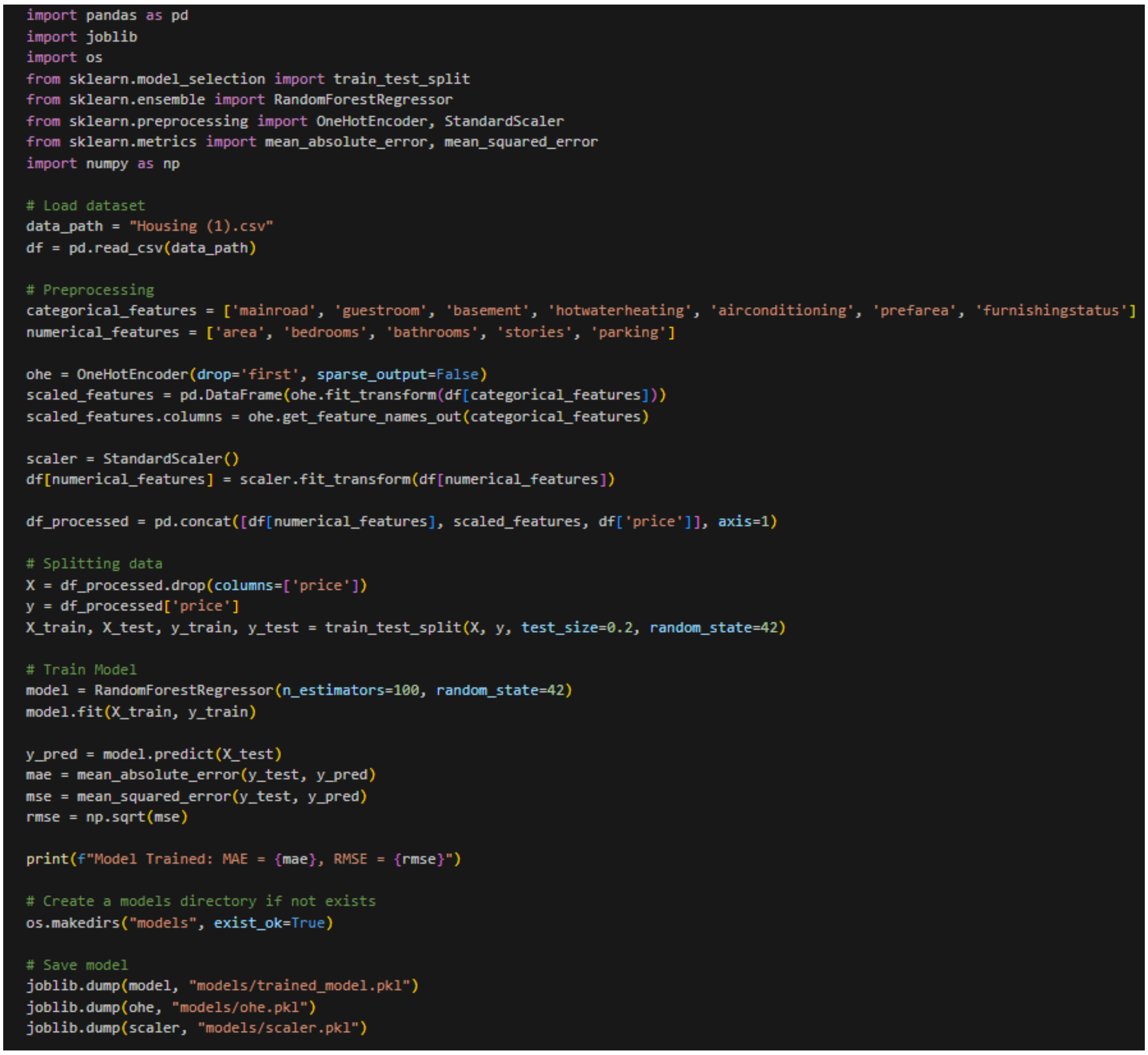
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
None
              price          area    bedrooms   bathrooms     stories  \
count  5.450000e+02    545.000000  545.000000  545.000000  545.000000
mean   4.766729e+06   5150.541284    2.965138    1.286239    1.805505
std    1.870440e+06   2170.141023    0.738064    0.502470    0.867492
...
25%      0.000000
50%      0.000000
75%      1.000000
max      3.000000
```



Pairplot visualization showing relationships between numerical features in the housing dataset

Code:

Code from model_train.py:



```
import pandas as pd
import joblib
import os
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Load dataset
data_path = "Housing (1).csv"
df = pd.read_csv(data_path)

# Preprocessing
categorical_features = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea', 'furnishingstatus']
numerical_features = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']

ohe = OneHotEncoder(drop='first', sparse_output=False)
scaled_features = pd.DataFrame(ohe.fit_transform(df[categorical_features]))
scaled_features.columns = ohe.get_feature_names_out(categorical_features)

scaler = StandardScaler()
df[numerical_features] = scaler.fit_transform(df[numerical_features])

df_processed = pd.concat([df[numerical_features], scaled_features, df['price']], axis=1)

# Splitting data
X = df_processed.drop(columns=['price'])
y = df_processed['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print(f"Model Trained: MAE = {mae}, RMSE = {rmse}")

# Create a models directory if not exists
os.makedirs("models", exist_ok=True)

# Save model
joblib.dump(model, "models/trained_model.pkl")
joblib.dump(ohe, "models/ohe.pkl")
joblib.dump(scaler, "models/scaler.pkl")
```

This code preprocesses a housing dataset and trains a **Random Forest Regressor** (an ensemble learning model that combines multiple decision trees for better accuracy and stability) to predict house prices.

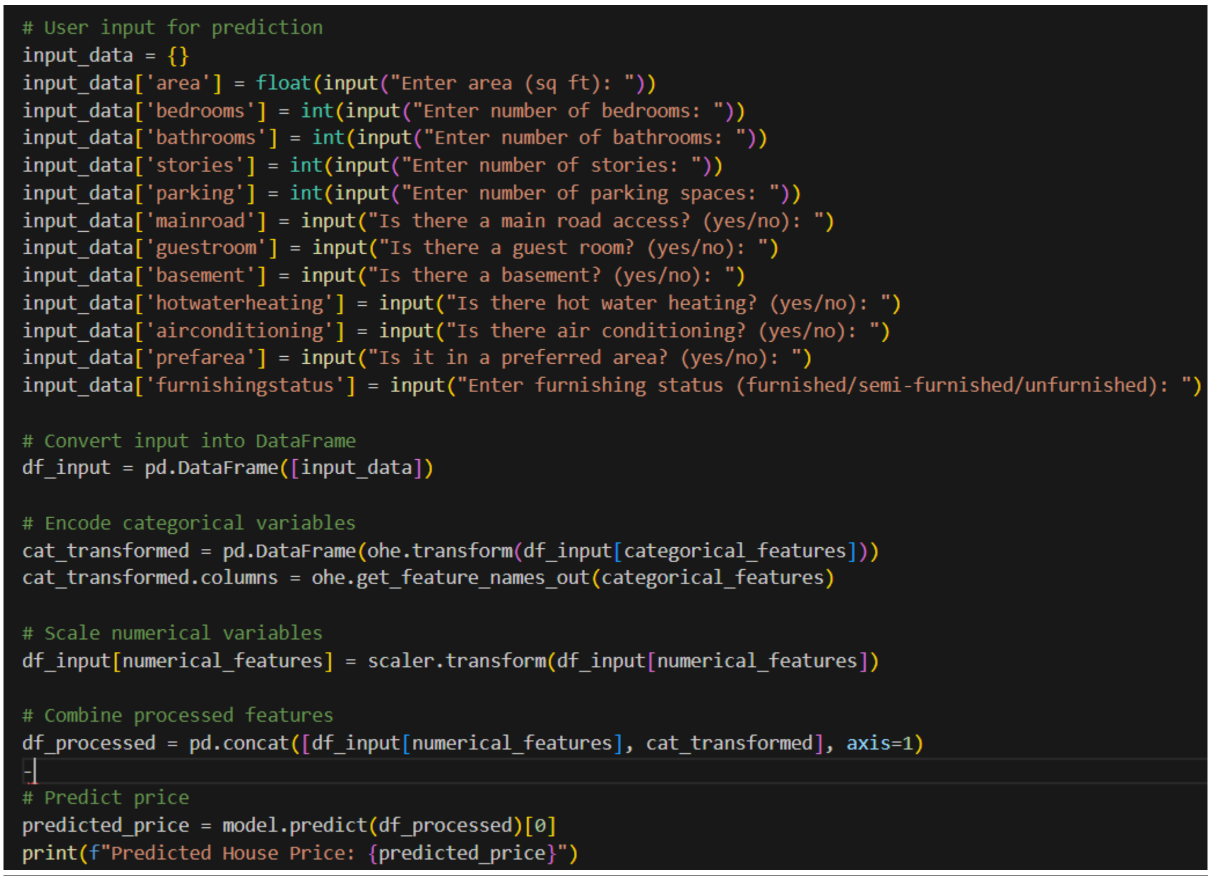
- **Data Preprocessing:**
    - Categorical features are encoded using **One-Hot Encoding** (converts categorical data into numerical form for machine learning models).
    - Numerical features are scaled using **StandardScaler** (standardizes data to have a mean of 0 and a standard deviation of 1 for better model performance).
- **Model Training:**
    - The dataset is split into training and testing sets (80-20 split).

 - A **Random Forest Regressor** (a model that averages predictions from multiple decision trees to improve accuracy and prevent overfitting) is trained on the data.
 - The model's performance is evaluated using **Mean Absolute Error (MAE)** and **Root Mean Squared Error (RMSE)** (common metrics for regression accuracy).

- **Model Saving:**
  - The trained model, along with the encoder and scaler, is saved using **Joblib** (a library for efficient model serialization and loading).
  - A models directory is created (if it doesn't already exist) to store the saved files.

User Input:

Code from predict.py:

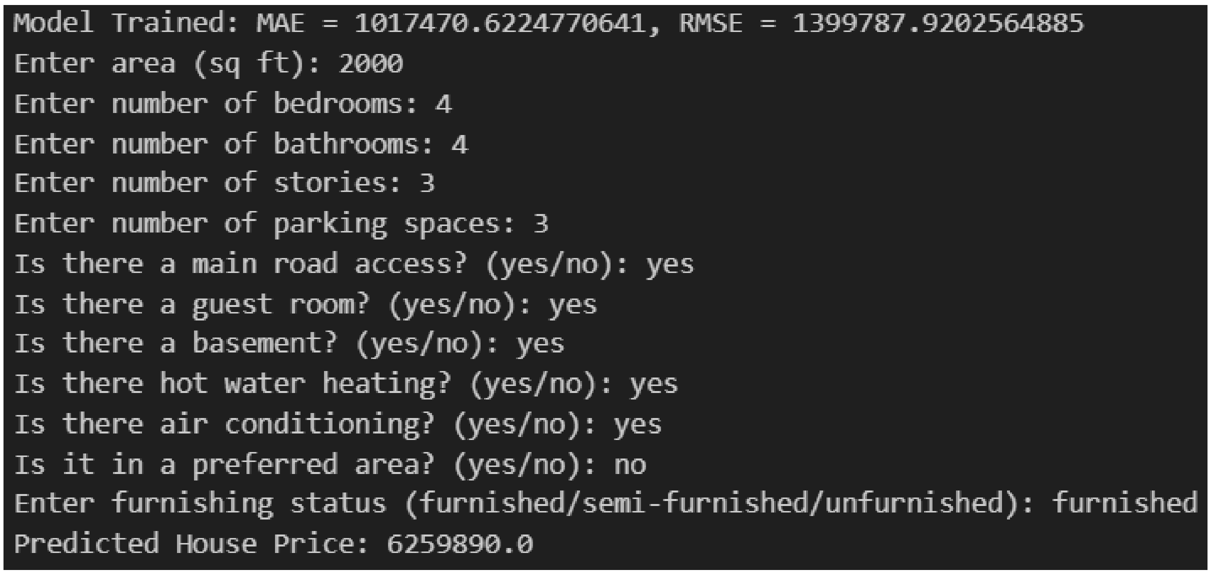


```
# User input for prediction
input_data = {}
input_data['area'] = float(input("Enter area (sq ft): "))
input_data['bedrooms'] = int(input("Enter number of bedrooms: "))
input_data['bathrooms'] = int(input("Enter number of bathrooms: "))
input_data['stories'] = int(input("Enter number of stories: "))
input_data['parking'] = int(input("Enter number of parking spaces: "))
input_data['mainroad'] = input("Is there a main road access? (yes/no): ")
input_data['guestroom'] = input("Is there a guest room? (yes/no): ")
input_data['basement'] = input("Is there a basement? (yes/no): ")
input_data['hotwaterheating'] = input("Is there hot water heating? (yes/no): ")
input_data['airconditioning'] = input("Is there air conditioning? (yes/no): ")
input_data['prefarea'] = input("Is it in a preferred area? (yes/no): ")
input_data['furnishingstatus'] = input("Enter furnishing status (furnished/semi-furnished/unfurnished): ")

# Convert input into DataFrame
df_input = pd.DataFrame([input_data])

# Encode categorical variables
cat_transformed = pd.DataFrame(ohe.transform(df_input[categorical_features]))
cat_transformed.columns = ohe.get_feature_names_out(categorical_features)

# Scale numerical variables
df_input[numerical_features] = scaler.transform(df_input[numerical_features])

# Combine processed features
df_processed = pd.concat([df_input[numerical_features], cat_transformed], axis=1)

# Predict price
predicted_price = model.predict(df_processed)[0]
print(f"Predicted House Price: {predicted_price}")
```

This code trains a Random Forest Regressor to predict house prices and allows users to input property details for price estimation.

- User Input & Prediction:
  - Users enter details such as area, bedrooms, and amenities.
  - Input data is preprocessed (categorical encoding + numerical scaling) to match the training format.
  - The trained model predicts the house price based on the user's input.

Output:

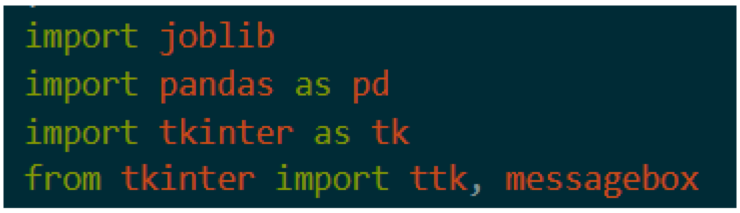


```
Model Trained: MAE = 1017470.6224770641, RMSE = 1399787.9202564885
Enter area (sq ft): 2000
Enter number of bedrooms: 4
Enter number of bathrooms: 4
Enter number of stories: 3
Enter number of parking spaces: 3
Is there a main road access? (yes/no): yes
Is there a guest room? (yes/no): yes
Is there a basement? (yes/no): yes
Is there hot water heating? (yes/no): yes
Is there air conditioning? (yes/no): yes
Is it in a preferred area? (yes/no): no
Enter furnishing status (furnished/semi-furnished/unfurnished): furnished
Predicted House Price: 6259890.0
```

User input processed and house price predicted using the trained Random Forest model.

PREDICT_UI.PY

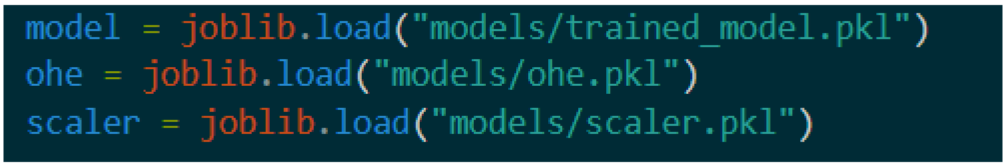
1. Import Libraries



```python
import joblib
import pandas as pd
import tkinter as tk
from tkinter import ttk, messagebox
```

- joblib: Loads the trained model and preprocessing tools (encoder & scaler).
- pandas: Handles tabular data — helps format user input like a dataset.
- tkinter & ttk: Build the GUI for user input.
- messagebox: Displays error alerts if input is incorrect.

2. Load the Trained Model & Preprocessors:

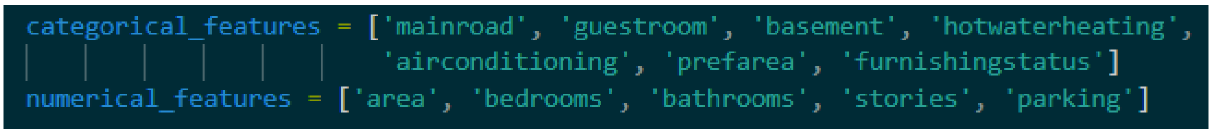


```python
model = joblib.load("models/trained_model.pkl")
ohe = joblib.load("models/ohe.pkl")
scaler = joblib.load("models/scaler.pkl")
```

- Loads:
    - model.pkl: The machine learning model that predicts house prices.

- ohe.pkl: One-hot encoder for categorical features.
- scaler.pkl: Scaler for normalizing numerical features.

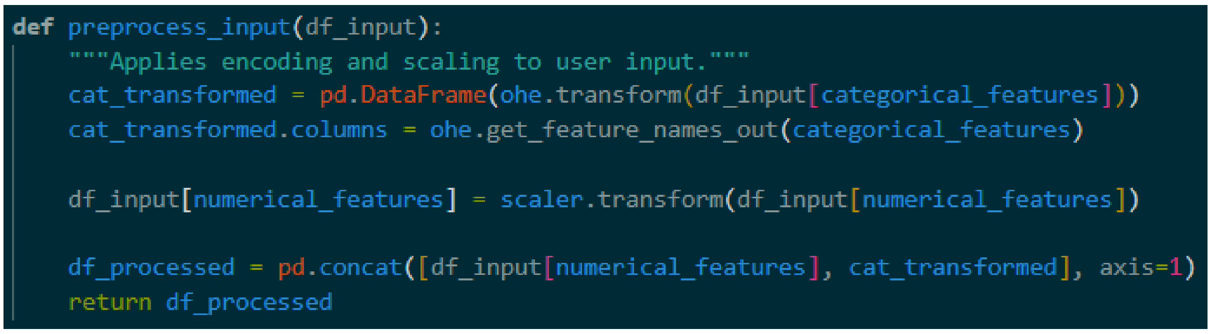
3. Define Feature Lists



```
categorical_features = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',
                        'airconditioning', 'prefarea', 'furnishingstatus']
numerical_features = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']
```

- Splits input features into:
  - categorical: Text values like "yes"/"no", or "furnished".
  - numerical: Numbers like area, bedrooms, etc.
- This helps apply the right transformation to each group.

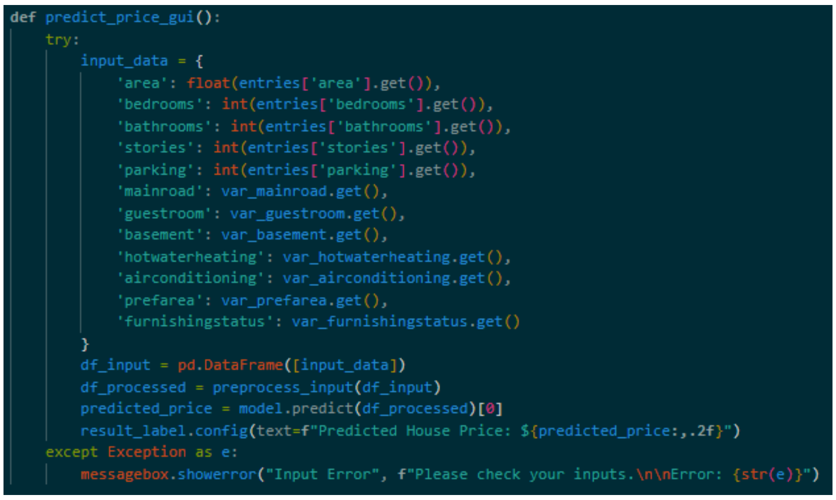
4. Preprocess User Input



```
def preprocess_input(df_input):
    """Applies encoding and scaling to user input."""
    cat_transformed = pd.DataFrame(ohe.transform(df_input[categorical_features]))
    cat_transformed.columns = ohe.get_feature_names_out(categorical_features)

    df_input[numerical_features] = scaler.transform(df_input[numerical_features])

    df_processed = pd.concat([df_input[numerical_features], cat_transformed], axis=1)
    return df_processed
```

- Applies one-hot encoding to categorical data.
- Applies scaling to numerical values.
- Combines both into a single DataFrame that the model understands.

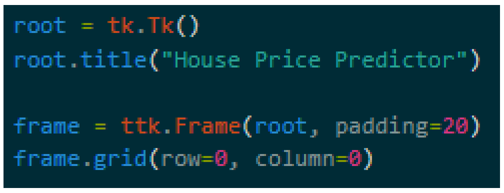
5. Predict House Price Based on GUI Input



```
def predict_price_gui():
    try:
        input_data = {
            'area': float(entries['area'].get()),
            'bedrooms': int(entries['bedrooms'].get()),
            'bathrooms': int(entries['bathrooms'].get()),
            'stories': int(entries['stories'].get()),
            'parking': int(entries['parking'].get()),
            'mainroad': var_mainroad.get(),
            'guestroom': var_guestroom.get(),
            'basement': var_basement.get(),
            'hotwaterheating': var_hotwaterheating.get(),
            'airconditioning': var_airconditioning.get(),
            'prefarea': var_prefarea.get(),
            'furnishingstatus': var_furnishingstatus.get()
        }
        df_input = pd.DataFrame([input_data])
        df_processed = preprocess_input(df_input)
        predicted_price = model.predict(df_processed)[0]
        result_label.config(text=f"Predicted House Price: ${predicted_price:,.2f}")
    except Exception as e:
        messagebox.showerror("Input Error", f"Please check your inputs.\n\nError: {str(e)}")
```

- Collects all user-entered values from GUI fields.
- Formats them into a DataFrame.
- Passes the input through preprocessing and prediction.
- Updates the label on the screen with the result.
- Handles invalid inputs with a popup error message.

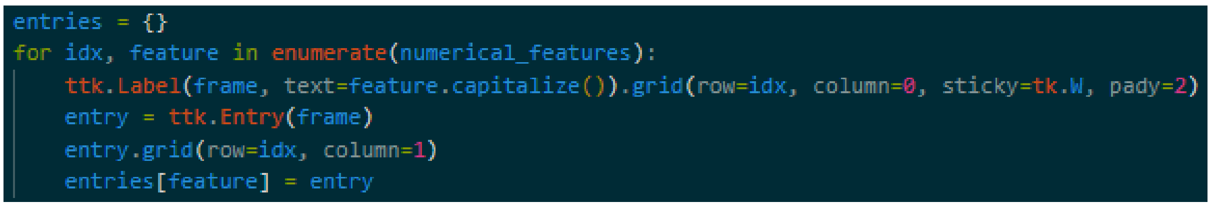
6. Set Up the GUI Window



```python
root = tk.Tk()
root.title("House Price Predictor")

frame = ttk.Frame(root, padding=20)
frame.grid(row=0, column=0)
```

- Starts the main app window.
- Adds a frame (container) to neatly organize inputs and outputs.

7. Create Input Fields (Numerical)



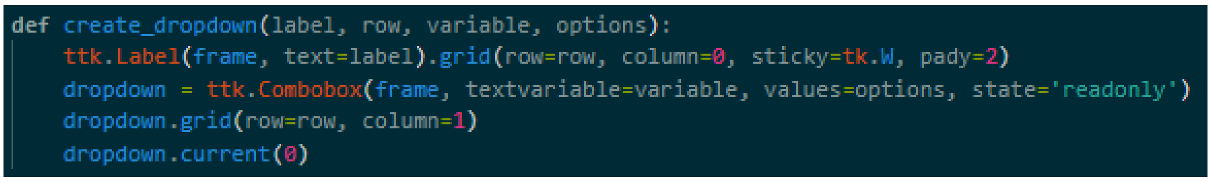
```python
entries = {}
for idx, feature in enumerate(numerical_features):
    ttk.Label(frame, text=feature.capitalize()).grid(row=idx, column=0, sticky=tk.W, pady=2)
    entry = ttk.Entry(frame)
    entry.grid(row=idx, column=1)
    entries[feature] = entry
```

- For each numerical feature, creates:
    - A label ("Area", "Bedrooms", etc.)
    - A text box for user input.
- Stores these inputs in a dictionary for easy access.

8. Dropdown Input Section – Categorical Data Handling

->Purpose

This part of the code handles categorical user inputs like “Main Road” or “Furnishing Status” using dropdown menus (comboboxes) in the Tkinter UI.



```python
def create_dropdown(label, row, variable, options):
    ttk.Label(frame, text=label).grid(row=row, column=0, sticky=tk.W, pady=2)
    dropdown = ttk.Combobox(frame, textvariable=variable, values=options, state='readonly')
    dropdown.grid(row=row, column=1)
    dropdown.current(0)
```

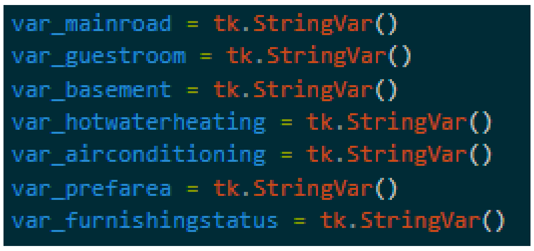
->What it does:

- label: Text label (e.g. "Main Road").
- row: Row position in the UI layout.
- variable: Tkinter variable to store selected value.
- options: The dropdown options (e.g. ['yes', 'no']).

->Key Features:

- Uses ttk.Combobox for a cleaner, modern dropdown.
- state='readonly' prevents user from typing manually.
- .current(0) sets default selected value to the first option.

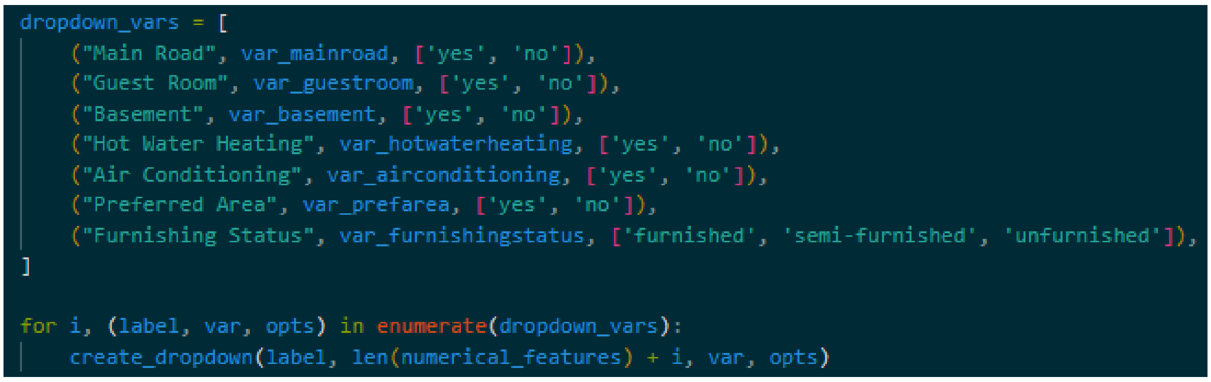
->Variables to Store Dropdown Selections



```python
var_mainroad = tk.StringVar()
var_guestroom = tk.StringVar()
var_basement = tk.StringVar()
var_hotwaterheating = tk.StringVar()
var_airconditioning = tk.StringVar()
var_prefarea = tk.StringVar()
var_furnishingstatus = tk.StringVar()
```

Each variable is a StringVar(), used to dynamically track and store the user's choice for that field.

->Dropdown Setup – Looping through all categorical fields



```python
dropdown_vars = [
    ("Main Road", var_mainroad, ['yes', 'no']),
    ("Guest Room", var_guestroom, ['yes', 'no']),
    ("Basement", var_basement, ['yes', 'no']),
    ("Hot Water Heating", var_hotwaterheating, ['yes', 'no']),
    ("Air Conditioning", var_airconditioning, ['yes', 'no']),
    ("Preferred Area", var_prefarea, ['yes', 'no']),
    ("Furnishing Status", var_furnishingstatus, ['furnished', 'semi-furnished', 'unfurnished']),
]

for i, (label, var, opts) in enumerate(dropdown_vars):
    create_dropdown(label, len(numerical_features) + i, var, opts)
```

->Explanation:

- dropdown_vars is a list of dropdown configuration tuples.
- The for loop dynamically creates all the dropdowns using the create_dropdown() function.
- Positioning: It adds dropdowns below the numerical entry fields by using len(numerical_features) + i for the row.

9. Add the Predict Button

```
ttk.Button(frame, text="Predict Price", command=predict_price_gui).grid(
    row=len(numerical_features) + len(dropdown_vars), columnspan=2, pady=10
)
```

- When clicked, this triggers the predict_price_gui() function.
- It gathers the data, makes the prediction, and displays the price.

10. Show the Predicted Price

```
result_label = ttk.Label(frame, text="Predicted House Price: $0.00", foreground='blue', font=('Helvetica', 12, 'bold'))
result_label.grid(row=len(numerical_features) + len(dropdown_vars) + 1, columnspan=2, pady=10)
```

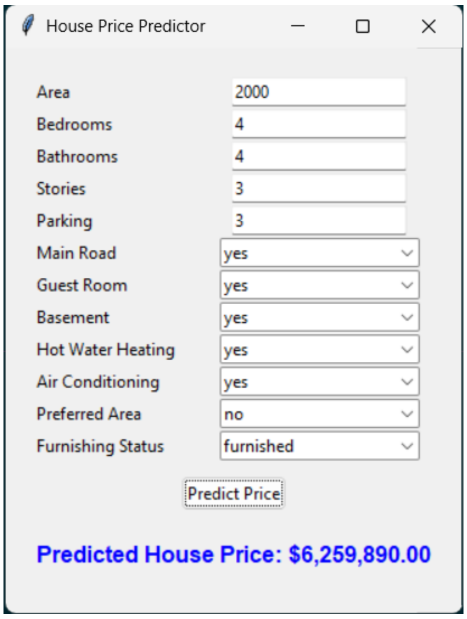
- Displays the predicted house price on the screen.
- Updated every time the user clicks "Predict".

11. Start the App Loop

```
root.mainloop()
```

- Starts the main GUI event loop — keeps the app running and responsive.

Output:


House Price Predictor

Area 2000
Bedrooms 4
Bathrooms 4
Stories 3
Parking 3
Main Road yes
Guest Room yes
Basement yes
Hot Water Heating yes
Air Conditioning yes
Preferred Area no
Furnishing Status furnished

Predicted House Price: $6,259,890.00

Summary:

- The app collects house details from the user.
- Preprocesses the input with ohe and scaler.
- Predicts the price using a trained model.
- Shows the result — all inside a nice interactive window

THE FINAL ITERATION OF OUR CODE, WHICH INCLUDES USER INPUT AND THE USER INTERFACE, IS INCLUDED IN THE RESULT.