SmartBridge Applied Data Science

Name: Routhu Sai Praneeth

Mail: routhusai.praneeth2020@vitstudent.ac.in

ADS Assignment 3

House Price Prediction:

Problem Description:

House price prediction is a common problem in the real estate industry and involves predicting the selling price of a house based on various features and attributes. The problem is typically approached as a regression problem, where the target variable is the price of the house, and the features are various attributes of the house The features used in house price prediction can include both quantitative and categorical variables, such as the number of bedrooms, house area, bedrooms, furnished, nearness to main road, and various amenities such as a garage and other factors that may influence the value of the property.

Accurate predictions can help agents and appraisers price homes correctly, while homeowners can use the predictions to set a reasonable asking price for their properties. Accurate house price prediction can also be useful for buyers who are looking to make informed decisions about purchasing a property and obtaining a fair price for their investment.

Attribute Information:

Name - Description

1- Price-Prices of the houses

2- Area- Area of the houses

3- Bedrooms- No of house bedrooms

4- Bathrooms- No of bathrooms

5- Stories- No of house stories

6- Main Road- Weather connected to Main road

7- Guestroom-Weather has a guest room

8- Basement-Weather has a basement

9- Hot water heating- Weather has a hot water heater

10-Airconditioning-Weather has a air conditioner

11-Parking- No of house parking

12-Furnishing Status-Furnishing status of house

Drive Link to Colab File:

Link

Building a Regression Model:

1. Downloaded the dataset.

2. Load the dataset into the tool.

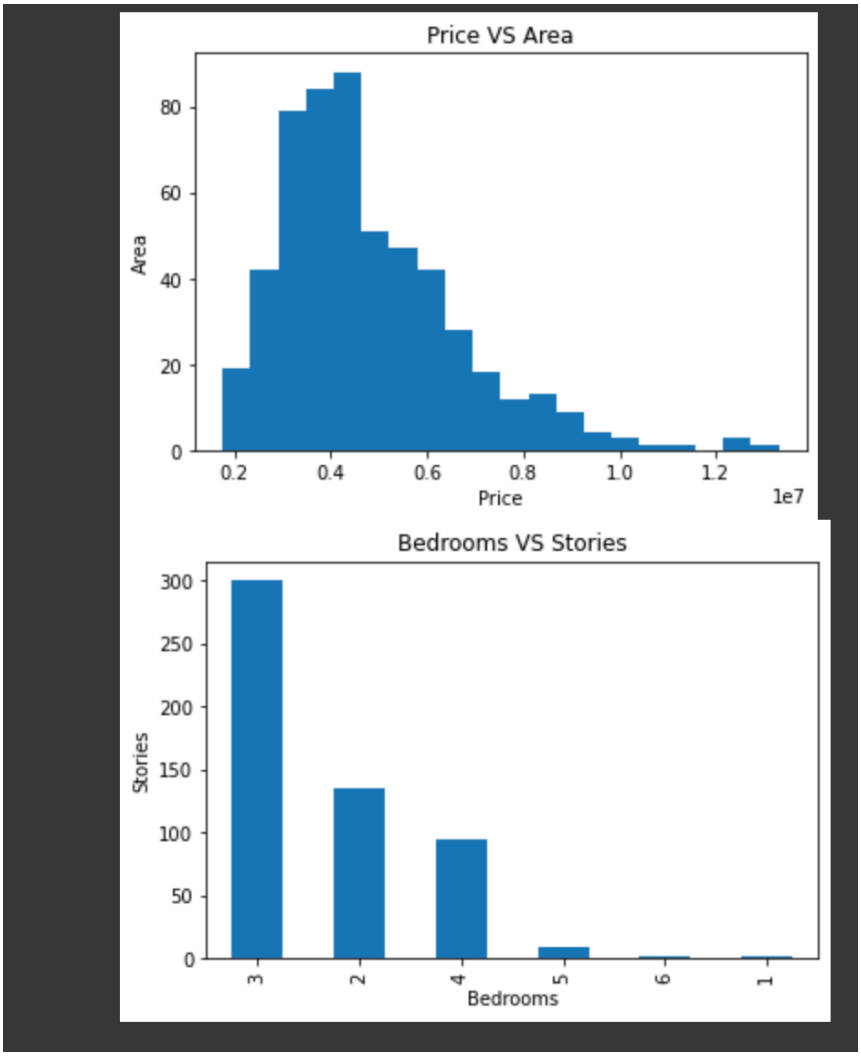
Routhu Sai Praneeth -> Assignment-3

```
import pandas as pd
data = pd.read_csv('housing.csv')
```

3. Perform Below Visualisations.

```
# Univariate analysis
import matplotlib.pyplot as plt

# Histogram  for price vs area
plt.hist(data['price'], bins=20)
plt.xlabel('Price')
plt.ylabel('Area')
plt.title('Price VS Area')
plt.show()

# Bar plot for bedrooms vs stories
data['bedrooms'].value_counts().plot(kind='bar')
plt.xlabel('Bedrooms')
plt.ylabel('Stories')
plt.title('Bedrooms VS Stories')
plt.show()
```

- **Univariate Analysis**

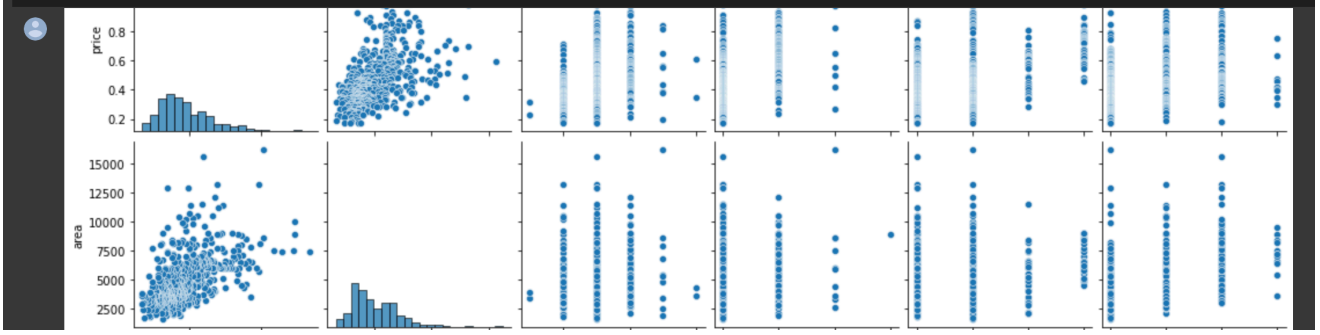


- **Bi-Variate Analysis**

```
# Bivariate analysis
import seaborn as sns

# Scatter plot for area vs price
sns.scatterplot(x='area', y='price', data=data)
plt.xlabel('Area')
plt.ylabel('Price')
plt.title('Price vs. Area')
plt.show()

# Box plot for parking price
sns.boxplot(x='parking', y='price', data=data)
plt.xlabel('Parking')
plt.ylabel('Price')
plt.title('Parking vs Price')
plt.show()
```

- **Multi-Variate Analysis**

```
# Multivariate analysis

corr_matrix = data.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()


sns.pairplot(data)
plt.show()
```



4. Perform descriptive statistics on the dataset.

```
[ ] statistics = data.describe()
    print(statistics)
```

```
                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
min      1.750000e+06   1650.000000    1.000000    1.000000    1.000000
25%      3.430000e+06   3600.000000    2.000000    1.000000    1.000000
50%      4.340000e+06   4600.000000    3.000000    1.000000    2.000000
75%      5.740000e+06   6360.000000    3.000000    2.000000    2.000000
max      1.330000e+07  16200.000000    6.000000    4.000000    4.000000

           parking
count   545.000000
mean      0.693578
std       0.861586
min       0.000000
25%       0.000000
50%       0.000000
75%       1.000000
max       3.000000
```

5.Check for Missing values and deal with them.

```
[ ]  # Missing values
     missing_values = data.isnull().sum()
     print(missing_values)
     data = data.dropna()
```

```
price               0
area                0
bedrooms            0
bathrooms           0
stories             0
mainroad            0
guestroom           0
basement            0
hotwaterheating     0
airconditioning     0
parking             0
furnishingstatus    0
dtype: int64
```

6. Find the outliers and replace them outliers

```
#Now we will be detecting outliers
import numpy as np
def detect_outliers_zscore(data, threshold=3):
    z_scores = np.abs((data - data.mean()) / data.std())
    outliers = data[z_scores > threshold]
    return outliers
numerical_columns = ['price', 'area', 'bedrooms', 'bathrooms','stories','parking']
for column in numerical_columns:
    outliers = detect_outliers_zscore(data[column])
    print(f"Outliers in {column}:")
    print(outliers)
```

```
Outliers in price:
0    13300000
1    12250000
2    12250000
3    12215000
4    11410000
5    10850000
Name: price, dtype: int64
Outliers in area:
7      16200
10     13200
66     13200
69     12090
125    15600
211    12900
403    12944
Name: area, dtype: int64
Outliers in bedrooms:
112    6
395    6
```

7. Check for Categorical columns and perform encoding.

```
[ ]  # Checking for Categorical Columns
     categorical_columns = data.select_dtypes(include=['object']).columns

     # One-Hot encoding
     data_encoded = pd.get_dummies(data, columns=categorical_columns)
```

8. Split the data into dependent and independent variables, Scale the independent variables, Split the data into training and testing

```
[ ]  # Spliting dependent and independent variables
     X = data.drop('price', axis=1)  # X is Independent variables
     y = data['price']  # Y is Dependent variable
```

```
[ ]  from sklearn.preprocessing import StandardScaler

     scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X)
```

```
[ ]  # Spliting the data into testing and training
     from sklearn.model_selection import train_test_split

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

11. Build the Model, Train the Model, Test the Model, Measure the performance using Metrics.

```
[ ] # Building the model
    from sklearn.linear_model import LinearRegression

    model = LinearRegression()
```

```
[ ] # Now we will Train the model
    model.fit(X_train, y_train)
```

```
    LinearRegression()
```

```
[ ] # Now Testing the model
    y_pred = model.predict(X_test)
```

```
    # Performance Measure

    from sklearn.metrics import mean_squared_error, r2_score

    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)

    print("RMSE:", rmse)
    print("R-squared:", r2)
```

```
    RMSE: 40.29813356022482
    R-squared: 0.5744138777194269
```