DASARI.NAGAVENI

EMAIL: dasari.nagaveni2020@vitstudent.ac.in

Vellore institute of technology

Chennai

Assignment-3

Problem Statement: House Price Prediction

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

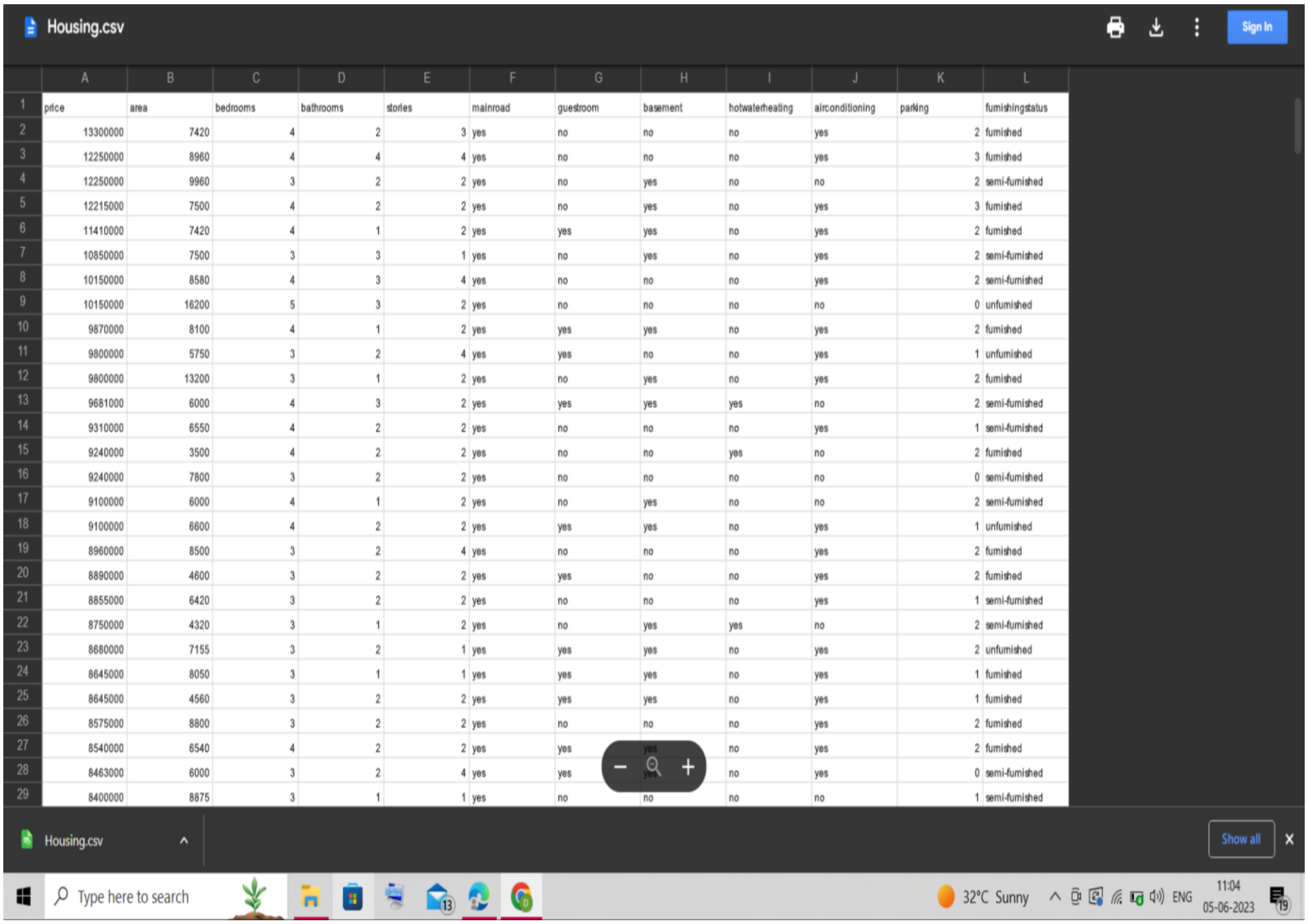
9- Hot water heating- Weather has a hot water heater

10-Airconditioning-Weather has a air conditioner

11-Parking- No of house parking

12-Furnishing Status-Furnishing status of house

1. Download the dataset: Dataset



	A	B	C	D	E	F	G	H	I	J	K	L
1	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnishingstatus
2	13300000	7420	4	2	3	yes	no	no	no	yes	2	furnished
3	12250000	8960	4	4	4	yes	no	no	no	yes	3	furnished
4	12250000	9960	3	2	2	yes	no	yes	no	no	2	semi-furnished
5	12215000	7500	4	2	2	yes	no	yes	no	yes	3	furnished
6	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	furnished
7	10850000	7500	3	3	1	yes	no	yes	no	yes	2	semi-furnished
8	10150000	8580	4	3	4	yes	no	no	no	yes	2	semi-furnished
9	10150000	16200	5	3	2	yes	no	no	no	no	0	unfurnished
10	9870000	8100	4	1	2	yes	yes	yes	no	yes	2	furnished
11	9800000	5750	3	2	4	yes	yes	no	no	yes	1	unfurnished
12	9800000	13200	3	1	2	yes	no	yes	no	yes	2	furnished
13	9681000	6000	4	3	2	yes	yes	yes	yes	no	2	semi-furnished
14	9310000	6550	4	2	2	yes	no	no	no	yes	1	semi-furnished
15	9240000	3500	4	2	2	yes	no	no	yes	no	2	furnished
16	9240000	7800	3	2	2	yes	no	no	no	no	0	semi-furnished
17	9100000	6000	4	1	2	yes	no	yes	no	no	2	semi-furnished
18	9100000	6600	4	2	2	yes	yes	yes	no	yes	1	unfurnished
19	8960000	8500	3	2	4	yes	no	no	no	yes	2	furnished
20	8890000	4600	3	2	2	yes	yes	no	no	yes	2	furnished
21	8855000	6420	3	2	2	yes	no	no	no	yes	1	semi-furnished
22	8750000	4320	3	1	2	yes	no	yes	yes	no	2	semi-furnished
23	8680000	7155	3	2	1	yes	yes	yes	no	yes	2	unfurnished
24	8645000	8050	3	1	1	yes	yes	yes	no	yes	1	furnished
25	8645000	4560	3	2	2	yes	yes	yes	no	yes	1	furnished
26	8575000	8800	3	2	2	yes	no	no	no	yes	2	furnished
27	8540000	6540	4	2	2	yes	yes	yes	no	yes	2	furnished
28	8463000	6000	3	2	4	yes	yes	yes	no	yes	0	semi-furnished
29	8400000	8875	3	1	1	yes	no	no	no	no	1	semi-furnished

2. Load the dataset into the tool.

```
[6] import numpy as np
    import pandas as pd
    import difflib
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.metrics.pairwise import cosine_similarity
```

```
[7] df=pd.read_csv('/content/Housing.csv')
```

```
import numpy as np

import pandas as pd

import difflib

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine_similarity

df=pd.read_csv('/content/Housing.csv')
```

3. Perform Below Visualizations.

- Univariate Analysis
- Bi-Variate Analysis
- Multi-Variate Analysis

Statistics summary

```
summary_stats = df.describe()

print(summary_stats)
```

```
[7] df=pd.read_csv('/content/Housing.csv')
```

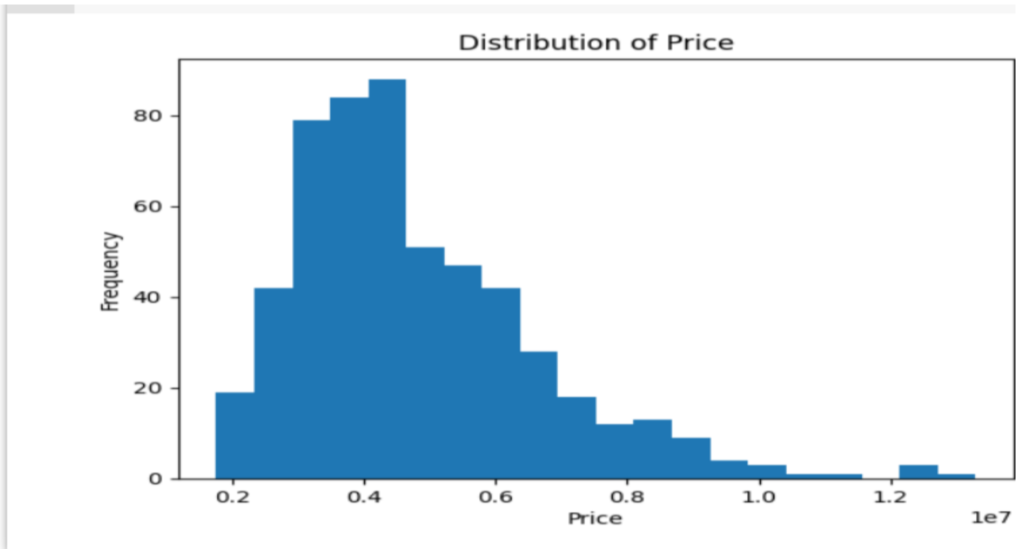
```
summary_stats = df.describe()
print(summary_stats)
```

```
              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
```

Univariate Analysis

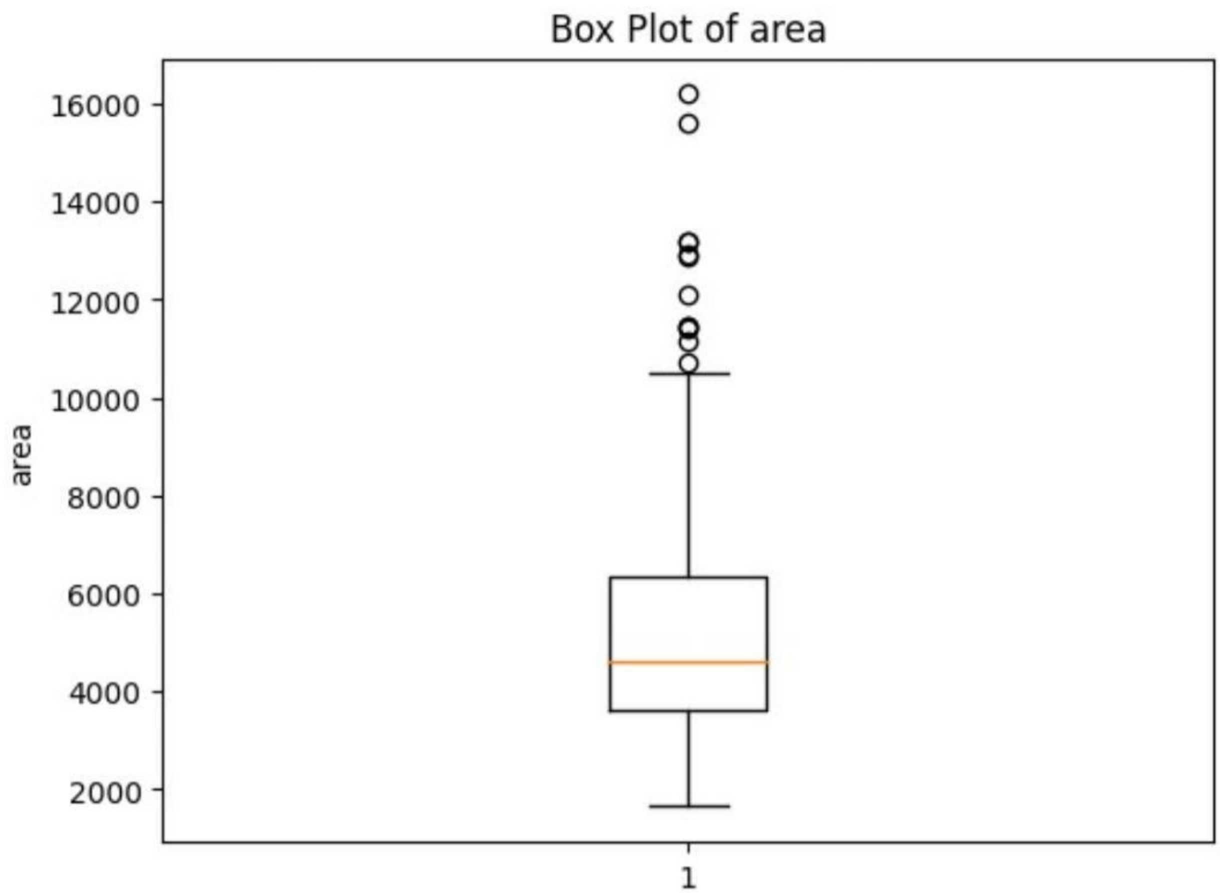
HISTOGRAM

```
# Histogram

import matplotlib.pyplot as plt

plt.hist(df['price'], bins=20)

plt.xlabel('Price')

plt.ylabel('Frequency')

plt.title('Distribution of Price')

plt.show()
```



BOXPLOT

```python
# Box plot
plt.boxplot(df['area'])
plt.ylabel('area')
plt.title('Box Plot of area')
plt.show()
```



COUNT OF UNIQUE VALUES

```
# Count of unique values

value_counts = df['bedroom

(kind='bar')

plt.xlabel('Bedrooms')

plt.ylabel('Count')

plt.title('Number of Bedrooms')

plt.show()
```

3   300

2   136

4   95

5   10

6   2

1   2
2   Name: bedrooms, dtype: int64

Bar chart

 # Bar chart

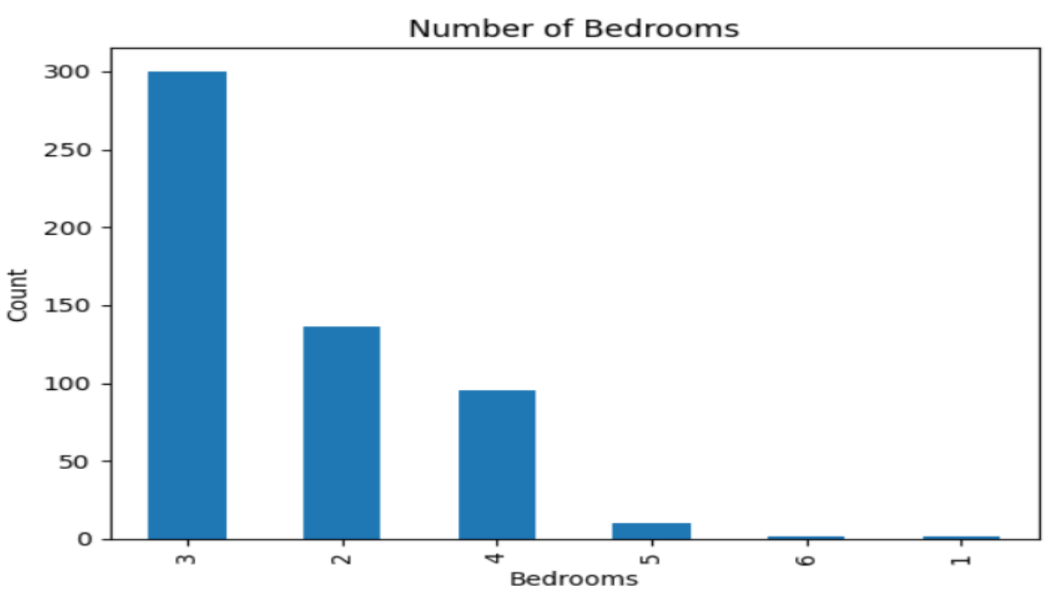
Value_counts.plot(kind=’bar’)

Plt.xlabel(‘Bedrooms’)

Plt.ylabel(‘Count’)

Plt.title(‘Number of Bedrooms’)

Plt.show()

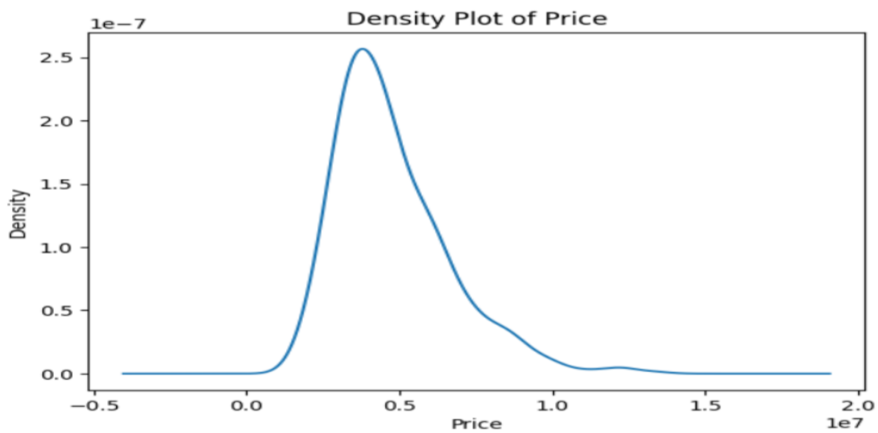


DENSITY PLOT

Density plot

df['price'].plot(kind='density')

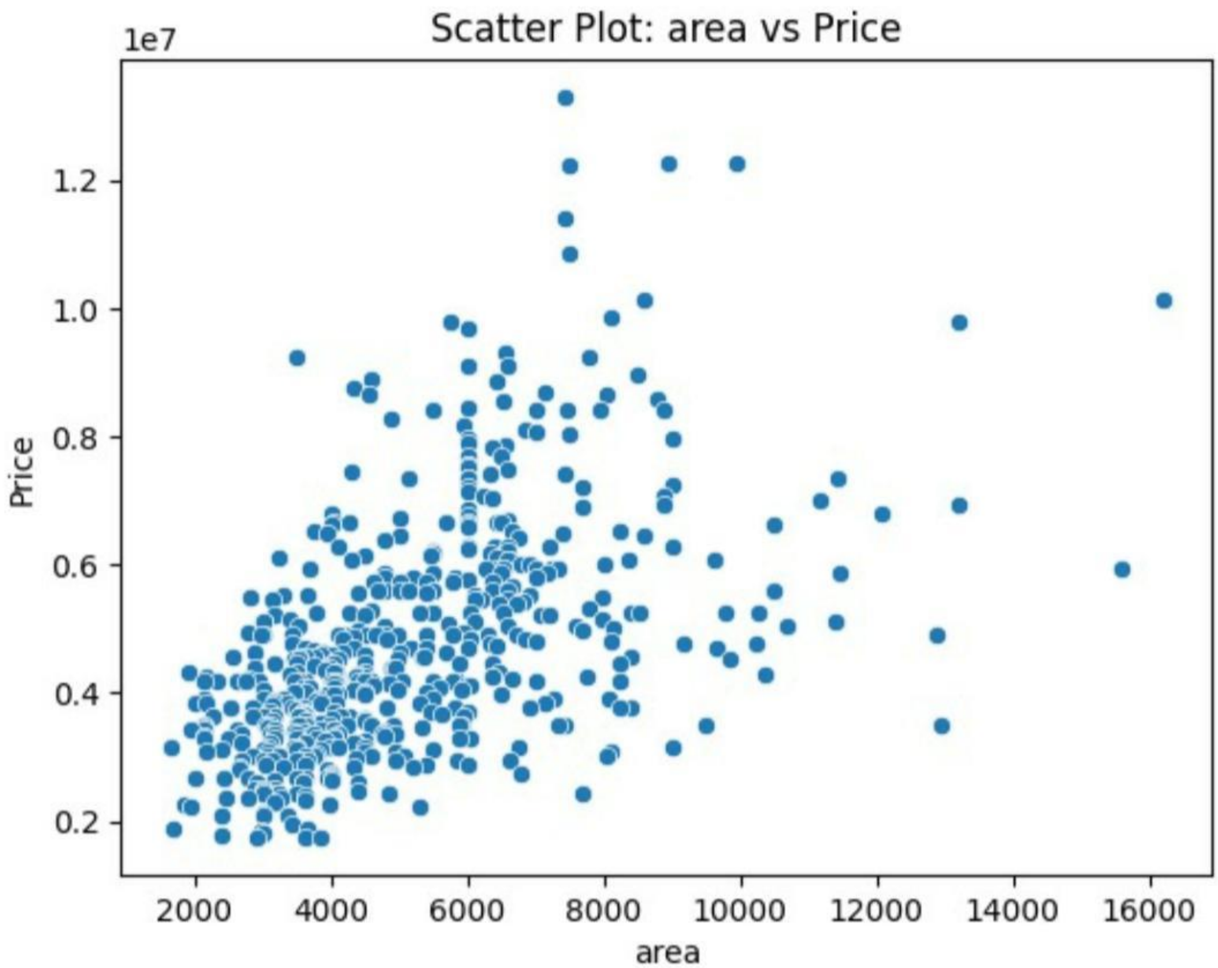
```
plt.xlabel('Price')

plt.title('Density Plot of Price')

plt.show()
```



BI-VARIATE ANALYSIS

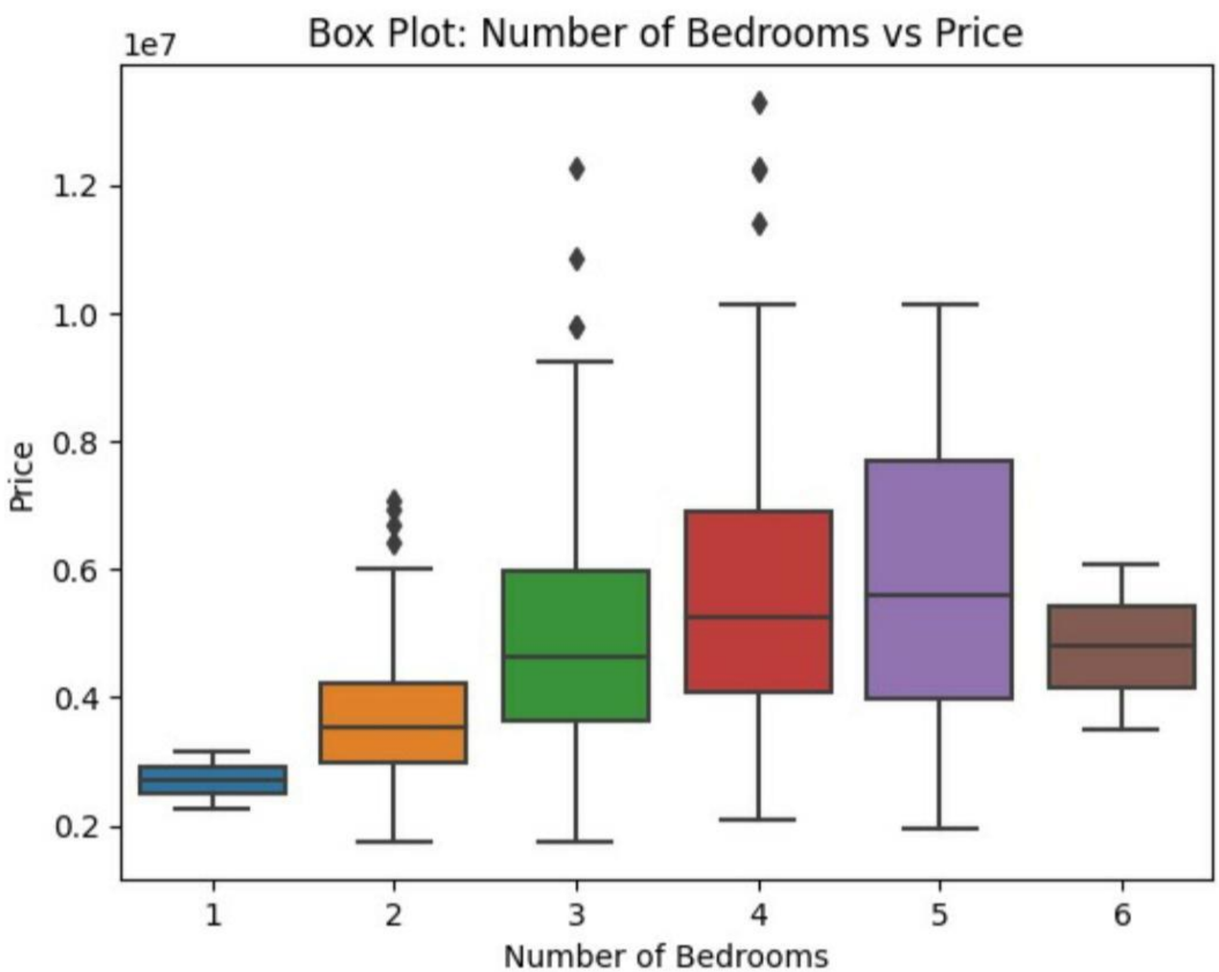
SCATTER PLOT

```
# Scatter plot

import seaborn as sns

sns.scatterplot(x='area', y='price', data=df)

plt.xlabel('area')

plt.ylabel('Price')

plt.title('Scatter Plot: area vs Price')

plt.show()
```


Scatter Plot: area vs Price
1e7
1.2
1.0
0.8
0.6
0.4
0.2
Price
2000
4000
6000
8000
10000
12000
14000
16000
area

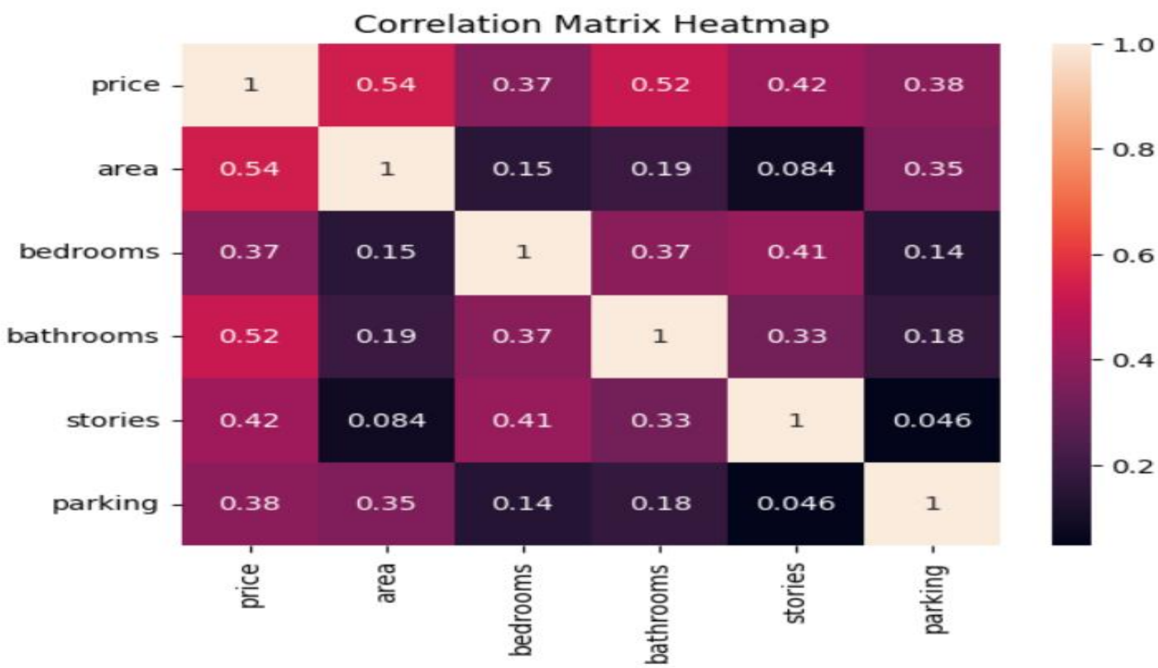
BOX PLOT

```
# Box plot

sns.boxplot(x='bedrooms', y='price', data=df)

plt.xlabel('Number of Bedrooms')

plt.ylabel('Price')

plt.title('Box Plot: Number of Bedrooms vs Price')

plt.show()
```


Box Plot: Number of Bedrooms vs Price
1e7
1.2
1.0
0.8
0.6
0.4
0.2
Price
1
2
3
4
5
6
Number of Bedrooms

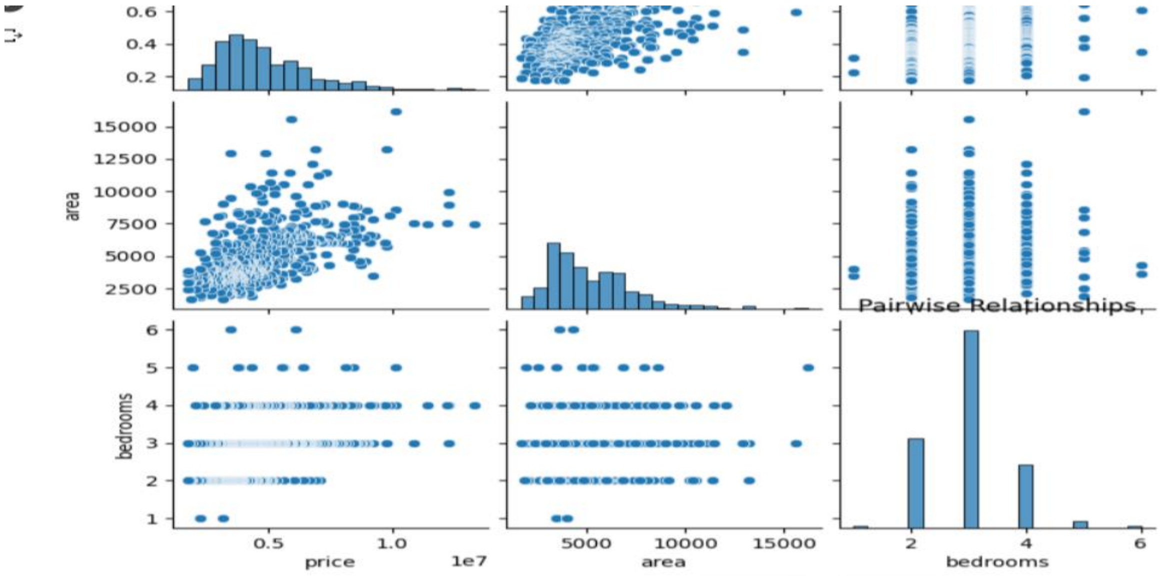
HEAT MAP

```
# Heatmap
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



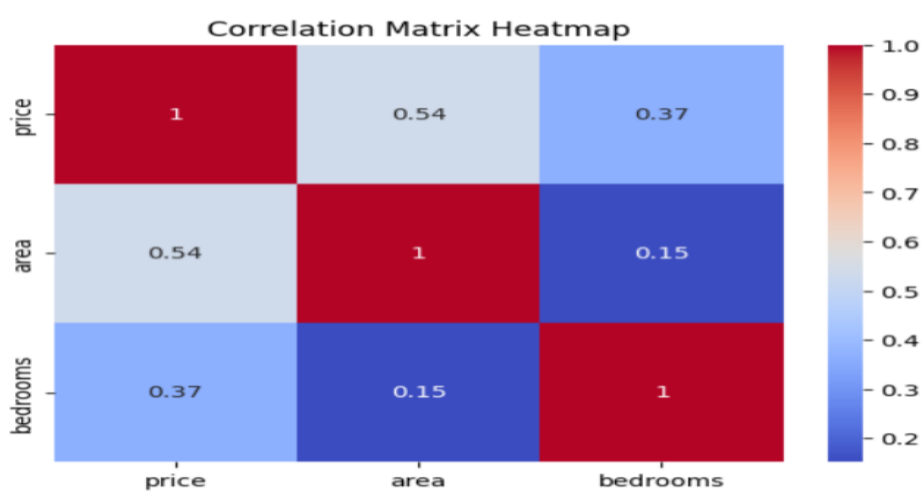
PAIRPLOT

```
variables_of_interest = ['price', 'area', 'bedrooms']


# Pairplot
sns.pairplot(df[variables_of_interest])
plt.title('Pairwise Relationships')
plt.show()
```



CORRELATION MATRIX

```
# Correlation matrix heatmap
corr_matrix = df[variables_of_interest].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```



4. Perform descriptive statistics on the dataset.

Calculate descriptive statistics

descriptive_stats = df.describe()

Print the descriptive statistics

print(descriptive_stats)

```
              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.

 # Check for missing values

missing_values = df.isnull().sum()

print(missing_values)

 # Option 1: Drop rows with missing values

df_dropped = df.dropna()

print("Shape after dropping missing values:", df_dropped.shape)

 # Option 2: Fill missing values with mean/median/mode

 # For example, fill missing values in the 'price' column with the mean

```
mean_price = df['price'].mean()

df_filled = df.fillna({'price': mean_price})

print("Missing values filled with mean:", df_filled.isnull().sum())
```

 # Option 3: Fill missing values with forward fill or backward fill

 # For example, fill missing values using forward fill method

```
df_ffilled = df.fillna(method='ffill')

print("Missing values filled with forward fill:", df_ffilled.isnull().sum())
```

```
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
Shape after dropping missing values: (545, 12)
Missing values filled with mean: 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
Missing values filled with forward fill: price              0
```

```
+ Code    + Text
Missing values filled with mean: 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
Missing values filled with forward fill: 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

```
# Define the columns to check for outliers

columns_to_check = ['price', 'area']



# Define the outlier detection method (e.g., z-score or IQR)

outlier_method = 'z-score'



# Define the threshold for outlier detection

threshold = 3



# Function to replace outliers with a suitable value (e.g., median)

def replace_outliers(data, col):

   if outlier_method == 'z-score':

      z_scores = (data[col] - data[col].mean()) / data[col].std()

      outliers = np.abs(z_scores) > threshold

   elif outlier_method == 'iqr':

      Q1 = data[col].quantile(0.25)

      Q3 = data[col].quantile(0.75)
```

```
        IQR = Q3 - Q1
        outliers = (data[col] < (Q1 - 1.5 * IQR)) | (data[col] > (Q3 + 1.5 * IQR))
    else:
        print("Invalid outlier detection method")
        return data

    data.loc[outliers, col] = data[col].median()
    return data


# Iterate over the columns to check for outliers and replace them
for col in columns_to_check:
    df = replace_outliers(df, col)



# Display the modified dataset
print(df)
```

```
       price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0    4340000  7420         4          2        3      yes        no       no
1    4340000  8960         4          4        4      yes        no       no
2    4340000  9960         3          2        2      yes        no      yes
3    4340000  7500         4          2        2      yes        no      yes
4    4340000  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 furnishingstatus
0                no             yes        2        furnished
1                no             yes        3        furnished
2                no              no        2   semi-furnished
3                no             yes        3        furnished
4                no             yes        2        furnished
..              ...             ...      ...              ...
540              no              no        2      unfurnished
541              no              no        0   semi-furnished
542              no              no        0      unfurnished
543              no              no        0        furnished
544              no              no        0      unfurnished

[545 rows x 12 columns]
```

7. Check for Categorical columns and perform encoding.

```
# Identify categorical columns
categorical_columns = df.select_dtypes(include=['object']).columns
```

```
print("Categorical Columns:", categorical_columns)



# Perform one-hot encoding

df_encoded = pd.get_dummies(df, columns=categorical_columns)

print("Encoded DataFrame:")

print(df_encoded)
```

```
Categorical Columns: Index(['mainroad', 'guestroom', 'basement', 'hotwaterheating',
       'airconditioning', 'furnishingstatus'],
      dtype='object')
```

```
Encoded DataFrame:
       price  area  bedrooms  bathrooms  stories  parking  mainroad_no  \
0    4340000  7420         4          2        3        2            0
1    4340000  8960         4          4        4        3            0
2    4340000  9960         3          2        2        2            0
3    4340000  7500         4          2        2        3            0
4    4340000  7420         4          1        2        2            0
..       ...   ...       ...        ...      ...      ...          ...
540  1820000  3000         2          1        1        2            0
541  1767150  2400         3          1        1        0            1
542  1750000  3620         2          1        1        0            0
543  1750000  2910         3          1        1        0            1
544  1750000  3850         3          1        2        0            0

     mainroad_yes  guestroom_no  guestroom_yes  basement_no  basement_yes  \
0               1             1              0            1             0
1               1             1              0            1             0
2               1             1              0            0             1
3               1             1              0            0             1
4               1             0              1            0             1
..            ...           ...            ...          ...           ...
540             1             1              0            0             1
541             0             1              0            1             0
542             1             1              0            1             0
543             0             1              0            1             0
544             1             1              0            1             0

     hotwaterheating_no  hotwaterheating_yes  airconditioning_no  \
0                     1                    0                   0
```

```
1                    1                    0                    0
2                    1                    0                    1
3                    1                    0                    0
4                    1                    0                    0
..                 ...                  ...                  ...
540                  1                    0                    1
541                  1                    0                    1
542                  1                    0                    1
543                  1                    0                    1
544                  1                    0                    1

     airconditioning_yes  furnishingstatus_furnished  \
0                      1                           1
1                      1                           1
2                      0                           0
3                      1                           1
4                      1                           1
..                   ...                         ...
540                    0                           0
541                    0                           0
542                    0                           0
543                    0                           1
544                    0                           0

     furnishingstatus_semi-furnished  furnishingstatus_unfurnished
0                                  0                             0
1                                  0                             0
2                                  1                             0
3                                  0                             0
4                                  0                             0
..                               ...                           ...
540                                0                             1
541                                1                             0
542                                0                             1
543                                0                             0
544                                0                             1

[545 rows x 19 columns]
```

8. Split the data

into dependent and independent variables.

```
# Split the data into dependent and independent variables

X = df.drop('price', axis=1)  # Independent variables

y = df['price']            # Dependent variable
```

```
# Print the shape of the variables

print("Shape of X:", X.shape)

print("Shape of y:", y.shape)
```

```
Shape of y: (545,)
```

9. Scale the independent variables

```
import pandas as pd

from sklearn.preprocessing import StandardScaler

#Assuming your Titanic dataset is stored in a pandas DataFrame called 'df"

#Assuming your independent variables are stored in a DataFrame called

#Select the independent variables you want to scale

independent_variables = ['price', 'area', 'bathrooms']

# Create a new DataFrame with only the selected independent variables

X_selected = df[independent_variables]
```

```
#Initialize the StandardScaler

scaler = StandardScaler()

# Fit and transform the selected independent variables

X_scaled = scaler.fit_transform(X_selected)

# Create a new DataFrame with the scaled independent variables

X_scaled_df = pd.DataFrame(X_scaled, columns=independent_variables)

# Print the first few rows of the scaled independent variables

print(X_scaled_df.head())
```

```python
import pandas as pd

from sklearn.model_selection import train_test_split




# Split the data into dependent and independent variables

X = df.drop('price', axis=1)  # Independent variables

y = df['price']             # Dependent variable



# Split the data into training and testing sets

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



# Print the shapes of the train and test sets

print("Shape of X_train:", X_train.shape)

print("Shape of X_test:", X_test.shape)

print("Shape of y_train:", y_train.shape)

print("Shape of y_test:", y_test.shape)
```

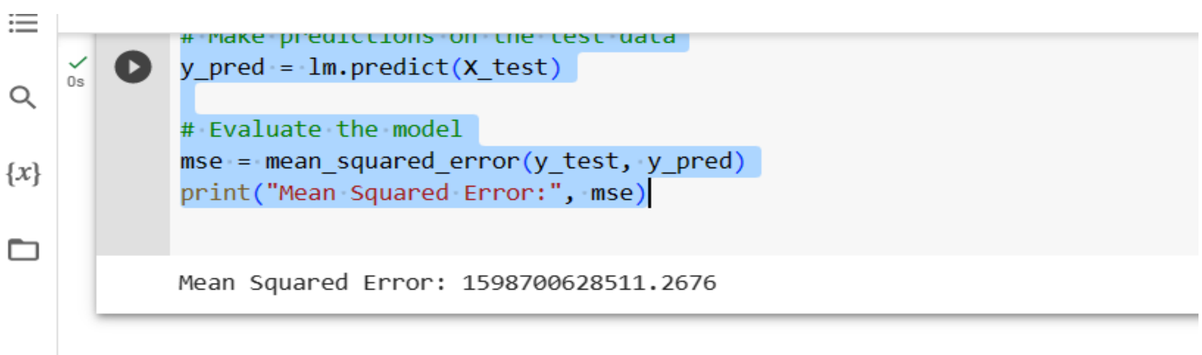
```
Shape of y_test: (109,)
```

11. Build the Model

```python
import pandas as pd

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split
```

```python
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_squared_error



# Split the data into dependent and independent variables
X = df.drop('price', axis=1)  # Independent variables
y = df['price']             # Dependent variable

# Identify the categorical columns
categorical_cols = X.select_dtypes(include=['object']).columns

# Perform one-hot encoding for categorical variables
ct = ColumnTransformer([('encoder', OneHotEncoder(), categorical_cols)], remainder='passthrough')
X_encoded = ct.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

# Build the linear regression model
lm = LinearRegression()

# Fit the model to the training data
lm.fit(X_train, y_train)
```

```
# Make predictions on the test data
y_pred = lm.predict(X_test)


# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

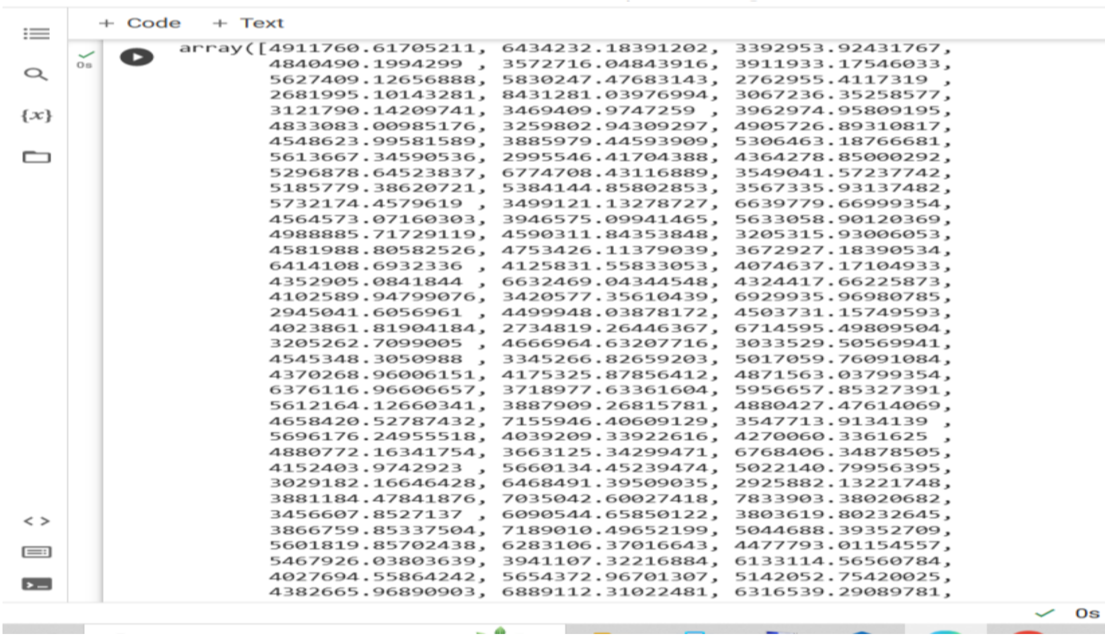


12. Train the Model

```
# Fit the model to the training data
lm.fit(X_train, y_train)
```

13. Test the Model

```
# Make predictions on the test data
y_pred = model.predict(X_test)


y_pred
```




```
array([4911760.61705211, 6434232.18391202, 3392953.92431767,
       4840490.1994299 , 3572716.04843916, 3911933.17546033,
       5627409.12656888, 5830247.47683143, 2762955.4117319 ,
       2681995.10143281, 8431281.03976994, 3067236.35258577,
       3121790.14209741, 3469409.9747259 , 3962974.95809195,
       4833083.00985176, 3259802.94309297, 4905726.89310817,
       4548623.99581589, 3885979.44593909, 5306463.18766681,
       5613667.34590536, 2995546.41704388, 4364278.85000292,
       5296878.64523837, 6774708.43116889, 3549041.57237742,
       5185779.38620721, 5384144.85802853, 3567335.93137482,
       5732174.4579619 , 3499121.13278727, 6639779.66999354,
       4564573.07160303, 3946575.09941465, 5633058.90120369,
       4988885.71729119, 4590311.84353848, 3205315.93006053,
       4581988.80582526, 4753426.11379039, 3672927.18390534,
       6414108.6932336 , 4125831.55833053, 4074637.17104933,
       4352905.0841844 , 6632469.04344548, 4324417.66225873,
       4102589.94799076, 3420577.35610439, 6929935.96980785,
       2945041.6056961 , 4499948.03878172, 4503731.15749593,
       4023861.81904184, 2734819.26446367, 6714595.49809504,
       3205262.7099005 , 4666964.63207716, 3033529.50569941,
       4545348.3050988 , 3345266.82659203, 5017059.76091084,
       4370268.96006151, 4175325.87856412, 4871563.03799354,
       6376116.96606657, 3718977.63361604, 5956657.85327391,
       5612164.12660341, 3887909.26815781, 4880427.47614069,
       4658420.52787432, 7155946.40609129, 3547713.9134139 ,
       5696176.24955518, 4039209.33922616, 4270060.3361625 ,
       4880772.16341754, 3663125.34299471, 6768406.34878505,
       4152403.9742923 , 5660134.45239474, 5022140.79956395,
       3029182.16646428, 6468491.39509035, 2925882.13221748,
       3881184.47841876, 7035042.60027418, 7833903.38020682,
```

```
3456607.8527137 , 6090544.65850122, 3803619.80232645,
3866759.85337504, 7189010.49652199, 5044688.39352709,
5601819.85702438, 6283106.37016643, 4477793.01154557,
5467926.03803639, 3941107.32216884, 6133114.56560784,
4027694.55864242, 5654372.96701307, 5142052.75420025,
4382665.96890903, 6889112.31022481, 6316539.29089781,
5479741.32966125])
```

14. Measure the performance using Metrics.

```python
from sklearn.metrics import r2_score



r2 = r2_score(y_test, y_pred)

print("R-squared Score:", r2)

r2
```

```
R-squared Score: 0.5928639479880735

0.5928639479880735
```