Idea: Predicting House Prices with Linear Regression

Description:

The objective of this project is to build a predictive model using linear regression to estimate a numerical outcome based on a dataset with relevant features. Linear regression is a fundamental machine learning algorithm, and this project provides hands-on experience in developing, evaluating, and interpreting a predictive model

```
#Importing necessary libraries
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import pointbiserialr
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Loading data

```
# Access the CSV file with the  path STEP1 LOAD DATA
path="/content/drive/MyDrive/OASIS/Housing.csv"
df = pd.read_csv(path, na_values=["NA", "NaN", "", "?","Not Available"])
```

```
df.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwater
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	

Next steps: View recommended plots

Data Inspection, Exploration and cleaning

```
df.columns
```

```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'furnishingstatus'],
      dtype='object')
```

```
df.dtypes
```

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

```
df.shape
```

```
(545, 13)
```

```
# Handle missing values (e.g., impute or drop rows/columns based on analysis)
(df.isnull().sum())
```

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

```
#Checking column types
df.info()
```

```
<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
```

```
df.nunique()
```

```
price               219
area                284
bedrooms              6
bathrooms             4
stories               4
mainroad              2
guestroom             2
basement              2
hotwaterheating       2
airconditioning       2
parking               4
prefarea              2
furnishingstatus      3
dtype: int64
```

```
print("Duplicates:", df.duplicated().sum())
```

```
Duplicates: 0
```

```
# Drop rows with missing values
df.dropna(inplace=True)
```

```
df.shape
```

```
(545, 13)
```

```
#first fetch all the categorical columns with Yes and NO
categorical =  ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', '
#write a function to change yes to 1 and no to 0
def binary_map(x):
    return x.map({'yes': 1, "no": 0})

# now replace yes and no with 1 and 0 in our dataset
df[categorical] = df[categorical].apply(binary_map)


df.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwater
0	13300000	7420	4	2	3	1	0	0	
1	12250000	8960	4	4	4	1	0	0	
2	12250000	9960	3	2	2	1	0	1	
3	12215000	7500	4	2	2	1	0	1	
4	11410000	7420	4	1	2	1	1	1	

Next steps: View recommended plots

Dummy variables : Dummy Variables - Now the last column(furnishingstatus) has 3 categories i.e. furnished,semi-furnished and unfurnished. We need to convert this to numbers as well

```
table = pd.get_dummies(df['furnishingstatus'])   #add the column into table variable
table.head()
```

	furnished	semi-furnished	unfurnished
0	1	0	0
1	1	0	0
2	0	1	0
3	1	0	0
4	1	0	0

Next steps: View recommended plots

```
table = pd.get_dummies(df['furnishingstatus'], drop_first = True)  #recreate table but now c
table.head()
```

	semi-furnished	unfurnished
0	0	0
1	0	0
2	1	0
3	0	0
4	0	0

Next steps: View recommended plots

```
df = pd.concat([df, table], axis = 1)  #attach the other two columns to our data set
df.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwater
0	13300000	7420	4	2	3	1	0	0	
1	12250000	8960	4	4	4	1	0	0	
2	12250000	9960	3	2	2	1	0	1	
3	12215000	7500	4	2	2	1	0	1	

Next steps: View recommended plots

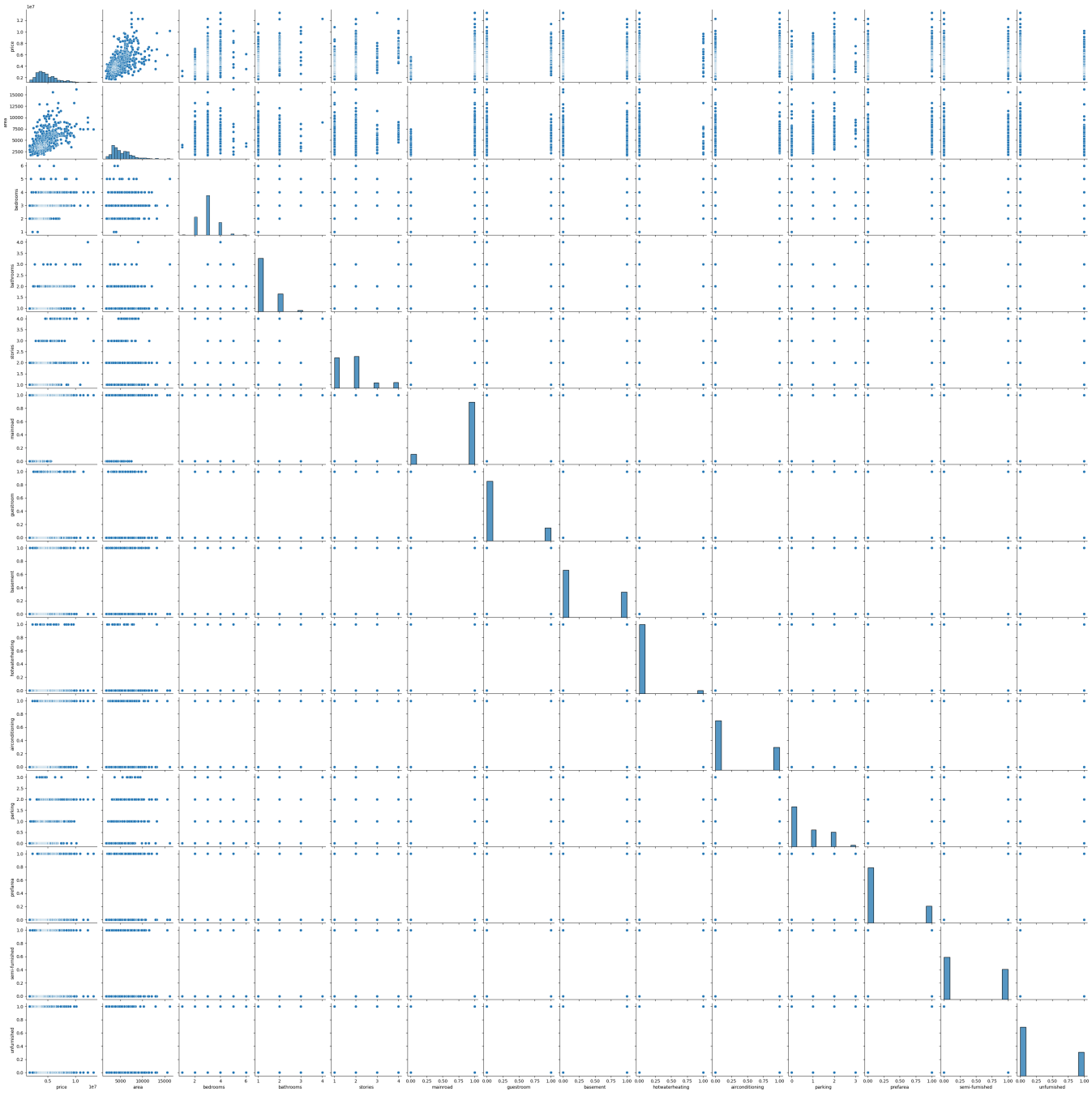
```
df.drop(['furnishingstatus'], axis = 1, inplace = True) #drop the old column from the datase
df.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwater
0	13300000	7420	4	2	3	1	0	0	
1	12250000	8960	4	4	4	1	0	0	
2	12250000	9960	3	2	2	1	0	1	
3	12215000	7500	4	2	2	1	0	1	

Next steps: View recommended plots

Plots and graphs

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


price
area
bedrooms
bathrooms
stories
mainroad
guestroom
basement
hotwaterheating
airconditioning
parking
prefarea
semi-furnished
unfurnished

Split data into training and testing data

```
from sklearn.model_selection import train_test_split
np.random.seed(0) #so data can have same values
df_train, df_test = train_test_split(df, train_size = 0.7, test_size = 0.3, random_state = 1
```

```
df_train.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwate
359	3710000	3600	3	1	1	1	0	0	
19	8855000	6420	3	2	2	1	0	0	
159	5460000	3150	3	2	1	1	1	1	
35	8080940	7000	3	2	4	1	0	0	

Next steps: View recommended plots

Scaling Training Data: MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler    #to make all the numbers to the same scale
scaler = MinMaxScaler()

var_to_scale = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking','price']

df_train[var_to_scale] = scaler.fit_transform(df_train[var_to_scale])

df_train.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hot
359	0.169697	0.155227	0.4	0.0	0.000000	1	0	0	
19	0.615152	0.403379	0.4	0.5	0.333333	1	0	0	
159	0.321212	0.115628	0.4	0.5	0.000000	1	1	1	
35	0.548133	0.454417	0.4	0.5	1.000000	1	0	0	

Next steps: View recommended plots

```
df_train.describe()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom
count	381.000000	381.000000	381.000000	381.000000	381.000000	381.000000	381.000000
mean	0.260333	0.288710	0.386352	0.136483	0.268591	0.855643	0.170604
std	0.157607	0.181420	0.147336	0.237325	0.295001	0.351913	0.376657
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.151515	0.155227	0.200000	0.000000	0.000000	1.000000	0.000000
50%	0.221212	0.234424	0.400000	0.000000	0.333333	1.000000	0.000000
75%	0.345455	0.398099	0.400000	0.500000	0.333333	1.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

```
y_train = df_train.pop('price')
x_train = df_train
y_train.head()
```

```
359    0.169697
19     0.615152
```

```
159    0.321212
35     0.548133
28     0.575758
Name: price, dtype: float64
```

using lineat regression

```
from sklearn.linear_model import LinearRegression
lm=LinearRegression()
lm.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
lm.coef_
```

```
array([ 0.23466354,  0.04673453,  0.19082319,  0.10851563,  0.05044144,
        0.03042826,  0.02159488,  0.08486327,  0.06688093,  0.06073533,
        0.05942788,  0.00092052, -0.03100561])
```

```
#values from 0 to 1
#0 model explain None of the variability
#1 model explain Entire of the variability
lm.score(x_train,y_train)
```

```
0.6814893088451202
```

```
var_to_scale = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking','price']
```

```
df_test[var_to_scale] = scaler.fit_transform(df_test[var_to_scale])
```

Run model using test data

```
y_test = df_test.pop('price')
x_test = df_test
```

```
predictions = lm.predict(x_test)
```

```
from sklearn.metrics import r2_score
r2_score(y_test, predictions)
```

```
0.5995575338728529
```

```
#AttributeError: 'Series' object has no attribute 'flatten' --to avoid this error in the nex
y_test.shape
y_test_matrix = y_test.values.reshape(-1,1)

#load actual and predecited values side by side
dframe=pd.DataFrame({'actual':y_test_matrix.flatten(),'Predicted':predictions.flatten()})
#flatten toget single axis of data (1 dimension only)


dframe.head(15)
```

	actual	Predicted
0	0.247651	0.202410
1	0.530201	0.374464
2	0.328859	0.305654
3	0.261745	0.293786
4	0.245638	0.258827
5	0.275168	0.189463
6	0.644295	0.499099
7	0.328859	0.297637
8	0.087248	0.122528
9	0.395973	0.316860
10	0.177852	0.085304
11	0.463087	0.370193
12	0.053691	0.219748
13	0.395302	0.419331
14	0.362416	0.325773

Next steps: View recommended plots

plotting graph

```
#using scatter plot compare the actual and predicted data
fig = plt.figure()
plt.scatter(y_test,predictions)
```