week04_Linear_Regression

In [2]:

```
#19IT016 MANAV_BUTANI
```

Multiple Linear Regression is basically indicating that we will be having many features Such as f1, f2, f3, f4, and our output feature f5. If we take the same example as above we discussed, suppose:

f1 is the size of the house.

f2 is bad rooms in the house.

f3 is the locality of the house.

f4 is the condition of the house and,

f5 is our output feature which is the price of the house.

Now, you can see that multiple independent features also make a huge impact on the price of the house, price can vary from feature to feature. When we are discussing multiple linear regression then the equation of simple linear regression y=A+Bx is converted to something like:

```
equation:  y = A+B1x1+B2x2+B3x3+B4x4
```

"If we have one dependent feature and multiple independent features then basically call it a multiple linear regression."

Flow of implementation

- Import required libraries
- Load dataset and Divide in Features and Target
- Clean Dataset (Remove missing values)
- Divide dataset in train-test split
- Fit the model on training data
- Check for Goodness of Fit using MSE/RMSE/R2_Score

Importing Libraries

In [3]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Why you want to apply regression on selected dataset?

Because our Y which is target variable is of continuous type so we can apply regression

In [4]:

```
df = pd.read_csv('HousePrices_HalfMil.CSV')
df.head()
```

Out[4]:

	Area	Garage	FirePlace	Baths	White Marble	Black Marble	Indian Marble	Floors	City	Solar	Electric	Fiber	Glass Doors	Swiming Pool	Garden	Prices
0	164	2	0	2	0	1	0	0	3	1	1	1	1	0	0	43800
1	84	2	0	4	0	0	1	1	2	0	0	0	1	1	1	37550
2	190	2	4	4	1	0	0	0	2	0	0	1	0	0	0	49500
3	75	2	4	4	0	0	1	1	1	1	1	1	1	1	1	50075
4	148	1	4	2	1	0	0	1	2	1	0	0	1	1	1	52400

How many total observations in data?

The dimensionality of the dataset is the number of rows x columns in the dataset. The rows represent the observations and the columns represent the variables. The shape attribute of the DataFrame class returns a tuple representing rows x columns.There are 500000 observations (rows) and 16 variables (columns) in the dataset.

In [5]:

```
print(len(df))
print(df.shape)
```

```
500000
(500000, 16)
```

How many independent variables?

In This dataset,we could say Area, Garage, FirePlace, Baths, White Marble, Black Marble, Indian Marble, Floors, City, Solar, Electric, Fiber, Glass Doors, Swiming Pool, Garden are the independent variables.

Which is dependent variable?

Prices is the dependent variable.

Which are most useful variable in estimation? Prove using correlation.

In this dataset Area, Baths, White Marble, Floors, City, Fiber, Glass Doors are useful.

In [6]:

```
df.corr()
```

Out[6]:

	Area	Garage	FirePlace	Baths	White Marble	Black Marble	Indian Marble	Floors	City	Solar	Electric	Fiber	Glass Doors	Swiming Pool	Garden	Prices
Area	1.000000	-0.000897	0.000374	-0.000398	0.002525	-0.001477	-0.001047	-0.000776	-0.003455	0.000526	-0.000128	0.000114	-0.001230	0.000610	0.001428	0.147717
Garage	-0.000897	1.000000	0.001307	-0.003647	0.000541	0.001847	-0.002385	-0.000931	0.000793	0.001480	0.000779	-0.000562	-0.002171	0.001088	-0.000669	0.100294
FirePlace	0.000374	0.001307	1.000000	0.000643	0.000952	-0.000922	-0.000030	0.000185	-0.000211	-0.000309	0.001342	0.001818	-0.000366	0.001295	0.000231	0.089139
Baths	-0.000398	-0.003647	0.000643	1.000000	0.002493	-0.002739	0.000246	-0.000880	-0.000858	-0.000755	0.001047	-0.000687	-0.001668	0.002212	0.001723	0.145087
White Marble	0.002525	0.000541	0.000952	0.002493	1.000000	-0.498893	-0.500723	0.000078	-0.000861	-0.001583	0.000558	-0.000576	-0.000402	-0.001898	0.000959	0.448154
Black Marble	-0.001477	0.001847	-0.000922	-0.002739	-0.498893	1.000000	-0.500383	-0.000368	-0.000324	0.001764	-0.000549	0.000073	0.000086	0.000713	0.000133	-0.078049
Indian Marble	-0.001047	-0.002385	-0.000030	0.000246	-0.500723	-0.500383	1.000000	0.000289	0.001184	-0.000180	-0.000010	0.000503	0.000316	0.001184	-0.001091	-0.369756
Floors	-0.000776	-0.000931	0.000185	-0.000880	0.000078	-0.000368	0.000289	1.000000	-0.000641	-0.002651	0.000082	0.001373	-0.000008	-0.000211	-0.000492	0.619451
City	-0.003455	0.000793	-0.000211	-0.000858	-0.000861	-0.000324	0.001184	-0.000641	1.000000	0.000488	0.000788	-0.002716	0.000770	0.000322	0.001207	0.233259
Solar	0.000526	0.001480	-0.000309	-0.000755	-0.001583	0.001764	-0.000180	-0.002651	0.000488	1.000000	0.001883	0.000238	-0.000817	-0.000466	-0.004263	0.008429
Electric	-0.000128	0.000779	0.001342	0.001047	0.000558	-0.000549	-0.000010	0.000082	0.000788	0.001883	1.000000	-0.000309	0.001088	0.000571	0.000772	0.052443
Fiber	0.000114	-0.000562	0.001818	-0.000687	-0.000576	0.000073	0.000503	0.001373	-0.002716	0.000238	-0.000309	1.000000	-0.002268	0.004127	-0.000023	0.484626
Glass Doors	-0.001230	-0.002171	-0.000366	-0.001668	-0.000402	0.000086	0.000316	-0.000008	0.000770	-0.000817	0.001088	-0.002268	1.000000	0.000396	0.003329	0.181973
Swiming Pool	0.000610	0.001088	0.001295	0.002212	-0.001898	0.000713	0.001184	-0.000211	0.000322	-0.000466	0.000571	0.004127	0.000396	1.000000	-0.000191	0.001787
Garden	0.001428	-0.000669	0.000231	0.001723	0.000959	0.000133	-0.001091	-0.000492	0.001207	-0.004263	0.000772	-0.000023	0.003329	-0.000191	1.000000	0.001540
Prices	0.147717	0.100294	0.089139	0.145087	0.448154	-0.078049	-0.369756	0.619451	0.233259	0.008429	0.052443	0.484626	0.181973	0.001787	0.001540	1.000000

Data Preparation

Finding Missing Values

In [7]:

```
df.head()
```

Out[7]:

	Area	Garage	FirePlace	Baths	White Marble	Black Marble	Indian Marble	Floors	City	Solar	Electric	Fiber	Glass Doors	Swiming Pool	Garden	Prices
0	164	2	0	2	0	1	0	0	3	1	1	1	1	0	0	43800
1	84	2	0	4	0	0	1	1	2	0	0	0	1	1	1	37550
2	190	2	4	4	1	0	0	0	2	0	0	1	0	0	0	49500
3	75	2	4	4	0	0	1	1	1	1	1	1	1	1	1	50075
4	148	1	4	2	1	0	0	1	2	1	0	0	1	1	1	52400

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500000 entries, 0 to 499999
Data columns (total 16 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Area           500000 non-null  int64
 1   Garage         500000 non-null  int64
 2   FirePlace      500000 non-null  int64
 3   Baths          500000 non-null  int64
 4   White Marble   500000 non-null  int64
 5   Black Marble   500000 non-null  int64
 6   Indian Marble  500000 non-null  int64
 7   Floors         500000 non-null  int64
 8   City           500000 non-null  int64
 9   Solar          500000 non-null  int64
 10  Electric       500000 non-null  int64
 11  Fiber          500000 non-null  int64
 12  Glass Doors    500000 non-null  int64
 13  Swiming Pool   500000 non-null  int64
 14  Garden         500000 non-null  int64
 15  Prices         500000 non-null  int64
dtypes: int64(16)
memory usage: 61.0 MB
```

In [9]:

```
df.isnull().sum()
```

Out[9]:

```
Area             0
Garage           0
FirePlace        0
Baths            0
White Marble     0
Black Marble     0
Indian Marble    0
Floors           0
City             0
Solar            0
Electric         0
Fiber            0
Glass Doors      0
Swiming Pool     0
Garden           0
Prices           0
dtype: int64
```

Prepare the data for training

In [10]:

```
X = df.iloc[:,0:-1]
Y = df.iloc[:,-1]
```

Split the data into training and testing sets

In [11]:

```
from sklearn.model_selection import train_test_split

# splits the training and test data set in 80% : 20%
# assign random_state to any value.This ensures consistency.
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(400000, 15)
(100000, 15)
(400000,)
(100000,)
```

Train the model using sklearn LinearRegression

In [12]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

lin_model = LinearRegression()
lin_model.fit(X_train, Y_train)
```

Out[12]:

```
LinearRegression()
```

Check for Goodness of Fit using MSE/RMSE/R2_Score

In [14]:

```
# model evaluation for training set

y_train_predict = lin_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("--------------------------------------")
print(f'RMSE is {rmse}')
print(f'R2 score is {r2}')
print("\n")

# model evaluation for testing set

y_test_predict = lin_model.predict(X_test)
# root mean square error of the model
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))

# r-squared score of the model
r2 = r2_score(Y_test, y_test_predict)

print("The model performance for testing set")
print("--------------------------------------")
print(f'RMSE is {rmse}')
print(f'R2 score is {r2}')
```

```
The model performance for training set
--------------------------------------
RMSE is 8.715951295532953e-11
R2 score is 1.0


The model performance for testing set
--------------------------------------
RMSE is 8.706073075697601e-11
R2 score is 1.0
```

In [15]:

```
# Check total co-eficients of model
# Here, it is 15 which indicates this data fits to  linear regression models
print(len(lin_model.coef_),lin_model.intercept_)
```

```
15 7333.333333333452
```

Quantify goodness of your model and discuss steps taken for improvement (RMSE, MSE, R2Score).

Regression using SKlearn Api is good as it gives highest R2 score and low RMSE.