19. Train Test Split in Dataset

1. splitting the data

- The data is split into train and test in supervised learning
- there is no need to split the data into train and test in unsupervised learning

2. depedent and independent variables

- separte the data according to dependent and independent variables (i.e. convert the data into input and output)

In [1]:
```python
import pandas as pd
```

In [2]:
```python
dataset = pd.read_csv("boston.csv")
dataset.head(3)
```

Out[2]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03

Separate the data into input and output

In [10]:
```python
# dataset.iloc [number of rows:number of columns]
input_data = dataset.iloc[:,:-1]
input_data.head(3)
```

Out[10]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03

In [18]:
```python
dataset.shape
```

Out[18]:
```
(506, 14)
```

In [11]:
```python
output_data = dataset['medv']
output_data.head(3)
```

Out[11]:
```
0    24.0
1    21.6
2    34.7
Name: medv, dtype: float64
```

Split the data into training and test dataset

In [14]:
```
from sklearn.model_selection import train_test_split
```

this will split data into 4 parts:

1. input training data, x_train
2. input test data, x_test
3. output training data, y_train
4. output test data, y_test

In [16]:
```
x_train, x_test, y_train, y_test = train_test_split(input_data, output_data, test_s
```

In [17]:
```
x_test
```

Out[17]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lst
30	1.13081	0.0	8.14	0	0.5380	5.713	94.1	4.2330	4	307	21.0	360.17	22.6
377	9.82349	0.0	18.10	0	0.6710	6.794	98.8	1.3580	24	666	20.2	396.90	21.2
79	0.08387	0.0	12.83	0	0.4370	5.874	36.6	4.5026	5	398	18.7	396.06	9.1
321	0.18159	0.0	7.38	0	0.4930	6.376	54.3	4.5404	5	287	19.6	396.90	6.8
204	0.02009	95.0	2.68	0	0.4161	8.034	31.9	5.1180	4	224	14.7	390.55	2.8
...	...	...	...	...	...	...	...	...	...	...	...	...	
12	0.09378	12.5	7.87	0	0.5240	5.889	39.0	5.4509	5	311	15.2	390.50	15.7
192	0.08664	45.0	3.44	0	0.4370	7.178	26.3	6.4798	5	398	15.2	390.49	2.8
288	0.04590	52.5	5.32	0	0.4050	6.315	45.6	7.3172	6	293	16.6	396.90	7.6
4	0.06905	0.0	2.18	0	0.4580	7.147	54.2	6.0622	3	222	18.7	396.90	5.3
441	9.72418	0.0	18.10	0	0.7400	6.406	97.2	2.0651	24	666	20.2	385.96	19.5

127 rows × 13 columns

In [23]:
```
dataset.shape
```

Out[23]:
```
((506, 14), (379,))
```

In [24]:
```
x_train.shape, y_train.shape
```

Out[24]:
```
((379, 13), (379,))
```

In [25]:
```python
x_test.shape, y_train.shape
```

Out[25]:
```
((127, 13), (379,))
```

In [ ]:
```python

```

20. Regression Analysis

- Depedning on type of data, On the basis of outcome, you decided whether to do classification or regression analysis for prediction
- outcome: continuous -> regression analysis

Regression Analysis - Real world applications:

1. Prediction of rain using temperature and other factors
2. Determining of Market trends
3. Prediction of road accidents due to rash driving

Regression analysis

- Linear Regression: Used when input and output have linear relationship
- Non-linear regression: used when input and output have non-linear relationship

Linear Regression:

1. Linear regression
2. Multi-linear regression
3. Lasso regression
4. Ridge regression

Non-Linear Regression:

1. Polynomial regression
2. Decision tree regression
3. Random Forest regression
4. Suppor vector regression
5. K-Neartest Neighbour

20.1 Linear Regression Algorithm (Simple Linear)

- Linear regression is used when independent/input variable is single y = mx + c

- m = slope of line (angle between x and y=axic)
- c = intercept (at how much distance the line is farther from y-axis)

m = x2 - x1 / y2 - y1

- m is +ve if anlge < 90
- m is -ve if anlge > 90
- m is 0 if angle = 0

21. Linear Regression (Practical)

In [21]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

In [11]:
```
dataset = pd.read_csv(r'Data/placement.csv')
dataset.head(3)
```

Out[11]:

	cgpa	package
0	6.89	3.26
1	5.12	1.98
2	7.82	3.25

In [12]:
```
dataset.isnull().sum()
```

Out[12]:
```
cgpa       0
package    0
dtype: int64
```

- data has to be in multidimensional or 2 dimentional at least

In [13]:
```
x = dataset["cgpa"]
x.ndim
```

Out[13]:
```
1
```

- So we will convert this data into 2 dimensional data:

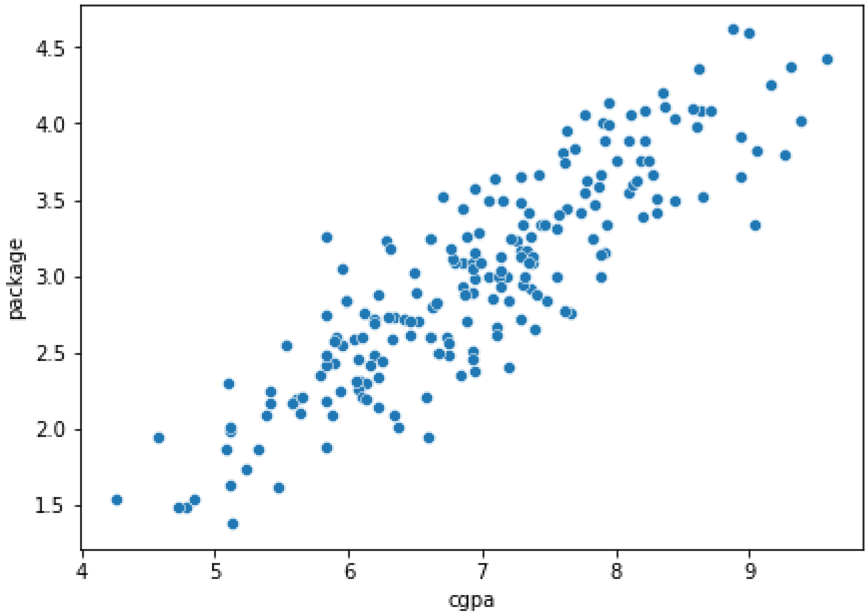
In [14]:
```
x = dataset[["cgpa"]]
x.ndim
```

Out[14]:
```
2
```

In [15]:
```
y = dataset['package']
```

- Before applying linear regression, check that if your data is following linearity or not

In [20]:
```
plt.figure(figsize=(7,5))
sns.scatterplot(x='cgpa', y='package', data=dataset)
plt.show()
```



- You can see the data is following simple linearity

In [22]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state
```

In [26]:
```
# y = mx + c
from sklearn.linear_model import LinearRegression
```

In [27]:
```
lr = LinearRegression()
# fit will train the data to fit linear equation,
# y = mx + c, this will search for best m and c value to train the data on this lin
lr.fit(x_train, y_train)
```

Out[27]:
```
▾ LinearRegression
LinearRegression()
```

- Now our model is trained now, and ready for testing

In [30]:
```
lr.predict([[6.89]])
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but LinearRegression was
fitted with feature names
  warnings.warn(
```

Out[30]:
```
array([2.92962016])
```

In [31]:
```
dataset.head(3)
```

Out[31]:

	cgpa	package
0	6.89	3.26
1	5.12	1.98
2	7.82	3.25

- To check if prediction is model is good or now, we will use **accuracy score**

In [33]:
```
lr.score(x_test, y_test)*100
```

Out[33]:
```
77.30984312051673
```

- To improve accuracy we will change random_state value in following code and see if the model accuracy has increased:
- x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=42)

To find the equation manually

In [41]:
```
# y = mx + c
```

In [36]:
```
m = lr.coef_
m
```

Out[36]:
```
array([0.57425647])
```

In [37]:
```
c = lr.intercept_
c
```

Out[37]:
```
-1.0270069374542108
```

In [40]:
```
y = (m * 6.89) + c
y
```

Out[40]:
```
array([2.92962016])
```

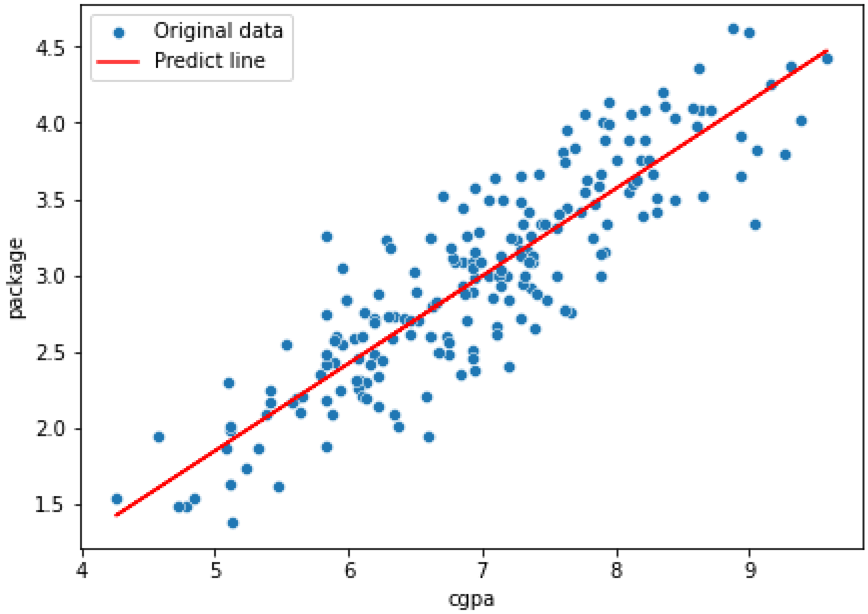
--

To draw prediction line

In [46]:
```
# y_pred = lr.predict([['cgpa']]) = y_pred = lr.predict(x)
y_pred = lr.predict(x)
```

In [55]:
```
plt.figure(figsize=(7,5))
sns.scatterplot(x='cgpa', y='package', data=dataset)
# plt.plot(x,y)
plt.plot(dataset['cgpa'], y_pred, c='red')
plt.legend(["Original data", "Predict line"])
```

```
plt.savefig(r"Generated_images/predict.jpg")
plt.show()
```



22. Multiple Linear Regression

- Used when input are more than one
- Multiple linear regression is an extension of simple linear regression as it takes more than one predictor variable to predict the response variable
- **y = m1x1 + m2x2 + m3x3 + ... mnxn + c**

In [102…
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [103…
```
dataset = pd.read_csv(r'Data/salary_data.csv')
dataset.head(3)
```

Out[103…

	Age	Experience	Salary
0	53	21	274930.685866
1	39	19	217753.696272
2	32	19	166660.977435

In [104…
```
dataset.shape
```

Out[104…
```
(1000, 3)
```

In [105…
```
dataset.isnull().sum()
```

Out[105…
```
Age           0
Experience    0
Salary        0
dtype: int64
```

In [106…
```
dataset.shape
```

Out[106…
```
(1000, 3)
```

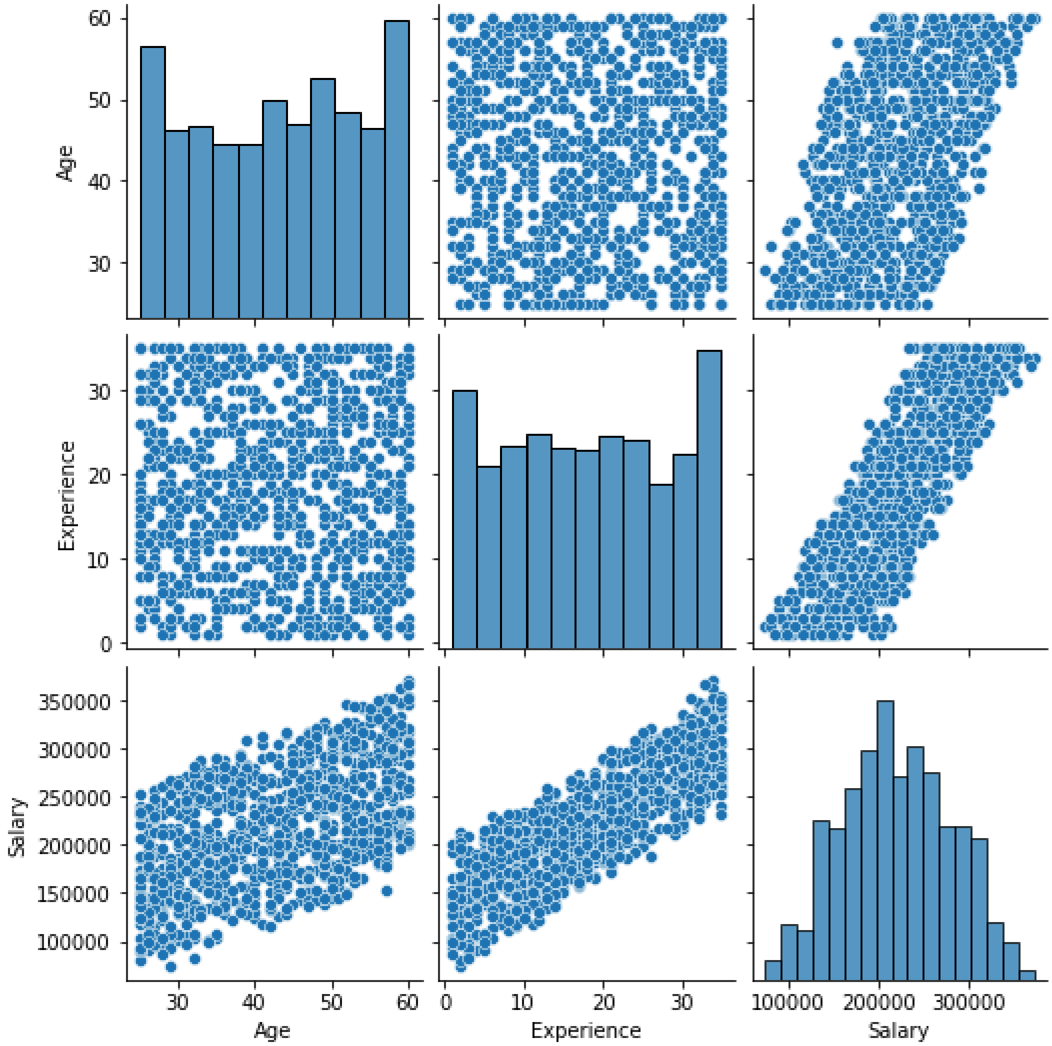
Also an important step before applying model is to check if your data needs scaling (if huge difference in data values)

- but for this exercise, we are not going to check it as we can see no much difference in values of age and experience

To Check if the data is linear before applying linear regression model

In [107…
```
sns.pairplot(data=dataset)
plt.show()
```

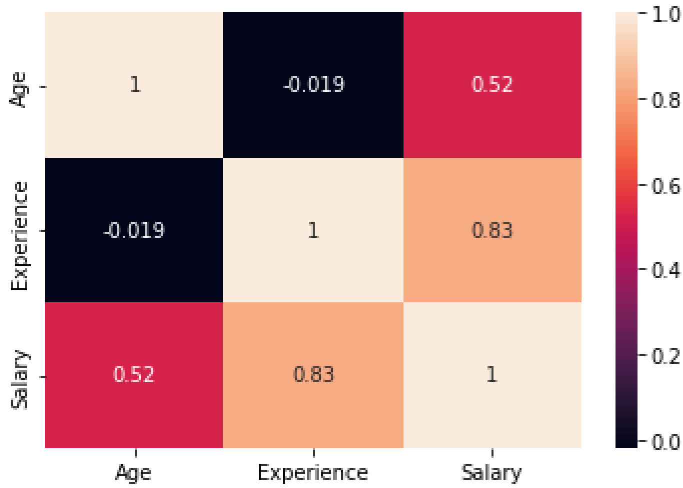
```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\axis
grid.py:123: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Use correlation frunction to check if the data is linear

In [108…

```
# annote = True -> to check correlation number
sns.heatmap(data=dataset.corr(), annot=True)
plt.show()
```



Both of above graph shows correlation between output (salary) and inputs (age and expereince)

In [109…
```
# Separate features and target
x = dataset[['Age', 'Experience']]
y = dataset['Salary']
```

In [110…
```
# Check the shape of X and y
print(X.shape)  # Should be (1000, 2)
print(y.shape)  # Should be (1000,)
```

```
(1000, 2)
(1000,)
```

Train the model

In [111…
```
from sklearn.model_selection import train_test_split
```

In [112…
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.20,random_state
```

Build Model

In [113…
```
from sklearn.linear_model import LinearRegression
```

In [114…
```
lr = LinearRegression()
```

In [115…
```
lr.fit(x_train, y_train)
```

Out[115…
```
▾ LinearRegression
LinearRegression()
```

In [116…
```
dataset.shape
```

Out[116…
```
(1000, 3)
```

Test Model

In [118…

```
lr.score(x_test, y_test)
```

Out[118…

```
0.9738985132159785
```

Make Prediction

In [120…

```
lr.predict(x_test)
```

Out[120…

```
array([127673.47833523, 263638.47930118, 350142.08171943, 145791.96000071,
       229782.58458827, 217703.59681128, 207200.43711274, 250171.09339619,
       167062.09782308, 260806.16230738, 209338.55504254, 244319.02945815,
       207387.86706319, 200973.5132738 , 249421.37359439, 290122.0027354 ,
       289052.9437705 , 186999.35825527, 144722.90103581, 156239.59896145,
       211101.81307144, 145604.53005026, 309309.54336578, 279618.84303686,
       303776.81859083, 189269.38539771, 169894.41481688, 184673.81037502,
       145042.24019891, 169949.93555468, 238279.53556966, 181091.77357942,
       286727.39589025, 166930.18861044, 260674.25309473, 202792.2920405 ,
       134594.60123817, 240549.5627121 , 295974.06667344, 199210.2552449 ,
       322082.73020676, 262624.94107408, 188575.18633371, 140071.80527531,
       207387.86706319, 157870.9477777 , 239348.59453456, 260618.73235693,
       202604.86209005, 216821.96779683, 258161.27526403, 305033.30750618,
       251427.58231154, 170081.84476733, 206693.66799919, 171525.76363313,
       239723.45443546, 154851.20083345, 222861.46168533, 268477.00501213,
       270747.03215457, 315349.03725427, 306102.36647108, 222861.46168533,
       235579.12778851, 184673.81037502, 265964.02718143, 191088.16416441,
       321895.30025631, 186117.72924082, 197259.56726555, 173851.31151338,
       185555.43938947, 308934.68346488, 245388.08842305, 291378.49165075,
       217516.16686083, 207894.63617674, 253003.41038999, 291191.0617003 ,
       198328.62623045, 292072.69071475, 235579.12778851, 264388.19910298,
       316230.66626872, 137801.77813287, 211851.53287324, 211983.44208588,
       187318.69741836, 186249.63845346, 197766.3363791 , 265457.25806788,
       195871.16913756, 270559.60220412, 115594.49055824, 292260.1206652 ,
       227269.60675757, 232934.24074516, 136732.71916797, 287796.45485515,
       236141.41763986, 205249.74913339, 241486.71246435, 221736.88198262,
       228151.23577202, 266151.45713188, 120377.49553139, 200973.5132738 ,
       289052.9437705 , 299368.67351859, 133525.54227327, 207200.43711274,
       338063.09394245, 247338.7764024 , 184111.52052366, 176308.76860627,
       225506.34872867, 106535.2497255 , 194614.68022221, 182723.12239567,
       179890.80540187, 345171.64679584, 149131.04610805, 243062.5405428 ,
       157683.51782725, 202042.5722387 , 231677.75182981, 283895.07889646,
       315349.03725427, 332211.0300044 , 281250.19185311, 196884.70736465,
       242743.2013797 , 252628.55048909, 107604.3086904 , 283707.648946  ,
       203861.3510054 , 291378.49165075, 232052.61173071, 218904.56498883,
       350142.08171943, 161210.03388504, 204368.12011894, 228713.52562337,
       238973.73463365, 188012.89648236, 102446.44381636, 268102.14511122,
       251240.15236109, 144160.61118446, 249796.23349529, 249421.37359439,
       146861.01896561, 206506.23804874, 302200.99051239, 169387.64570333,
       138121.11729596, 146111.29916381, 271121.89205547, 282638.58998111,
       124278.87149008, 310003.74242978, 151269.16403785, 258480.61442713,
       340013.78192179, 256529.92644778, 320319.47217787, 301131.93154749,
       248220.40541685, 278417.87485931, 155920.25979835, 102446.44381636,
       235953.98768941, 195308.8792862 , 128367.67739923, 304845.87755573,
       171900.62353403, 207950.15691454, 287983.8848056 , 263131.71018763,
       241806.05162745, 225506.34872867, 162091.66289949, 151831.4538892 ,
       114338.00164289, 255460.86748288, 236516.27754076, 296348.92657434,
       251240.15236109, 276786.52604306, 297230.55558879, 283707.648946  ,
       168880.87658979, 277536.24584487, 222299.17183397, 275210.69796462,
       325984.10616546, 213934.13006523, 299875.44263214, 297737.32470234])
```

In [ ]:

23. Polynomial Regression

- When data is not following any linearity
- Polynomial regression is a regression algorithm that models the relationshi between a dependent(y)and independent variable(x) as nth degree polynomial
- **Y = b0 + b1x1 + b2x1^2 + b3x1^3 + ... + bnx1^n**

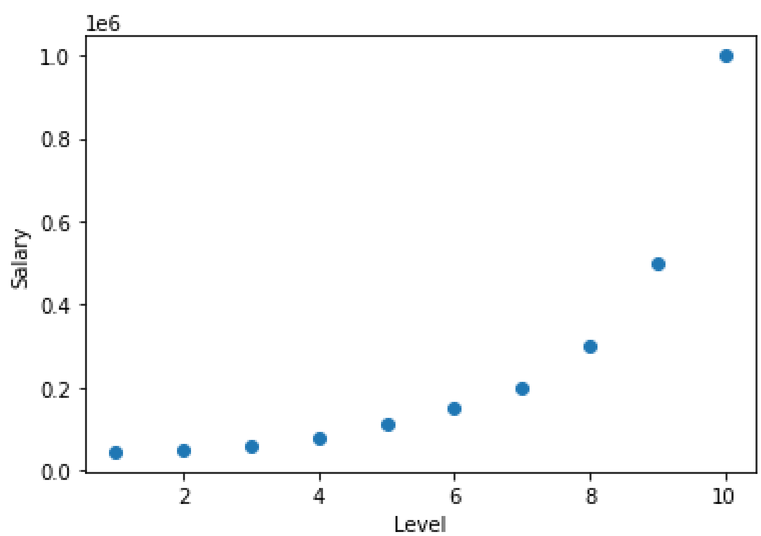
In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [4]:
```
dataset = pd.read_csv(r'Data/polynomial.csv')
dataset.head(3)
```

Out[4]:

	Level	Salary
0	1	45000
1	2	50000
2	3	60000

In [8]:
```
plt.scatter(dataset["Level"], dataset["Salary"])
plt.xlabel("Level")
plt.ylabel("Salary")
plt.show()
```



- So this graph is showing that data is not linear

To check correlation

```
In [6]: dataset.corr()
```

Out[6]:

	Level	Salary
Level	1.000000	0.817949
Salary	0.817949	1.000000

Separate data into input and output

```
In [9]: # Remember that data should be multidimensional
x = dataset[['Level']]
y = dataset['Salary']
```

Convert data into polynomial nature

```
In [10]: from sklearn.preprocessing import PolynomialFeatures
```

```
In [31]: # Change the degree to 2 and so on, depend on your need, to make the model more acc
pf = PolynomialFeatures(degree=2)
pf.fit(x)
x = pf.transform(x)
x
```

```
Out[31]: array([[1.000e+00, 1.000e+00, 1.000e+00, 1.000e+00, 1.000e+00, 1.000e+00,
         1.000e+00, 1.000e+00, 1.000e+00, 1.000e+00],
        [1.000e+00, 1.000e+00, 2.000e+00, 4.000e+00, 1.000e+00, 2.000e+00,
         4.000e+00, 4.000e+00, 8.000e+00, 1.600e+01],
        [1.000e+00, 1.000e+00, 3.000e+00, 9.000e+00, 1.000e+00, 3.000e+00,
         9.000e+00, 9.000e+00, 2.700e+01, 8.100e+01],
        [1.000e+00, 1.000e+00, 4.000e+00, 1.600e+01, 1.000e+00, 4.000e+00,
         1.600e+01, 1.600e+01, 6.400e+01, 2.560e+02],
        [1.000e+00, 1.000e+00, 5.000e+00, 2.500e+01, 1.000e+00, 5.000e+00,
         2.500e+01, 2.500e+01, 1.250e+02, 6.250e+02],
        [1.000e+00, 1.000e+00, 6.000e+00, 3.600e+01, 1.000e+00, 6.000e+00,
         3.600e+01, 3.600e+01, 2.160e+02, 1.296e+03],
        [1.000e+00, 1.000e+00, 7.000e+00, 4.900e+01, 1.000e+00, 7.000e+00,
         4.900e+01, 4.900e+01, 3.430e+02, 2.401e+03],
        [1.000e+00, 1.000e+00, 8.000e+00, 6.400e+01, 1.000e+00, 8.000e+00,
         6.400e+01, 6.400e+01, 5.120e+02, 4.096e+03],
        [1.000e+00, 1.000e+00, 9.000e+00, 8.100e+01, 1.000e+00, 9.000e+00,
         8.100e+01, 8.100e+01, 7.290e+02, 6.561e+03],
        [1.000e+00, 1.000e+00, 1.000e+01, 1.000e+02, 1.000e+00, 1.000e+01,
         1.000e+02, 1.000e+02, 1.000e+03, 1.000e+04]])
```

Split data into train and test

```
In [13]: from sklearn.model_selection import train_test_split
```

```
In [18]: x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state
```

Build model using polynomial regression

In [19]:
```
from sklearn.linear_model import LinearRegression
```

In [20]:
```
lr = LinearRegression()
lr.fit(x_train, y_train)
```

Out[20]:
```
▾ LinearRegression
LinearRegression()
```

Check model accuracy

In [22]:
```
lr.score(x_test, y_test)*100
```

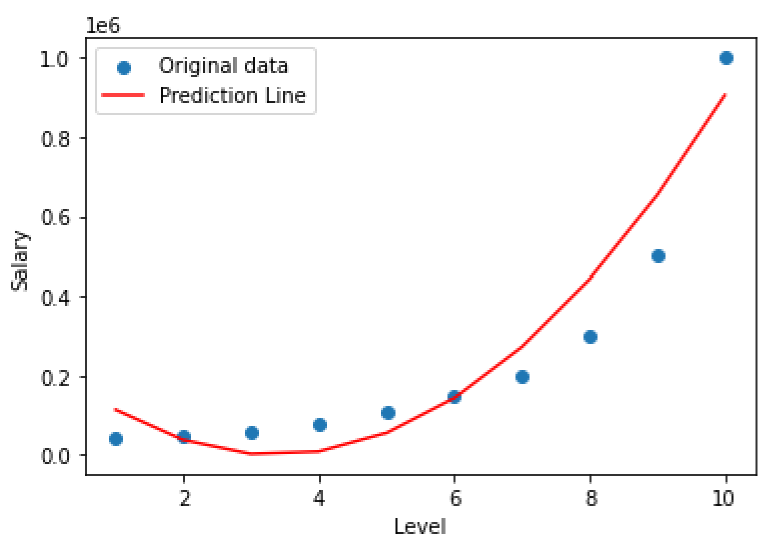
Out[22]:
```
76.66492889299911
```

Draw Prediction Line

In [23]:
```
pred = lr.predict(x)
pred
```

Out[23]:
```
array([114155.94968909,  38027.48728095,   2903.12323346,   8782.85754664,
        55666.69022046, 143554.62125495, 272446.65065008, 442342.77840588,
       653243.00452233, 905147.32899944])
```

In [26]:
```
plt.scatter(dataset["Level"], dataset["Salary"])
plt.plot(dataset['Level'], pred, c='red')
plt.xlabel("Level")
plt.ylabel("Salary")
plt.legend(["Original data", "Prediction Line"])
plt.show()
```



Remember, before testing any data, you have to convert it into polynomial feature, then use it for predcition, like below:

In [29]:
```
test = pf.transform([[9]])
test
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but PolynomialFeatures w
as fitted with feature names
  warnings.warn(
```

Out[29]:
```
array([[ 1.,  9., 81.]])
```

In [30]:
```
lr.predict(test)
```

Out[30]:
```
array([653243.00452233])
```

Beware of overfitting / underfitting, your model should not be that much accurate, so it go to overfitting - rahter it should be best fit

In [ ]:
```

```

24. Cost Function

What is Cost Function:

- A cost function is an important parameter that determines how well a machine learning model performs for a given dataset
- Cost function is a measure of how wrong the model is in estimating the relationship b/w x(input) and y(ouput) parameter.
- With the help of cost function, you draw best fit line
- Cost function and loss functions are both functions of error - to make the error minmum from the best fit line

Types of cost function:

- Regression cost function
- Classification cost function

1) Regression Cost Function:

- Regression models are used to make a prediction for the continuous variables.

1. MSE (Mean Square Error)
2. RMSE (Root Mean Square Error)
3. MAE (Mean Absolute Error)
4. R^2 Accuracy

2) Binary Classification Cost Function: Classification models are used to make predictions of categorical variables, such as predicitions for 0 or 1, cat or dog, etc.

3) Multi-class Classification Cost Function: A multi-class classification cost function is used in the classification problems for which instances are allocated to one of more than two classes. Binary Cross Entropy Cost Function or Log Loss Function

24.1 Regression Cost Function

No description has been provided for this image

- red line represents prediction line
- blue points represent original data
- red triangles represent error

- please note that the error value should be minimum
- For this we use **cost function to make the error minimum from prediction line**

24.2 Mean Square Error

Mean Sqaure Error (MSE) is the mean squared difference b/w the actual and predicted values. MSE penalizes high errors caused by outliers by squaring the error. MSE is also known as **L2 Loss**.

The Mean Squared Error (MSE) is calculated as:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$Whereas : -y_i = Originalvalue - \hat{y}_i = Predictedvalue - n = numberofrows$$

Advantages of using MSE are:

1. It is defferentiable

Disadvantages of using MSE:

1. When outlier is present in data, it will increase exponentially when squaring took place, so it will give wrong predictions
2. The data does not remain in original form, rather available in sqaured form, and also if original data is in cm. then it will change the unit in cm^2 as well. So the data as well as units will not be in original format.

In [ ]:

```
```

The differentiation of (y = mx + c) with respect to (x) is:

$$\frac{d}{dx}(y) = \frac{d}{dx}(mx + c) = m$$

In this differentiation:

- (\frac{d}{dx}(y)) represents the derivative of (y) with respect to (x).
- (\frac{d}{dx}(mx + c)) is the derivative of the function (mx + c).
- The result (m) is the slope of the line, which is constant in this linear equation.

The update formula for finding **m(new)** is given by:

$$M_{\text{new}} = M_{\text{old}} - \lambda \left(\frac{dz}{dm} \right)$$

24.3 Mean Absolute Error

Mean Absolute Error (MAE) is the mean absolute difference b/w the actual values and the predicted values. MAE is more robut to outliers. the insensitivity to outliers is b/c it does not penalize high errors caused by outliers.

The Mean Absolute Error (MAE) is calculated as:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Advatages of MAE:

1. Error remains in original form
2. It treats outlier well

Disadvantages are:

1. This is not differentiable equation,

24.4 Root Mean Sqaured Error

Root Mean Squared Error (RMSE) is the root sqaured mean of the difference b/w actual and predicted values. RMSE can be used in situations where we want to penalize high errors but not as much as MSE does.

The Root Mean Square Error (RMSE) is calculated as:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

24.5 How to Find Best Fit Line

- For finding best line:

1. Keep the error (loss) minimum (Which will be calculated through cost function)
2. Thorugh quardratic equation, gradient descent, we take the minimum value

No description has been provided for this image

No description has been provided for this image

- loss function will be minimum after looking for best m (slope) and c (intercept) values in $y = mx + c$
- m: donates angle
- c: donates intercept at y-axis.
- this whole process is called gradient descent technique

In [ ]:
```

```

25 Regularization Technique

L1 (Lasso Regularization) L2 (Ridge Regularization)

- Used in linear regression mostly
- This is a form of regression, that constraints/regularizes or shrinks the coefficients estimates towards zero
- This technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting.
- Regularization can achive this motive with 2 techniques:

1. Ridge Regularization/L2
2. Lasso Regularization/L1

- it helps in feature selection
- it helps reducing overfitting
- it removes the data with smaller coefficients or unwanted columns or columns/data which will have neglibile impact on the final outcome

25.1 Regularization Technique (Lasoo Regularization/L1)

- This is a regularization technique used in feature selection using a **shrinkage method** also referred as the **penalized regression method**.
- Lasso regression magnitude of coefficient can be exactly zero

The cost function is defined as:

$$\text{Cost Function} = \text{Loss} + \lambda \sum_{i=1}^{n} \|w_i\|$$

Loss= sum of squared residual, **lambda** = penalty, **w** = slope of the curve

- It helps in feature selection
- It makes the column (feature) zero which do not have function in the model

No description has been provided for this image

- the black line (lambda mod(w)) shifts towards zero iteratively

25.2 Regularization Technique (Ridge Regularization/L2)

- it is called overfitting regularization technique and it redueces overfitting

Ridge regression, also known as L2 regularization, is an extension to linear regression that introduces a regularization term to redue model complexity and **help prevent overfitting**. Ridge Regression is working value/magnitude of coefficients is almost equal to zero

Its cost function is defined as:

$$\text{Cost Function} = \text{Loss} + \lambda \sum_{i=1}^{n} \|w_i\|^2$$

Loss= sum of squared residual, **lambda** = penalty, **w** = slope of the curve

No description has been provided for this image

- it will not make exactly zero, but bring it towards zero
- L2 also reduces computational power, means reduces complexity of problem, it speeds up the model building

In [ ]:

26. Regularization Technique (Practical)

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [3]:
```
dataset = pd.read_csv(r'Data/housing.csv')
dataset.head(3)
```

Out[3]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating
0	7420	4	2	3	yes	no	no	no
1	8960	4	4	4	yes	no	no	no
2	9960	3	2	2	yes	no	yes	no

In [4]:
```
dataset.isnull().sum()
```

Out[4]:
```
area                0
bedrooms            0
bathrooms           0
stories             0
mainroad            0
guestroom           0
basement            0
hotwaterheating     0
airconditioning     0
parking             0
prefarea            0
furnishingstatus    0
price               0
dtype: int64
```

Encoding the Data into Numerical Form

In [5]:
```
en_data = dataset[['mainroad','guestroom', 'basement', 'hotwaterheating', 'aircondi
en_data.head(3)
```

Out[5]:

	mainroad	guestroom	basement	hotwaterheating	airconditioning	prefarea	furnishing
0	yes	no	no	no	yes	yes	fur
1	yes	no	no	no	yes	no	fur
2	yes	no	yes	no	no	yes	semi-fur

In [6]:
```python
pd.get_dummies(en_data)
```

Out[6]:

	mainroad_no	mainroad_yes	guestroom_no	guestroom_yes	basement_no	basement_y
0	0	1	1	0	1	
1	0	1	1	0	1	
2	0	1	1	0	0	
3	0	1	1	0	0	
4	0	1	0	1	0	
...	...	...	...	...	...	
540	0	1	1	0	0	
541	1	0	1	0	1	
542	0	1	1	0	1	
543	1	0	1	0	1	
544	0	1	1	0	1	

545 rows × 15 columns

In [7]:
```python
pd.get_dummies(en_data).info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 15 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   mainroad_no                      545 non-null    uint8
 1   mainroad_yes                     545 non-null    uint8
 2   guestroom_no                     545 non-null    uint8
 3   guestroom_yes                    545 non-null    uint8
 4   basement_no                      545 non-null    uint8
 5   basement_yes                     545 non-null    uint8
 6   hotwaterheating_no               545 non-null    uint8
 7   hotwaterheating_yes              545 non-null    uint8
 8   airconditioning_no               545 non-null    uint8
 9   airconditioning_yes              545 non-null    uint8
 10  prefarea_no                      545 non-null    uint8
 11  prefarea_yes                     545 non-null    uint8
 12  furnishingstatus_furnished       545 non-null    uint8
 13  furnishingstatus_semi-furnished  545 non-null    uint8
 14  furnishingstatus_unfurnished     545 non-null    uint8
dtypes: uint8(15)
memory usage: 8.1 KB
```

In [8]:
```python
from sklearn.preprocessing import OneHotEncoder
```

In [9]:
```
ohe = OneHotEncoder()
ohe.fit_transform(en_data)
```

Out[9]:
```
<545x15 sparse matrix of type '<class 'numpy.float64'>'
	with 3815 stored elements in Compressed Sparse Row format>
```

In [10]:
```
ohe=OneHotEncoder()
arr = ohe.fit_transform(en_data).toarray()
arr
```

Out[10]:
```
array([[0., 1., 1., ..., 1., 0., 0.],
       [0., 1., 1., ..., 1., 0., 0.],
       [0., 1., 1., ..., 0., 1., 0.],
       ...,
       [0., 1., 1., ..., 0., 0., 1.],
       [1., 0., 1., ..., 1., 0., 0.],
       [0., 1., 1., ..., 0., 0., 1.]])
```

In [11]:
```
pd.DataFrame(arr, columns=['mainroad_Yes','mainroad_No','guestroom_Yes', 'guestroom
```

Out[11]:

	mainroad_Yes	mainroad_No	guestroom_Yes	guestroom_No	basement_Yes	basement
0	0.0	1.0	1.0	0.0	1.0	
1	0.0	1.0	1.0	0.0	1.0	
2	0.0	1.0	1.0	0.0	0.0	
3	0.0	1.0	1.0	0.0	0.0	
4	0.0	1.0	0.0	1.0	0.0	
...	...	...	...	...	...	
540	0.0	1.0	1.0	0.0	0.0	
541	1.0	0.0	1.0	0.0	1.0	
542	0.0	1.0	1.0	0.0	1.0	
543	1.0	0.0	1.0	0.0	1.0	
544	0.0	1.0	1.0	0.0	1.0	

545 rows × 15 columns

In [12]:
```
ohe = OneHotEncoder(drop='first')
ar = ohe.fit_transform(en_data).toarray()
ar
```

Out[12]:
```
array([[1., 0., 0., ..., 1., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 1., ..., 1., 1., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 1.]])
```

In [13]:
```
pd.DataFrame(arr, columns=['mainroad_Yes','mainroad_No','guestroom_Yes', 'guestroom
```

Out[13]:

	mainroad_Yes	mainroad_No	guestroom_Yes	guestroom_No	basement_Yes	basement
0	0.0	1.0	1.0	0.0	1.0	
1	0.0	1.0	1.0	0.0	1.0	
2	0.0	1.0	1.0	0.0	0.0	
3	0.0	1.0	1.0	0.0	0.0	
4	0.0	1.0	0.0	1.0	0.0	
...	...	...	...	...	...	
540	0.0	1.0	1.0	0.0	0.0	
541	1.0	0.0	1.0	0.0	1.0	
542	0.0	1.0	1.0	0.0	1.0	
543	1.0	0.0	1.0	0.0	1.0	
544	0.0	1.0	1.0	0.0	1.0	

545 rows × 15 columns

In [14]:
```
ohe = OneHotEncoder(drop='first')
ar = ohe.fit_transform(en_data).toarray()
ar
```

Out[14]:
```
array([[1., 0., 0., ..., 1., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 1., ..., 1., 1., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 1.]])
```

In [15]:
```
ar.shape
```

Out[15]:
```
(545, 8)
```

In [16]:
```
encoded_data = pd.DataFrame(ar, columns=['mainroad_Yes','guestroom_Yes', 'basement_
```

In [17]:
```
encoded_data
```

Out[17]:

	mainroad_Yes	guestroom_Yes	basement_Yes	hotwaterheating_Yes	airconditioning_Yes
0	1.0	0.0	0.0	0.0	1.0
1	1.0	0.0	0.0	0.0	1.0
2	1.0	0.0	1.0	0.0	0.0
3	1.0	0.0	1.0	0.0	1.0
4	1.0	1.0	1.0	0.0	1.0
...	...	...	...	...	...
540	1.0	0.0	1.0	0.0	0.0
541	0.0	0.0	0.0	0.0	0.0
542	1.0	0.0	0.0	0.0	0.0
543	0.0	0.0	0.0	0.0	0.0
544	1.0	0.0	0.0	0.0	0.0

545 rows × 8 columns

In [18]:

```
encoded_data.to_csv(r'Data/encoded_data_file.csv', index=False)
```

Loading the Encoded Data for Applying Regularization Techniques

In [19]:

```
dataset = pd.read_csv('Data/housing_2.csv')
dataset
```

Out[19]:

	area	bedrooms	bathrooms	stories	parking	mainroad_Yes	guestroom_Yes	baseme
0	7420	4	2	3	2	1	0	
1	8960	4	4	4	3	1	0	
2	9960	3	2	2	2	1	0	
3	7500	4	2	2	3	1	0	
4	7420	4	1	2	2	1	1	
...	...	...	...	...	...	...	...	
540	3000	2	1	1	2	1	0	
541	2400	3	1	1	0	0	0	
542	3620	2	1	1	0	1	0	
543	2910	3	1	1	0	0	0	
544	3850	3	1	2	0	1	0	

545 rows × 14 columns

In [20]:

```
dataset = pd.read_csv(r'Data/housing_2.csv')
dataset.head(3)
```

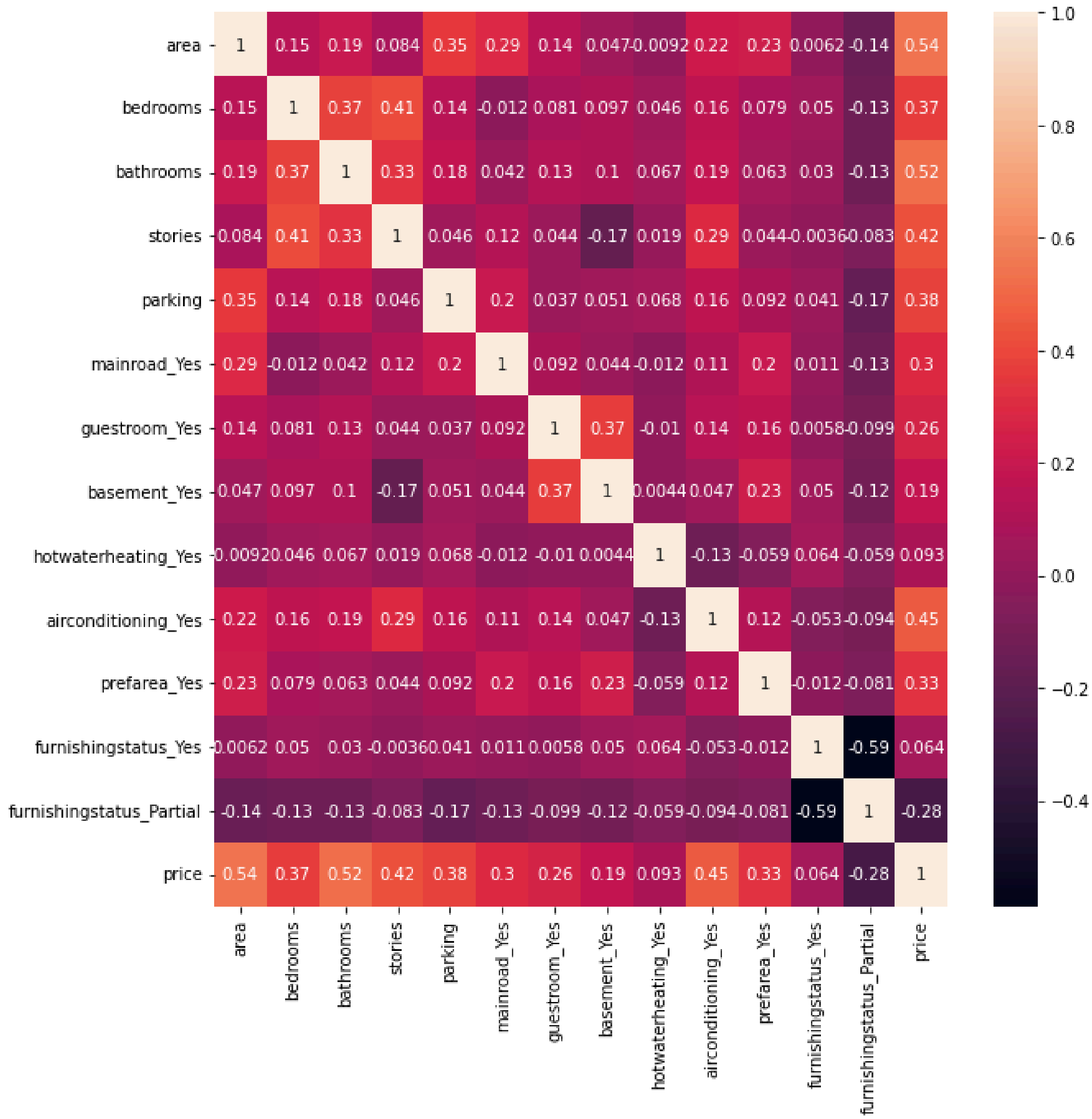
Out[20]:

	area	bedrooms	bathrooms	stories	parking	mainroad_Yes	guestroom_Yes	basement_
0	7420	4	2	3	2	1	0	
1	8960	4	4	4	3	1	0	
2	9960	3	2	2	2	1	0	

Check Correlation in Data

In [21]:

```
plt.figure(figsize=(10,10))
sns.heatmap(data=dataset.corr(), annot=True)
plt.show()
```



In [22]:
```python
x = dataset.iloc[:,:-1]
x
```

Out[22]:

	area	bedrooms	bathrooms	stories	parking	mainroad_Yes	guestroom_Yes	baseme
0	7420	4	2	3	2	1	0	
1	8960	4	4	4	3	1	0	
2	9960	3	2	2	2	1	0	
3	7500	4	2	2	3	1	0	
4	7420	4	1	2	2	1	1	
...	...	...	...	...	...	...	...	
540	3000	2	1	1	2	1	0	
541	2400	3	1	1	0	0	0	
542	3620	2	1	1	0	1	0	
543	2910	3	1	1	0	0	0	
544	3850	3	1	2	0	1	0	

545 rows × 13 columns

In [23]:

```
y=dataset['price']
y
```

Out[23]:

```
0      13300000
1      12250000
2      12250000
3      12215000
4      11410000
         ...
540     1820000
541     1767150
542     1750000
543     1750000
544     1750000
Name: price, Length: 545, dtype: int64
```

Perform Scaling on Data

In [28]:

```
sc = StandardScaler()
sc.fit(x)
sc.transform(x)
```

Out[28]:
```
array([[ 1.04672629,  1.40341936,  1.42181174, ...,  1.80494113,
        -0.84488844, -0.6964292 ],
       [ 1.75700953,  1.40341936,  5.40580863, ..., -0.55403469,
        -0.84488844, -0.6964292 ],
       [ 2.21823241,  0.04727831,  1.42181174, ...,  1.80494113,
         1.18358821, -0.6964292 ],
       ...,
       [-0.70592066, -1.30886273, -0.57018671, ..., -0.55403469,
        -0.84488844,  1.43589615],
       [-1.03338891,  0.04727831, -0.57018671, ..., -0.55403469,
        -0.84488844, -0.6964292 ],
       [-0.5998394 ,  0.04727831, -0.57018671, ..., -0.55403469,
        -0.84488844,  1.43589615]])
```

In [29]:
```
# transform data to csv sheet
x = pd.DataFrame(sc.transform(x), columns=x.columns)
x
```

Out[29]:

	area	bedrooms	bathrooms	stories	parking	mainroad_Yes	guestroom_Yes
0	1.046726	1.403419	1.421812	1.378217	1.517692	0.405623	-0.465315
1	1.757010	1.403419	5.405809	2.532024	2.679409	0.405623	-0.465315
2	2.218232	0.047278	1.421812	0.224410	1.517692	0.405623	-0.465315
3	1.083624	1.403419	1.421812	0.224410	2.679409	0.405623	-0.465315
4	1.046726	1.403419	-0.570187	0.224410	1.517692	0.405623	2.149083
...	...	...	...	...	...	...	...
540	-0.991879	-1.308863	-0.570187	-0.929397	1.517692	0.405623	-0.465315
541	-1.268613	0.047278	-0.570187	-0.929397	-0.805741	-2.465344	-0.465315
542	-0.705921	-1.308863	-0.570187	-0.929397	-0.805741	0.405623	-0.465315
543	-1.033389	0.047278	-0.570187	-0.929397	-0.805741	-2.465344	-0.465315
544	-0.599839	0.047278	-0.570187	0.224410	-0.805741	0.405623	-0.465315

545 rows × 13 columns

Split data into train and test

In [30]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state
```

26.1 Model by Linear Regression

In [31]:
```
from sklearn.linear_model import LinearRegression, Lasso, Ridge
```

In [32]:
```
lr = LinearRegression()
lr.fit(x_train, y_train)
```

Out[32]:
```
▾ LinearRegression
LinearRegression()
```

Test Model

In [35]:
```
lr.score(x_test, y_test)*100
```

Out[35]:
```
65.29242642153177
```

Graphical representation of constant and coefficient

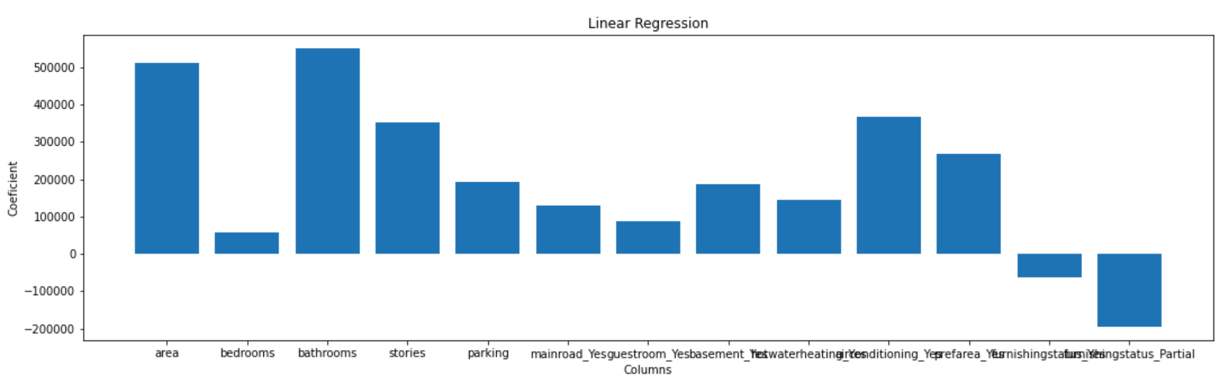
In [36]:
```
lr.coef_
```

Out[36]:
```
array([ 511615.56377666,   56615.57245779,  549420.50124098,
        353158.42985604,  193542.78167455,  128151.92129533,
         88590.21346152,  186194.15050566,  143233.20624958,
        367817.89491558,  267018.66081239,  -62550.29721128,
       -193987.7810882 ])
```

In [37]:
```
x.columns
```

Out[37]:
```
Index(['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'mainroad_Yes',
       'guestroom_Yes', 'basement_Yes', 'hotwaterheating_Yes',
       'airconditioning_Yes', 'prefarea_Yes', 'furnishingstatus_Yes',
       'furnishingstatus_Partial'],
      dtype='object')
```

In [43]:
```
#plt.bar(x_data, y_data)
plt.figure(figsize=(18,5))
plt.title("Linear Regression")
plt.bar(x.columns, lr.coef_)
plt.xlabel("Columns")
plt.ylabel("Coeficient")
plt.show()
```



26.2 Model by Lasso (L1)

This technique is used for feature selection

In [45]:
```
# alpha: penalty corner, default 1.0
la = Lasso (alpha=0.5)
la.fit(x_train, y_train)
```

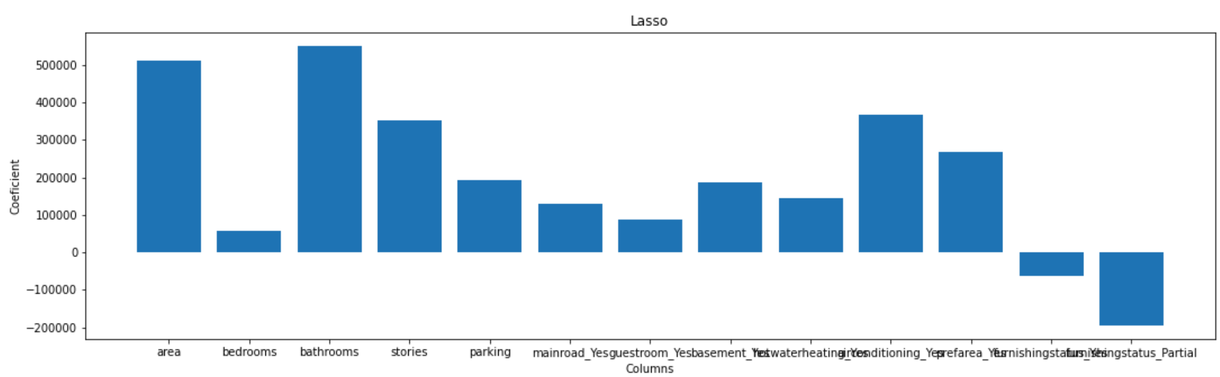
Out[45]:
```
▾ Lasso
Lasso(alpha=0.5)
```

Test the Model

In [47]:
```
la.score(x_test, y_test)*100
```

Out[47]:
```
65.29241383553659
```

In [50]:
```
#plt.bar(x_data, y_data)
plt.figure(figsize=(18,5))
plt.title("Lasso")
plt.bar(x.columns, la.coef_)
plt.xlabel("Columns")
plt.ylabel("Coeficient")
plt.show()
```



26.3 Model by Ridge (L2)

- It reduces coefficient values and save model from over-fitting

In [51]:
```
ri = Ridge(alpha=10)
ri.fit(x_train, y_train)
```

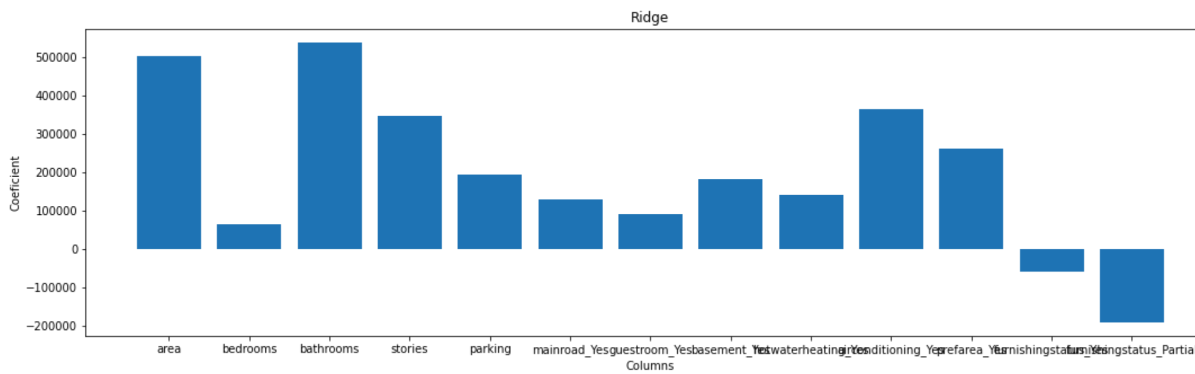
Out[51]:
```
▾ Ridge
Ridge(alpha=10)
```

Test the Model

In [53]:
```
ri.score(x_test, y_test)*100
```

Out[53]: 65.19079253215374

In [54]:
```python
#plt.bar(x_data, y_data)
plt.figure(figsize=(18,5))
plt.title("Ridge")
plt.bar(x.columns, ri.coef_)
plt.xlabel("Columns")
plt.ylabel("Coeficient")
plt.show()
```



26.4 To check which model is best

26.4.1 Regression Model

In [56]:
```python
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np
```

In [61]:
```python
#mean_squared_error(y_true, y_pred)
print(mean_squared_error(y_test, lr.predict(x_test)))
#mean_absolute_error(y_true, y_pred)
print(mean_absolute_error(y_test, lr.predict(x_test)))
# Root mean sqaure error
print(np.sqrt(mean_squared_error(y_test, lr.predict(x_test))))
```

```
1754318687330.6672
970043.4039201641
1324506.96009144
```

26.4.2 Lasso (L1) Model

In [62]:
```python
#mean_squared_error(y_true, y_pred)
print(mean_squared_error(y_test, la.predict(x_test)))
#mean_absolute_error(y_true, y_pred)
print(mean_absolute_error(y_test, la.predict(x_test)))
# Root mean sqaure error
print(np.sqrt(mean_squared_error(y_test, la.predict(x_test))))
```

```
1754319323498.6353
970043.3950649527
1324507.2002441646
```

26.4.3 Ridge (L2) Model

In [63]:
```
#mean_squared_error(y_true, y_pred)
print(mean_squared_error(y_test, ri.predict(x_test)))
#mean_absolute_error(y_true, y_pred)
print(mean_absolute_error(y_test, ri.predict(x_test)))
# Root mean sqaure error
print(np.sqrt(mean_squared_error(y_test, ri.predict(x_test))))
```

```
1759455843663.3877
967942.6216085082
1326444.8136516602
```

We will use Ridge model as it is showing comparatively less error as compared to Lasso and Linear regression model

26.4.3 To compare coefficient of all models

In [64]:
```
df = pd.DataFrame({"col_name":x.columns, "LinearRegression":lr.coef_, "Lasso":la.co
df
```

Out[64]:

	col_name	LinearRegression	Lasso	Ridge
0	area	511615.563777	511615.467912	502252.286215
1	bedrooms	56615.572458	56615.441731	65132.373585
2	bathrooms	549420.501241	549420.321462	537574.041615
3	stories	353158.429856	353158.186082	346006.857732
4	parking	193542.781675	193542.619408	194954.682792
5	mainroad_Yes	128151.921295	128151.745183	130790.775299
6	guestroom_Yes	88590.213462	88590.029990	91998.609421
7	basement_Yes	186194.150506	186193.873949	181385.995261
8	hotwaterheating_Yes	143233.206250	143232.743062	140133.580908
9	airconditioning_Yes	367817.894916	367817.774947	364207.282689
10	prefarea_Yes	267018.660812	267018.388019	262517.337220
11	furnishingstatus_Yes	-62550.297211	-62549.219050	-58988.254578
12	furnishingstatus_Partial	-193987.781088	-193986.867394	-190415.566289

In [ ]: