Linear Regression

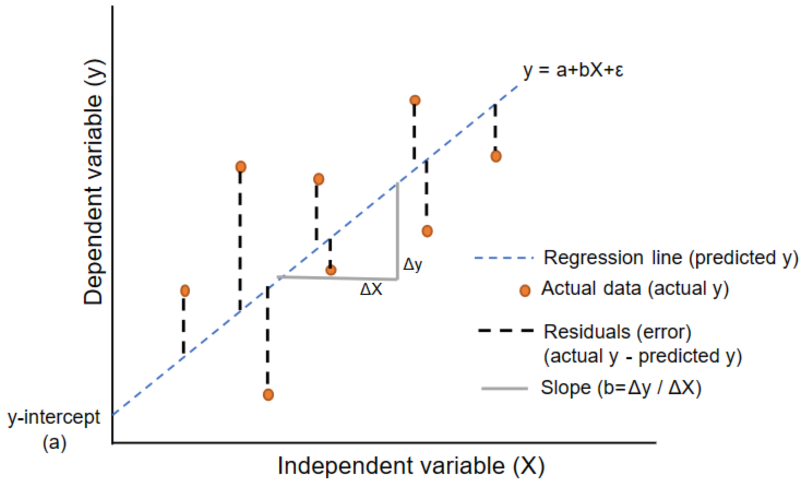
- It is used to estimate real values (cost of house, no. of calls, total sales etc.) based on continuous variable.
- Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation.

Simple Linear Regression

One independent feature and one dependent feature.

Eg: Aim: to create a model, which takes input as height and predict weight.
Dataset : Height, Weight.

Eg: Model: year of experience and salary
Predict: salary based on input salary



$\hat{Y} = m\,\bar{x}+c$

Where:- y = Dependent Variable
m = b = Slope
x = Independent Variable
c = a = Intercept

- Difference b/w real points and predicted points is called residuals or errors.
- Based on the training dataset, it finds the best fit line in such a way that the sum of difference between real points and predicted showed be minimum.

First we need to understand why are creating a straight line?
Best fit line is nothing but a equation of straight line.

$\hat{Y} = m\,\bar{x}+c$

c = intercept: when x = 0, the line meeting the y-axis that particular point is known as intercept.
m = slope: with the unit movement in the x-axis what is the moment in the y-axis by changing c and m, best fit line will be change.

Mathematical Solution:

Experience	1	2	3	4	5
Salary (LPA)	7	14	15	18	19

$\hat{Y}$ = m $\bar{x}$+c (If slope is + ive)
$\hat{Y}$ = - m $\bar{x}$+c (If slope is - ive)

$$b_1 = \frac{\sum_{i=1}^{n}\left((x_i - \bar{x})(y_i - \bar{y})\right)}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

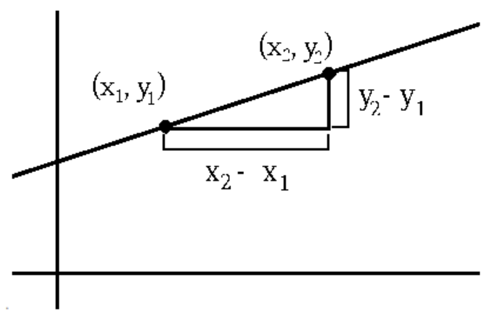
Experience	Salary (LPA)	$(x-\bar{x})$	$(y-\bar{y})$	$(x-\bar{x})2$	$(x-\bar{x})(y-\bar{y})$
1	7	-2	-7.6	4	+15.2
2	14	-1	-0.6	1	0.6
3	15	0	0	0	0
4	18	1	3.4	1	3.4
5	19	2	4.4	4	8.8
$\Sigma x = 15$	Σy			$\Sigma(x-\bar{x})2 = 10$	$\Sigma(x-\bar{x})(y-\bar{y}) = 28$
$\bar{x} = 3$	$\bar{y} = 14.6$				

m = 28/10
   = 2.8 (slope)

$\hat{Y}$ = m $\bar{x}$+c

14.6 = 2.8*3+c
c = 14.6 – 8.4
c = 6.2 (Intercept)

<u>Formula of Slope</u>



$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

m = Δy/Δx

Using Coding:-

In [1]:
```
# Simple Linear Regression
import pandas as pd
```

In [2]:
```
df = pd.DataFrame({'Place':[1,2,3,4,5,],"Profit":[7,14,15,18,19]})
df
```

Out[2]:

	Place	Profit
0	1	7
1	2	14
2	3	15
3	4	18
4	5	19

In [3]:
```
x = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [4]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
ypre = model.fit(x,y)
```

```
C:\Users\hp\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for thi
s version of SciPy (detected version 1.23.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [5]:
```
ypre
```

Out[5]:
```
▾ LinearRegression
LinearRegression()
```

In [6]:
```
model.coef_
```

Out[6]:
```
array([2.8])
```

In [7]:
```
model.intercept_
```

Out[7]:
```
6.1999999999999975
```

In [8]:
```
ypre=model.predict(x)
```

In [9]:
```
ypre
```

Out[9]:
```
array([ 9. , 11.8, 14.6, 17.4, 20.2])
```

In [10]:
```
check = pd.DataFrame({'Place':df['Place'],'Actual Profit':df['Profit'],"Predicted Profit":ypre})
check
```

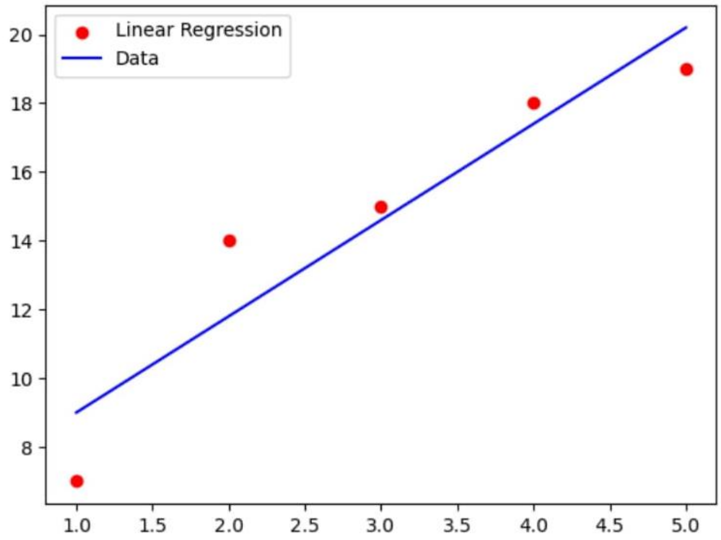
Out[10]:

	Place	Actual Profit	Predicted Profit
0	1	7	9.0
1	2	14	11.8
2	3	15	14.6
3	4	18	17.4
4	5	19	20.2

In [11]:
```
from matplotlib import pyplot as plt
plt.scatter(x,y,c='r')
plt.plot(x,ypre,c='b')

# Function add a legend
plt.legend(["Linear Regression", "Data"], loc ="upper left")
```

Out[11]: <matplotlib.legend.Legend at 0x205ea123ca0>



In [12]:
```
# Analyze the performance of the model by calculating mean squared error and R2
import numpy as np
error = y - ypre
se = np.sum(error**2)
print('Squared Error: ', se)
n = np.size(x)
mse = se/n
print('Mean Squared Error: ', mse)

rmse = np.sqrt(mse)
print('Root Mean Square Error: ', rmse)
ymean = np.mean(y)
SSt = np.sum((y - ymean)**2)
R2 = 1- (se/SSt)
print('R2 Score: ', R2)
```

```
Squared Error:  10.800000000000004
Mean Squared Error:  2.160000000000001
Root Mean Square Error:  1.4696938456699071
R2 Score:  0.8789237668161435
```

Multiple Linear Regression

Multiple Linear Regression is one of the important regression algorithms which model the linear relationship between a single dependent continuous variable and (more than 1) independent variable.

- It can be applied to many pactical fields like politics, economics, medical, research works and many different kinds of businesses.

$$\boxed{Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ \dots + \beta_i X_i}$$

Y : Dependent variable
β_0 : Intercept
β_i : Slope for X_i
X = Independent variable

$$\hat{b}_1 = \frac{(\sum x_2^2)(\sum x_1 y) - (\sum x_1 x_2)(\sum x_2 y)}{(\sum x_1^2)(\sum x_2^2) - (\sum x_1 x_2)^2}$$

$$\hat{b}_2 = \frac{(\sum x_1^2)(\sum x_2 y) - (\sum x_1 x_2)(\sum x_1 y)}{(\sum x_1^2)(\sum x_2^2) - (\sum x_1 x_2)^2}$$

Solving by Hand

Suppose we have the following dataset with one response variable y and two predictor variables

X_1 and X.

y	X_1	X_2
140	60	22
155	62	25
159	67	24
179	70	20
192	71	15
200	72	14
212	75	14
215	78	11

Use the following steps to fit a multiple linear regression model to this dataset.

Step 1: Calculate X_1^2, X_2^2, X_1y, X_2y and X_1X_2

	y	X_1	X_2
	140	60	22
	155	62	25
	159	67	24
	179	70	20
	192	71	15
	200	72	14
	212	75	14
	215	78	11
Mean	181.5	69.375	18.125
Sum	1452	555	145

	X_1^2	X_2^2	X_1y	X_2y	X_1X_2
	3600	484	8400	3080	1320
	3844	625	9610	3875	1550
	4489	576	10653	3816	1608
	4900	400	12530	3580	1400
	5041	225	13632	2880	1065
	5184	196	14400	2800	1008
	5625	196	15900	2968	1050
	6084	121	16770	2365	858
Sum	38767	2823	101895	25364	9859

Step 2: Calculate Regression Sums.

Next, make the following regression sum calculations:

- $\Sigma x_1^2 = \Sigma X_1^2 - (\Sigma X_1)^2 / n = 38{,}767 - (555)^2 / 8 = \mathbf{263.875}$
- $\Sigma x_2^2 = \Sigma X_2^2 - (\Sigma X_2)^2 / n = 2{,}823 - (145)^2 / 8 = \mathbf{194.875}$
- $\Sigma x_1 y = \Sigma X_1 y - (\Sigma X_1 \Sigma y) / n = 101{,}895 - (555*1{,}452) / 8 = \mathbf{1{,}162.5}$
- $\Sigma x_2 y = \Sigma X_2 y - (\Sigma X_2 \Sigma y) / n = 25{,}364 - (145*1{,}452) / 8 = \mathbf{-953.5}$
- $\Sigma x_1 x_2 = \Sigma X_1 X_2 - (\Sigma X_1 \Sigma X_2) / n = 9{,}859 - (555*145) / 8 = \mathbf{-200.375}$

	y	X_1	X_2
	140	60	22
	155	62	25
	159	67	24
	179	70	20
	192	71	15
	200	72	14
	212	75	14
	215	78	11
Mean	181.5	69.375	18.125
Sum	1452	555	145

	X_1^2	X_2^2	X_1y	X_2y	X_1X_2
	3600	484	8400	3080	1320
	3844	625	9610	3875	1550
	4489	576	10653	3816	1608
	4900	400	12530	3580	1400
	5041	225	13632	2880	1065
	5184	196	14400	2800	1008
	5625	196	15900	2968	1050
	6084	121	16770	2365	858
Sum	38767	2823	101895	25364	9859
Reg Sums	263.875	194.875	1162.5	-953.5	-200.375

Step 3: Calculate b_0, b_1, and b_2.

The formula to calculate b_1 is: $[(\Sigma x_2^2)(\Sigma x_1 y) - (\Sigma x_1 x_2)(\Sigma x_2 y)] / [(\Sigma x_1^2)(\Sigma x_2^2) - (\Sigma x_1 x_2)^2]$

Thus, $\mathbf{b_1} = [(194.875)(1162.5) - (-200.375)(-953.5)] / [(263.875)(194.875) - (-200.375)^2] = \mathbf{3.148}$

The formula to calculate b_2 is: $[(\Sigma x_1^2)(\Sigma x_2 y) - (\Sigma x_1 x_2)(\Sigma x_1 y)] / [(\Sigma x_1^2)(\Sigma x_2^2) - (\Sigma x_1 x_2)^2]$

Thus, $\mathbf{b_2} = [(263.875)(-953.5) - (-200.375)(1152.5)] / [(263.875)(194.875) - (-200.375)^2] = \mathbf{-1.656}$

The formula to calculate b_0 is: $y - b_1 X_1 - b_2 X_2$

Thus, $\mathbf{b_0} = 181.5 - 3.148(69.375) - (-1.656)(18.125) = \mathbf{-6.867}$

Step 5: Place b_0, b_1, and b_2 in the estimated linear regression equation.

The estimated linear regression equation is: $\hat{y} = b_0 + b_1 * x_1 + b_2 * x_2$

In our example, it is $\mathbf{\hat{y} = -6.867 + 3.148x_1 - 1.656x_2}$

How to Interpret a Multiple Linear Regression Equation

Here is how to interpret this estimated linear regression equation: $\hat{y} = -6.867 + 3.148x_1 - 1.656x_2$

b_0 = -6.867. When both predictor variables are equal to zero, the mean value for y is -6.867.

b_1 = 3.148. A one unit increase in x_1 is associated with a 3.148 unit increase in y, on average, assuming x_2 is held constant.

b_2 = -1.656. A one unit increase in x_2 is associated with a 1.656 unit decrease in y, on average, assuming x_1 is held constant.

Steps to implement Multivariate regression:-

1. Feature selection

The selection of features plays the most important role in multivariate regression.

Finding the feature that is needed for finding which variable is dependent on this feature.

2. Normalizing Features

For better analysis features are need to be scaled to get them into a specific range. We can also change the value of each feature.

3. Select Loss function and Hypothesis

The loss function calculates the loss when the hypothesis predicts the wrong value.

And hypothesis means predicted value from the feature variable.

4. Set Hypothesis Parameters

Set the hypothesis parameter that can reduce the loss function and can predict.

5. Minimize the Loss Function

Minimizing the loss by using some lose minimization algorithm and use it over the dataset which can help to adjust the hypothesis parameters. Once the loss is minimized then it can be used for prediction.

There are many algorithms that can be used for reducing the loss such as gradient descent.

6. Test the hypothesis function

Check the hypothesis function how correct it predicting values, test it on test data.

In [13]:
```
import pandas as pd
```

In [14]:
```
data = pd.read_csv('50_Startups.csv')
data.head()
```

Out[14]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

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

Out[15]: (50, 5)

In [16]:
```python
# All the variables in the numeric form so it will change the catoragical data in numeric form.
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [17]:
```python
data['State']=le.fit_transform(data['State'])
```

In [18]:
```python
data.tail()
```

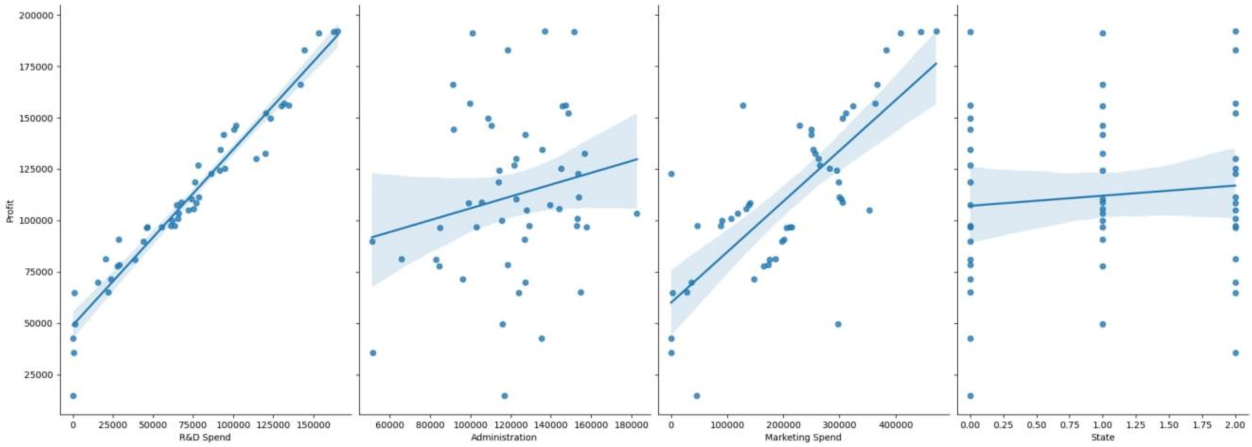
Out[18]:

	R&D Spend	Administration	Marketing Spend	State	Profit
45	1000.23	124153.04	1903.93	2	64926.08
46	1315.46	115816.21	297114.46	1	49490.75
47	0.00	135426.92	0.00	0	42559.73
48	542.05	51743.15	0.00	2	35673.41
49	0.00	116983.80	45173.06	0	14681.40

In [19]:
```python
import seaborn as sns
%matplotlib inline
sns.pairplot(data,x_vars=['R&D Spend', 'Administration', 'Marketing Spend', 'State'], y_vars="Profit",
             size=7,aspect=0.7,kind='reg')
```

```
C:\Users\hp\anaconda3\lib\site-packages\seaborn\axisgrid.py:2095: UserWarning: The `size` parameter has been renamed to `height`; pl
ease update your code.
  warnings.warn(msg, UserWarning)
```

Out[19]:
```
<seaborn.axisgrid.PairGrid at 0x205ec184820>
```



In [20]:
```python
# divided data into independent and dependent variable.
x = data.iloc[:,:-1]
y = data.iloc[:,-1]
```

In [21]:
```python
# spliting the dataset into training set and test set.
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)
```

In [22]:
```python
# import the regression model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

In [23]:
```python
# fitting multi-regression in train test.
model.fit(x_train,y_train)
```

Out[23]:
```
▾ LinearRegression
LinearRegression()
```

In [24]:
```python
# preadcting the test set result
ypre = model.predict(x_test)
```

In [25]:
```python
ypre
```

Out[25]:
```
array([150032.53333639, 112810.92048286, 119789.51493002, 134519.22150981,
       129389.03112571, 112204.72469995, 116057.74096167,  51652.30687496,
       117577.907593  , 178811.87143387])
```

In [26]:
```python
# comparing predicted profit with actual profit.
check = pd.DataFrame({'actual':y_test,'predicted':ypre})
check
```

Out[26]:

	actual	predicted
14	132602.65	150032.533336
24	108552.04	112810.920483
21	111313.02	119789.514930
10	146121.95	134519.221510
12	141585.52	129389.031126
26	105733.54	112204.724700
27	105008.31	116057.740962
49	14681.40	51652.306875
16	126992.93	117577.907593
2	191050.39	178811.871434

In [27]:

```
# find the MAE,MSE,RMSE,R2 Score,Adjusted R2 Score
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score

print('MAE: ',mean_absolute_error(y_test,ypre))

print('MSE:',mean_squared_error(y_test,ypre))

print('RMSE:',np.sqrt(mean_squared_error(y_test,ypre)))

print('R2 Score:',r2_score(y_test,ypre))
```

```
MAE:  13010.95396234671
MSE: 244640560.64363137
RMSE: 15640.98975907955
R2 Score: 0.8648710271208794
```

Polynomial Regression

A regression equation is a polynomial regression equation if the power of the independent variable is more than 1. The equation below represents a polynomial equation.

$$y= b_0+b_1x_1+ b_2x_1^2+ b_2x_1^3+...... b_nx_1^n$$

In this regression technique, the best fit line is not a straight line. It is rather a curve that fits into the data points.

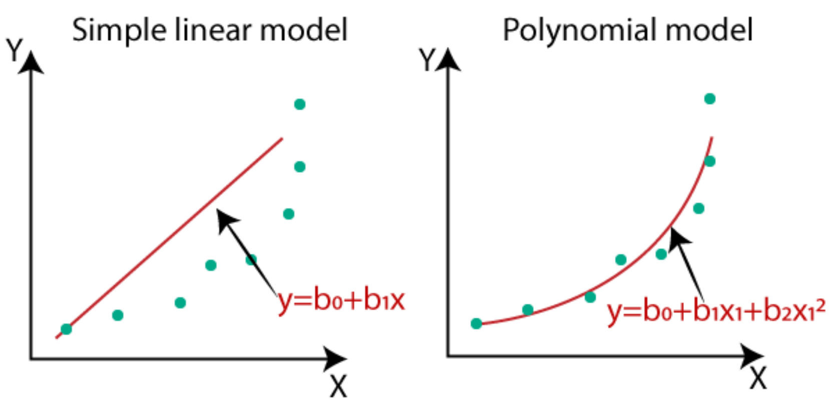
- It is also called the special case of Multiple Linear Regression in ML. Because we add some polynomial terms to the Multiple Linear regression equation to convert it into Polynomial Regression.
- It is a linear model with some modification in order to increase the accuracy.
- The dataset used in Polynomial regression for training is of non-linear nature.
- It makes use of a linear regression model to fit the complicated and non-linear functions and datasets.

Hence, *"In Polynomial regression, the original features are converted into Polynomial features of required degree (2,3,..,n) and then modeled using a linear model."*

Need for Polynomial Regression:

The need of Polynomial Regression in ML can be understood in the below points:

- If we apply a linear model on a **linear dataset**, then it provides us a good result as we have seen in Simple Linear Regression, but if we apply the same model without any modification on a **non-linear dataset**, then it will produce a drastic output. Due to which loss function will increase, the error rate will be high, and accuracy will be decreased.
- So for such cases, **where data points are arranged in a non-linear fashion, we need the Polynomial Regression model**. We can understand it in a better way using the below comparison diagram of the linear dataset and non-linear dataset.



- In the above image, we have taken a dataset which is arranged non-linearly. So if we try to cover it with a linear model, then we can clearly see that it hardly covers any data point. On the other hand, a curve is suitable to cover most of the data points, which is of the Polynomial model.
- Hence, *if the datasets are arranged in a non-linear fashion, then we should use the Polynomial Regression model instead of Simple Linear Regression.*

Note: A Polynomial Regression algorithm is also called Polynomial Linear Regression because it does not depend on the variables, instead, it depends on the coefficients, which are arranged in a linear fashion.

Equation of the Polynomial Regression Model:

Simple Linear Regression equation: $y = b_0+b_1x$ ....(a)

Multiple Linear Regression equation: $y= b_0+b_1x+ b_2x_2+ b_3x_3+....+ b_nx_n$ ....(b)

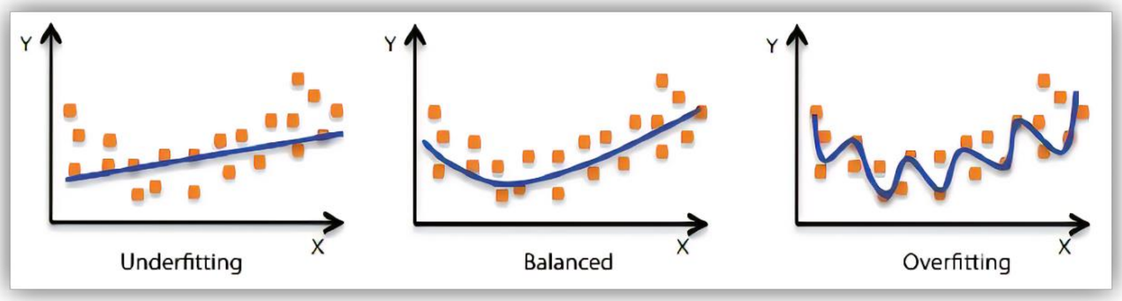
Polynomial Regression equation: $y= b_0+b_1x + b_2x^2+ b_3x^3+....+ b_nx^n$ ....(c)

When we compare the above three equations, we can clearly see that all three equations are Polynomial equations but differ by the degree of variables. The Simple and Multiple Linear equations are also Polynomial equations with a single degree, and the Polynomial regression equation is Linear equation with the nth degree. So if we add a degree to our linear equations, then it will be converted into Polynomial Linear equations.

***Note:** To better understand Polynomial Regression, you must have knowledge of Simple Linear Regression.*

Important Points:-

- While there might be a temptation to fit a higher degree polynomial to get lower error, this can result in over-fitting. Always plot the relationships to see the fit and focus on making sure that the curve fits the nature of the nature problem.
- Here is an example of how ploting can help.



- Espically look out for curve towards the neds and see whether those shapes and trends make sense. Higher polynomials can end up producing wired results on extrapolation.

Implementation of Polynomial Regression using Python:

Here we will implement the Polynomial Regression using Python. We will understand it by comparing Polynomial Regression model with the Simple Linear Regression model. So first, let's understand the problem for which we are going to build the model.

Steps for Polynomial Regression:

The main steps involved in Polynomial Regression are given below:

- Data Pre-processing
- Build a Linear Regression model and fit it to the dataset
- Build a Polynomial Regression model and fit it to the dataset
- Visualize the result for Linear Regression and Polynomial Regression model.
- Predicting the output.

Note: *Here, we will build the Linear regression model as well as Polynomial Regression to see the results between the predictions. And Linear regression model is for reference.*

Data Pre-processing Step:

The data pre-processing step will remain the same as in previous regression models, except for some changes. In the Polynomial Regression model, we will not use feature scaling, and also we will not split our dataset into training and test set. It has two reasons:

- The dataset contains very less information which is not suitable to divide it into a test and training set, else our model will not be able to find the correlations between the salaries and levels.
- In this model, we want very accurate predictions for salary, so the model should have enough information.

In [23]:
```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

In [24]:
```
data_set = pd.DataFrame({'Position':["Business Analyst","Junior Consultant","Senior Consultant","Manager","Country Manager",
                          "Region Manager","Partner","Senior Partner","C-level","CEO"],
                          "Level":[1,2,3,4,5,6,7,8,9,10],
                          "Salary":[45000,50000,60000,80000,110000,150000,200000,300000,500000,1000000]})
data_set
```

Out[24]:

	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

In [25]:
```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, 1:2].values
y= data_set.iloc[:, 2].values
```

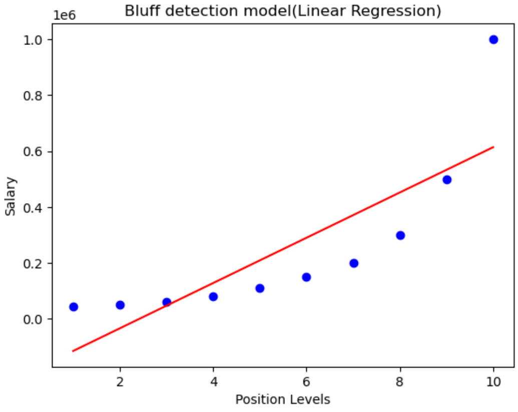
In [26]:
```
#Fitting the Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin_regs= LinearRegression()
lin_regs.fit(x,y)
```

Out[26]:
```
▾ LinearRegression
LinearRegression()
```

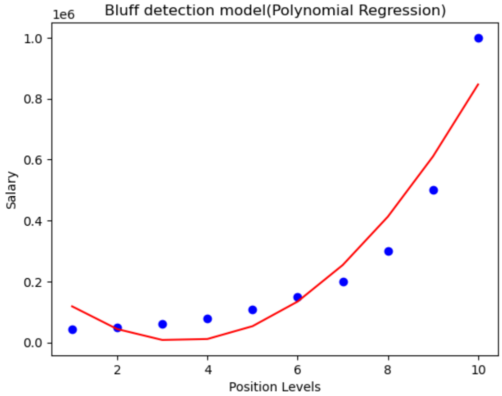
In [27]:
```python
#Fitting the Polynomial regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_regs= PolynomialFeatures(degree= 2)
x_poly= poly_regs.fit_transform(x)
lin_reg_2 =LinearRegression()
lin_reg_2.fit(x_poly, y)
```

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

In [28]:
```python
#Visulaizing the result for Linear Regression model
mtp.scatter(x,y,color="blue")
mtp.plot(x,lin_regs.predict(x), color="red")
mtp.title("Bluff detection model(Linear Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```

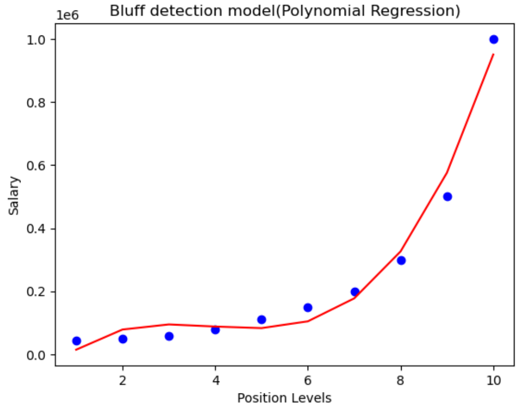


In [29]:
```python
#Visulaizing the result for Polynomial Regression
mtp.scatter(x,y,color="blue")
mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
mtp.title("Bluff detection model(Polynomial Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```



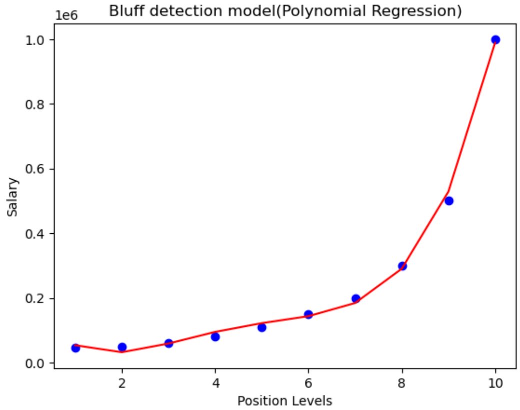
In [31]:
```python
#Fitting the Polynomial regression to the dataset for degree 3
poly_regs= PolynomialFeatures(degree= 3)
x_poly= poly_regs.fit_transform(x)
lin_reg_2 =LinearRegression()
lin_reg_2.fit(x_poly, y)

mtp.scatter(x,y,color="blue")
mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
mtp.title("Bluff detection model(Polynomial Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```



In [32]:
```
#Fitting the Polynomial regression to the dataset for degree 4
poly_regs= PolynomialFeatures(degree= 4)
x_poly= poly_regs.fit_transform(x)
lin_reg_2 =LinearRegression()
lin_reg_2.fit(x_poly, y)

mtp.scatter(x,y,color="blue")
mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
mtp.title("Bluff detection model(Polynomial Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```



In [33]:
```
#Predicting the final result with the Linear Regression model:
lin_pred = lin_regs.predict([[6.5]])
print(lin_pred)
```

```
[330378.78787879]
```

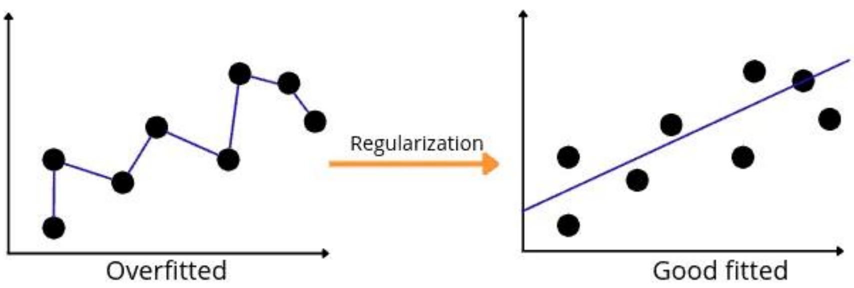
In [34]:
```
#Predicting the final result with the Polynomial Regression model:
poly_pred = lin_reg_2.predict(poly_regs.fit_transform([[6.5]]))
print(poly_pred)
```

```
[158862.45265153]
```

Regularization

Regularization is one of the most important concepts of machine learning. It is a technique to prevent the model from overfitting by adding extra information to it.

Regularization refers to techniques used to calibrate machine learning models to minimize the adjusted loss function and prevent overfitting or underfitting. Using regularization, we can fit our machine learning model appropriately on a given test set and reduce its errors.



Let us now learn how the L1 (the Lasso regression) and L2 (the Ridge regression) help to regularize the model.

Sometimes the machine learning model performs well with the training data but does not perform well with the test data. It means the model is not able to predict the output when deals with unseen data by introducing noise in the output, and hence the model is called overfitted. This problem can be deal with the help of a regularization technique.

This technique can be used in such a way that it will allow to maintain all variables or features in the model by reducing the magnitude of the variables. Hence, it maintains accuracy as well as a generalization of the model.

It mainly regularizes or reduces the coefficient of features toward zero. In simple words, "*In regularization technique, we reduce the magnitude of the features by keeping the same number of features.*"

Regularization works by adding a penalty or complexity term to the complex model. Let's consider the simple linear regression equation:

y= β0+β1x1+β2x2+β3x3+⋯+βnxn +b

In the above equation, Y represents the value to be predicted

X1, X2, …Xn are the features for Y.

β0,β1,…..βn are the weights or magnitude attached to the features, respectively. Here represents the bias of the model, and b represents the intercept.

Linear regression models try to optimize the β0 and b to minimize the cost function. The equation for the cost function for the linear model is given below:

$$\sum_{i=1}^{M}\left(y_i - y'_i\right)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{n} \beta_j * Xij\right)^2$$

Now, we will add a loss function and optimize parameter to make the model that can predict the accurate value of Y. The loss function for the linear regression is called as **RSS or Residual sum of squares.**

Techniques of Regularization

There are mainly two types of regularization techniques, which are given below:

Ridge Regression

Lasso Regression

Ridge Regression

- Ridge regression is one of the types of linear regression in which a small amount of bias is introduced so that we can get better long-term predictions.
- Ridge regression is a regularization technique, which is used to reduce the complexity of the model. It is also called as **L2 regularization**.
- In this technique, the cost function is altered by adding the penalty term to it. The amount of bias added to the model is called **Ridge Regression penalty**. We can calculate it by multiplying with the lambda to the squared weight of each individual feature.
- As we know, the simple linear equation uses the following mathematical equation to find the best-fitted line:

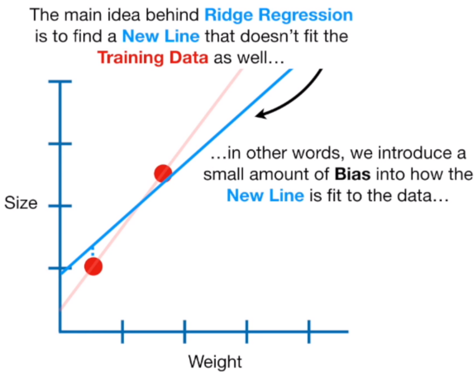
$$y = y\text{-intercept} + slope*x$$

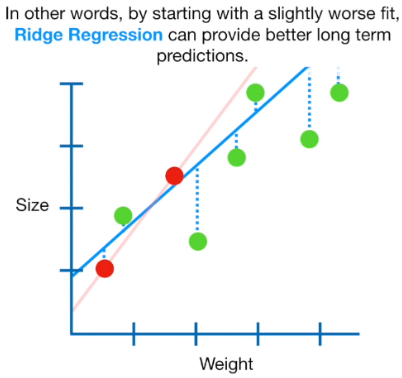
- While the Ridge regression uses a slightly modified equation with the penalty term to find the new best-fitted line to introduce a small bias and reduce variance. The penalty term, also known as the penalty function or cost function, contains lambda and slope square, as shown below.

$$y = (y\text{-intercept} + slope*x) + (lamda * slope^2)$$

$$\sum_{i=1}^{M}(y_i - y'_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{n}\beta_j * x_{ij}\right)^2 + \lambda\sum_{j=0}^{n}\beta_j^{\,2}$$

- In the above equation, the penalty term regularizes the coefficients of the model, and hence ridge regression reduces the amplitudes of the coefficients that decreases the complexity of the model.
- As we can see from the above equation, if the values of **λ tend to zero, the equation becomes the cost function of the linear regression model.** Hence, for the minimum value of λ, the model will resemble the linear regression model.
- A general linear or polynomial regression will fail if there is high collinearity between the independent variables, so to solve such problems, Ridge regression can be used.
- It helps to solve the problems if we have more parameters than samples.





Lasso Regression:

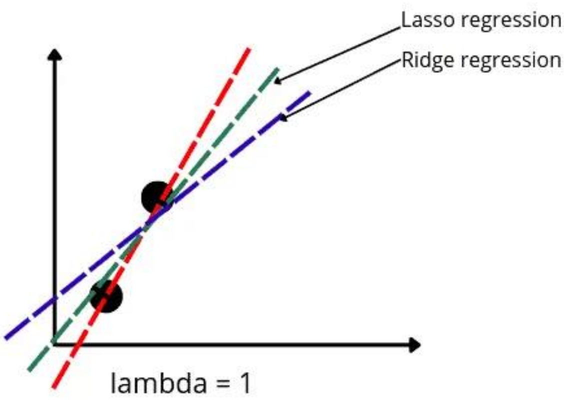
- Lasso regression is another regularization technique to reduce the complexity of the model. It stands for **Least Absolute and Selection Operator.**
- It is similar to the Ridge Regression except that the penalty term contains only the absolute weights instead of a square of weights.
- Since it takes absolute values, hence, it can shrink the slope to 0, whereas Ridge Regression can only shrink it near to 0.
- It is also called as **L1 regularization.** The equation for the cost function of Lasso regression will be:

y = (y-intercept+slope*x) + (lamda*|slope|)

$$\sum_{i=1}^{M}(y_i - y'_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{n}\beta_j * x_{ij}\right)^2 + \lambda\sum_{j=0}^{n}|\beta_j|$$

- Some of the features in this technique are completely neglected for model evaluation.
- Hence, the Lasso regression can help us to reduce the overfitting in the model as well as the feature selection.

Note that instead of squaring the slope for the penalty term, the Lasso regression takes the absolute value of the slope (The absolute value will return all positive values). That means for the same value of lambda, the Lasso regression will produce less bias than the Ridge regression, as shown below:



Key Difference between Ridge Regression and Lasso Regression

- **Ridge regression** is mostly used to reduce the overfitting in the model, and it includes all the features present in the model. It reduces the complexity of the model by shrinking the coefficients.
- **Lasso regression** helps to reduce the overfitting in the model as well as feature selection.

Implementation in Jupyter

In [35]:
```python
# Import standard modules
import io
import urllib3

# importing the Pandas module
import pandas as pd

# Download dataset from our site
http = urllib3.PoolManager()
r = http.request('GET', 'https://hands-on.cloud/wp-content/uploads/2022/04/Dushanbe_house.csv')

# importing the dataset
Dushanbe = pd.read_csv(io.StringIO(r.data.decode('utf-8')))

# get dataset demo
Dushanbe.head()
```

Out[35]:

	Unnamed: 0	number_of_rooms	floor	area	latitude	longitude	price
0	0	1	1	58.0	38.585834	68.793715	330000
1	1	1	14	68.0	38.522254	68.749918	340000
2	2	3	8	50.0	NaN	NaN	700000
3	3	3	14	84.0	38.520835	68.747908	700000
4	4	3	3	83.0	38.564374	68.739419	415000

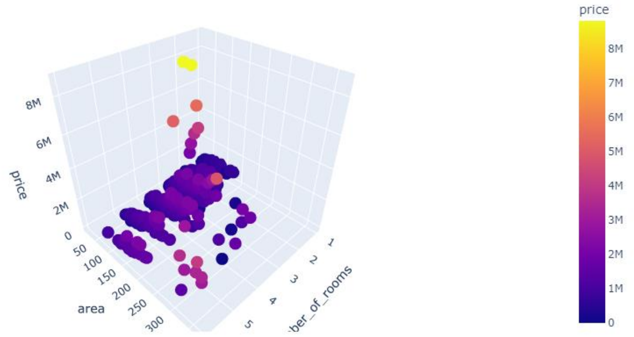
In [36]:
```python
# removing null values
Dushanbe.dropna(axis=0, inplace=True)

# display null values if exist
Dushanbe.isnull().sum()
```

Out[36]:
```
Unnamed: 0         0
number_of_rooms    0
floor              0
area               0
latitude           0
longitude          0
price              0
dtype: int64
```

In [37]:
```python
# importing the plotly module
import plotly.express as px

# plotting 3-d plot
fig = px.scatter_3d(Dushanbe, x='number_of_rooms', y='area', z='price',
              color='price')
fig.show()
```



Implementing Ridge regression

In [39]:
```
# taking the columns from the dataset
columns = Dushanbe.columns

# storing the input and output variables
Inputs = Dushanbe[columns[0:-1]]

#dependent variable
outputs = Dushanbe[columns[-1]]
```

In [40]:
```
# importing the module
from sklearn.model_selection import train_test_split

# splitting into test data and traind data for ridge regression
X_train, X_test, y_train, y_test = train_test_split(Inputs, outputs, test_size=0.25, random_state=42)
```

In [41]:
```
from sklearn.linear_model import Ridge

# alpha parameter 0.9 and initializing ridge regression
model = Ridge(alpha=0.9)

# ridge function
model.fit(X_train, y_train)
```

Out[41]:
```
Ridge
Ridge(alpha=0.9)
```

In [42]:
```
# predictive models of ridge regression models
y_pred = model.predict(X_test)
```

In [43]:
```
# Importing the required module
from sklearn.metrics import r2_score

# Evaluating model performance
print('R-square score is :', r2_score(y_test, y_pred))
```

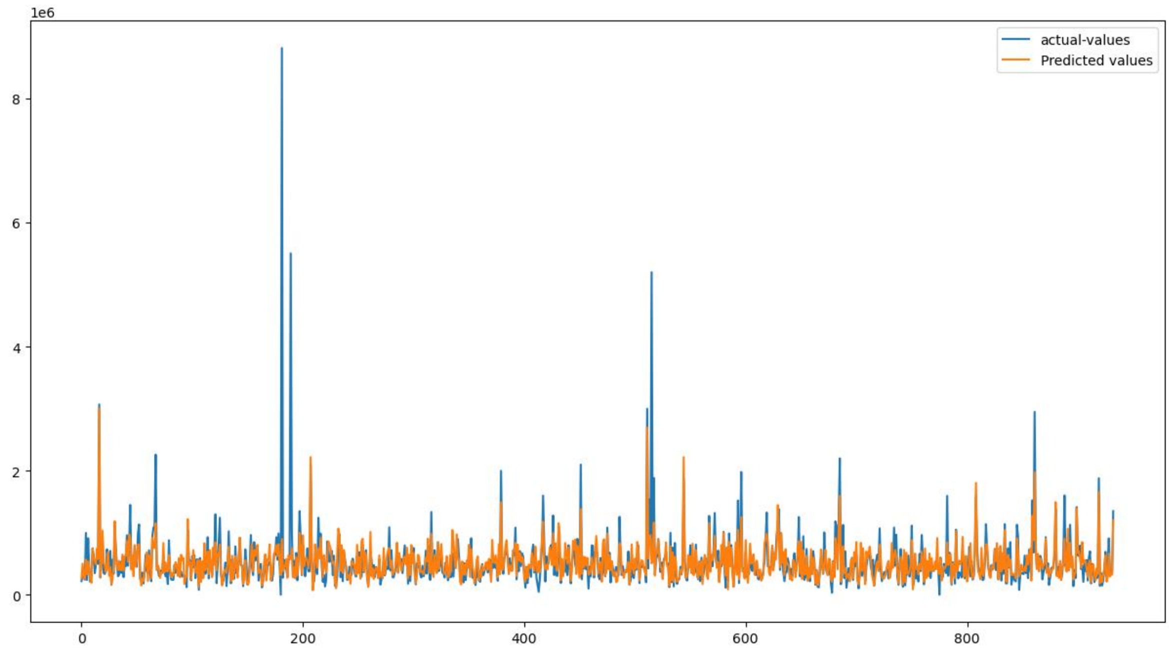
```
R-square score is : 0.3734242547708655
```

In [44]:
```
# importing the module
import matplotlib.pyplot as plt

# fitting the size of the plot
plt.figure(figsize=(15, 8))

# plotting the graphs for actual-value and predicted values
plt.plot([i for i in range(len(y_test))],y_test, label="actual-values")
plt.plot([i for i in range(len(y_test))],y_pred, label="Predicted values")

# showing the plotting of predictive modelling technique
plt.legend()
plt.show()
```



Implementing Lasso regression

In [45]:
```
# lasso regression implementation
from sklearn.linear_model import Lasso

# lasso regression select initialization
lasso_model = Lasso(alpha=0.9)

# training the lasso regression model
lasso_model.fit(X_train, y_train)
```

Out[45]:
```
▾ Lasso
Lasso(alpha=0.9)
```

In [46]:
```
# predictive model of lasso regression with test data
lasso_predictions = lasso_model.predict(X_test)
```

In [47]:
```
# Evaluating model performance to see accurate model
print('R-square score is :', r2_score(y_test, lasso_predictions))
```

```
R-square score is : 0.3742383050894751
```

In [48]:
```
# fitting the size of the plot
plt.figure(figsize=(15, 8))

# plotting the graphs for observed value and real values
plt.plot([i for i in range(len(y_test))],y_test, label="actual-values")
plt.plot([i for i in range(len(y_test))],lasso_predictions, label="Predicted values")

# showing the plotting of lasso regression
plt.legend()
plt.show()
```