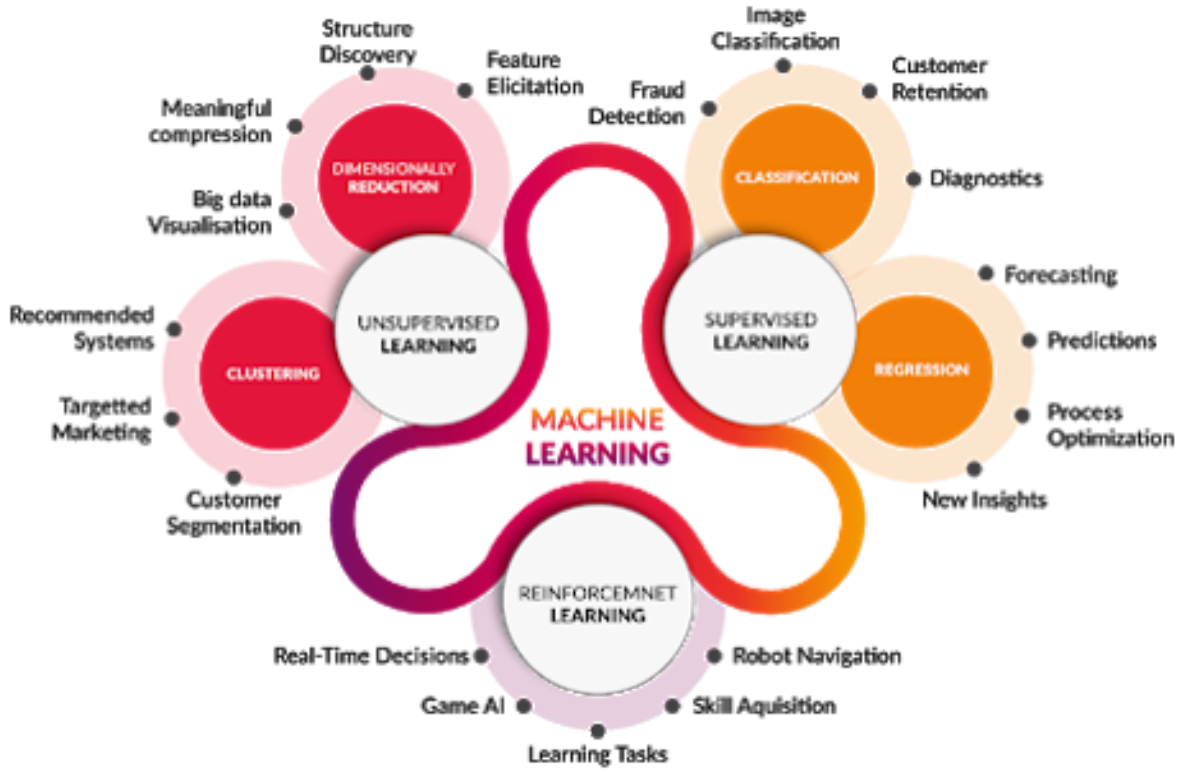
Data Science Nigeria: Introductory Machine Learning Training

INTRODUCTION TO REGRESSION

Remember,

Machine Learning Models/ Algorithms allows computer automate tasks that would otherwise take manual efforts, time, as well as resources. It learns how to interpret data to provide insight to humans.

> Machine Learning performance improves with experience



TYPES OF SUPERVISED LEARNING ALGORITHM

- Logistic Regresssor
- K Nearest Neighbour
- Support Vector Machine
- Decision Tree Classifier
- Random Forest Classifier
- Naïve Bayes etc.

- Linear Regresssor
- Polynomial √
- Support Vector √
- Decision Tree √
- Random Forest √ etc.

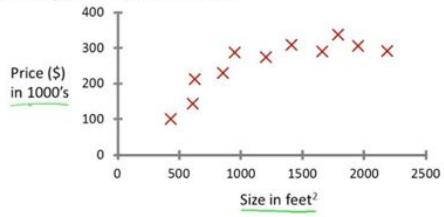
Regression is a set of processes used to estimate relationship between variables.

Regression:

A type of SL in which labelled data is used to train the Algorithm to make predictions in continuous form.

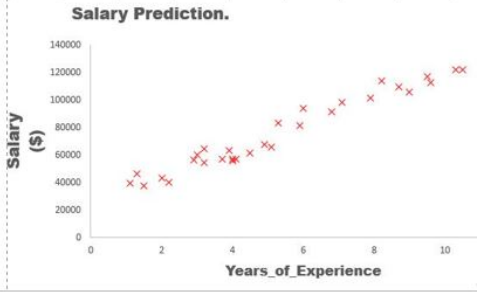
Example1:

Housing price prediction.



Example2:

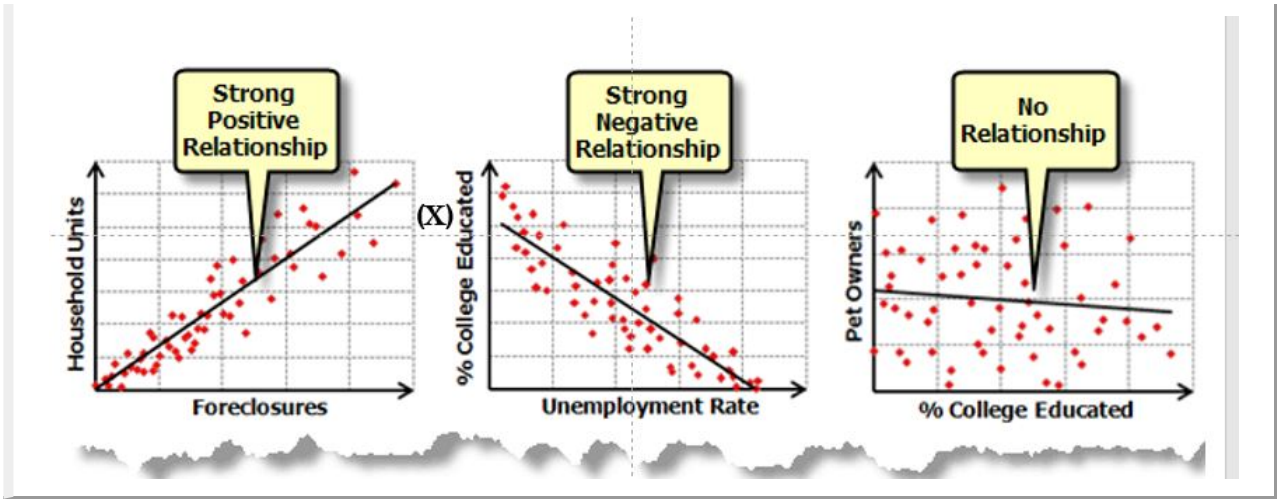
Salary Prediction.



LINEAR REGRESSION

Linear means **"Line"**. For instance, plotting one X axis against one Y axis. Linearity occurs when relationship between variables can be represented with a straight line.

Relationship between variables could be



Therefore, **ensure you establish linearity between your dataset before implementing a regression model**

Simple Linear Regression

A predictive model that uses a straight line to estimate the relationship between one variable (Predictor / independent variable) and another variable (Dependent /Predicted variable).

Remember the equation of straight line y = mx + b

where m = slope

```
b = intercept
```

Simple Linear Rgression equation is represented by a

$$\hat{y}_i = b_0 + b_1 x_i$$

where:

- **X:** Independent variable that causes y to change
- **bo:** Constant estimator/ intercept for y. Its value is the point at which regression line crosses y axis
- **b1:** Cofficient/multiplier that depict the proportion to which a unit change in x is inflated or deflated

To conduct a regression analysis, we need to solve for b0 and b1

$$b_1 = \frac{\Sigma[(X - \ddot{x})(y - \bar{y})]}{\Sigma[(X - \ddot{x})^2]}$$

where:

$\ddot{x}$: mean (X)

$\bar{y}$: mean(y)

n	X	y	X^2	y^2	Xy
1	-1	-1	1	1	1
2	1	2	1	4	2
3	2	3	4	9	6
4	4	3	16	9	12
5	6	5	36	25	30
6	7	8	49	64	56
Sum	**19**	**20**	**107**	**112**	**107**

$$b_1 = \frac{(\Sigma X)(\Sigma y) - n(\Sigma Xy)}{(\Sigma X)^2 - n\Sigma X^2}$$

$$b_1 = \frac{(19)(20) - 6(107)}{(19)^2 - 6(107)} = \frac{-262}{-281}$$

$$\boldsymbol{Slope\ (b_1) = 0.9324}$$

$$b0 = \frac{(\Sigma X)(\Sigma Xy) - \Sigma y \Sigma X^2}{(\Sigma X)^2 - n\Sigma X^2}$$

$$b0 = \frac{(19)(107) - (20)(107]}{(19)^2 - 6(107)} = \frac{-107}{-281}$$

$$\boldsymbol{intercept\ (b0) = 0.3808}$$

$$Sx = \sqrt{\frac{\Sigma X^2 - \frac{1}{n}(\Sigma X)^2}{n-1}} = \sqrt{\frac{107 - \frac{1}{6}(19)^2}{5}}$$

$$\boldsymbol{Standard\ Deviation\ (Sx) = 3.0605}$$

$$Sy = \sqrt{\frac{\Sigma Y^2 - \frac{1}{n}(\Sigma Y)^2}{n-1}} = \sqrt{\frac{112 - \frac{1}{6}(20)^2}{5}}$$

$$\boldsymbol{Standard\ Deviation\ (Sy) = 3.0111}$$

$$r(x,y) = \frac{Sx}{Sy}\ b1$$

$$r(x,y) = \frac{3.0605}{3.0111}\ 0.9324$$

$$\boldsymbol{Correlation\ (r(x,y)) = 0.9477}$$

- Predicting salary from years of experience
- Determining Glucose level from Age of patients

- Predicting salary from years of experience
- Predicting students grade based on total study time.
- Predicting examination score based on students' test score etc.

In [1]:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Creating a dataset for linear regression task

In [2]:

```
#Creating a list with 5 rows,2 columns
data = [[-1,-1],[1,2],[2,3],[4,3],[6,5],[7,8]]
data
```

Out[2]:

```
[[-1, -1], [1, 2], [2, 3], [4, 3], [6, 5], [7, 8]]
```

Data Representation

Since input data to our model **must be an N Dimensional Array**, the list created above must be represented as an Array

Option 1

In [3]:

```
# Create the pandas DataFrame with the list and add column headers
dataset= pd.DataFrame(data, columns = ['X', 'y'])

print(dataset)
print('Datatype of X and Y : ', type(dataset))


# Split the dataset into input (feature X) and output (Target y) from Pandas dataFrame r

X = dataset.iloc[:, :-1].values #All rows, all columns excluding the last column indexed
y = dataset.iloc[:, 1].values #All rows, column 1

print ('\n Input Feature X = ', X, '\n The shape of X = ',X.shape,  ' \n Output (Y) = ',
print('Datatype of X and Y : ', type(X), type(y))
```

```
   X  y
0 -1 -1
1  1  2
2  2  3
3  4  3
4  6  5
5  7  8
Datatype of X and Y :  <class 'pandas.core.frame.DataFrame'>

 Input Feature X =  [[-1]
 [ 1]
 [ 2]
 [ 4]
 [ 6]
 [ 7]]
 The shape of X =  (6, 1)
 Output (Y) =  [-1  2  3  3  5  8]
 Shape of y =  (6,)
Datatype of X and Y :  <class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

Option 2

In [4]:

```
#OR transforming the LIST data into a Numpy array of input (feature X) and output (Targe
Stack_data = np.column_stack(data)
print('Outcome of Stacking the list data', Stack_data)

X = Stack_data[0].reshape(-1,1) #To change the input feature to 2D array i.e (6,1) inste
y = Stack_data[1]

print ('\n Input Feature X = ', X, '\n The shape of X = ',X.shape,  ' \n Output (Y) = ',
print('\n Datatype of X and Y : ', type(X), type(y))
```

```
Outcome of Stacking the list data [[-1  1  2  4  6  7]
 [-1  2  3  3  5  8]]

 Input Feature X =  [[-1]
 [ 1]
 [ 2]
 [ 4]
 [ 6]
 [ 7]]
 The shape of X =  (6, 1)
 Output (Y) =  [-1  2  3  3  5  8]
 Shape of y =  (6,)

 Datatype of X and Y :  <class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

We are done with Data representation!!!

TASK

Given that X(an independent Variable) is a predictor of y(dependent) variable,

1. What linear regression equation best estimates y, based on X?
2. what new value of y will the linear regression equation suggest in line with the previous data X?
3. How well does the regression equation fit the data?

1. What linear regression equation best estimates y, based on X?

Using Scikit Learn (SKLEARN) Library, lets implement LinearRegression Class to train the Model and generate the regression equation

Model Training

The dataset has only 6 observations so, we use all that observations to train and test the model

In [5]:

```
# Fitting Simple Linear Regression to the Dataset
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X, y)
```

Out[5]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fals
e)
```

For Interpretation of model parameters: copy_X=True, fit_intercept=True, n_jobs=1 and normalize=False, click here (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegressio

In [6]:

```
regressor.score(X,y)
```

Out[6]:

```
0.8981055055474146
```

In [7]:

```
#Intercept b0

print(regressor.intercept_)
```

```
0.38078291814946663
```

In [8]:

```
#coefficient or slope of regression

regressor.coef_
```

Out[8]:

```
array([0.93238434])
```

Interpretaion of Linear Regression coefficient ~= 1.0

A unit increase in X causes almost the same increase in y

In [9]:

```
#TO check correlation between variables

cor = dataset.corr()

print(cor)
```

```
          X         y
X  1.000000  0.947684
y  0.947684  1.000000
```

Therefore, the equation of Linear regression for the task is

yhat = 0..9324X + 0.38078

ie. when X = 7, yhat =

2. what new values of y will the linear regression equation suggest in line with the previous data X?

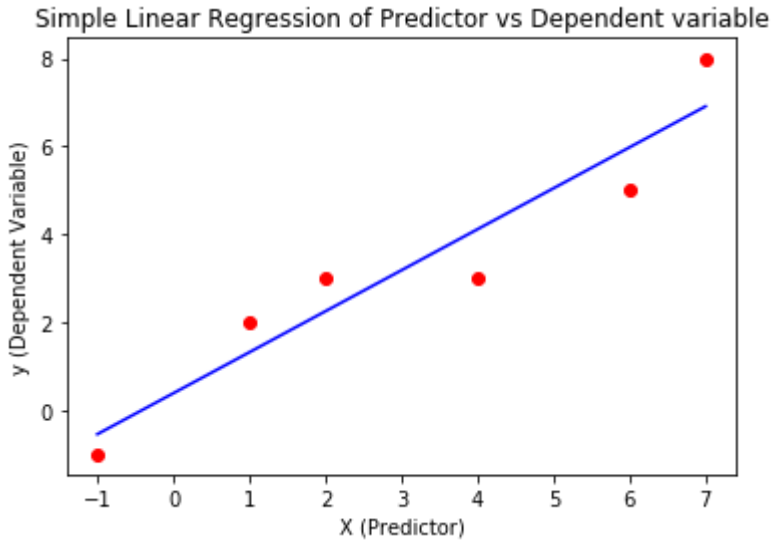
MODEL TESTING

In [10]:

```
#Predicting the Dataset results on input feature alone

y_pred = regressor.predict(X)
#Note: store the result in y_pred
```

In [11]:

```
# Visualising the results
plt.scatter(X, y, color = 'red') #Scatter plot of the X an y dataset

plt.plot(X, regressor.predict(X), color = 'blue')#Line plot of X against predicted Y
#(remember that output of regressor.predict is yhat)

plt.title('Simple Linear Regression of Predictor vs Dependent variable')
plt.xlabel('X (Predictor)')
plt.ylabel('y (Dependent Variable)')
plt.show()
```



In [12]:

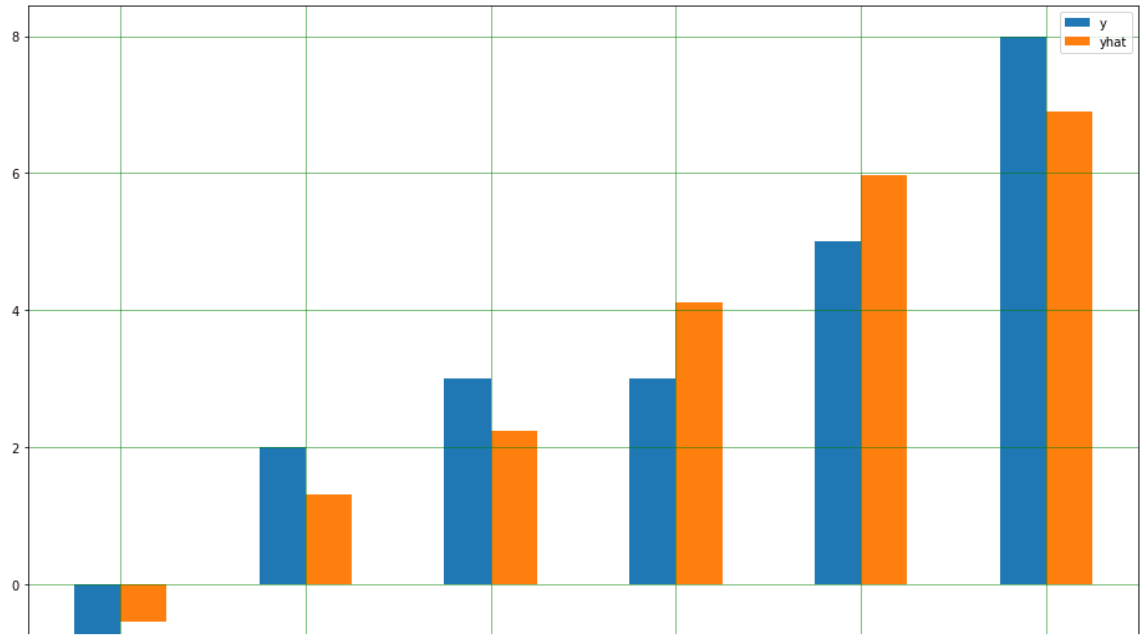
```
#Adding the result column to the existing dataset table

dataset['yhat'] = y_pred

dataset
```

Out[12]:

	X	y	yhat
0	-1	-1	-0.551601
1	1	2	1.313167
2	2	3	2.245552
3	4	3	4.110320
4	6	5	5.975089
5	7	8	6.907473

In [13]:

```
#A bar plot showing the diffrence between actual value (y) and predicted value (yhat)
dataset[['y','yhat']].plot(kind='bar',figsize=(16,10))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



3. How well does the regression equation fit the data?

This is a function of the model error established from model evaluation

Evaluation Metrics for Regression Models

Error is the difference between y and yhat at each observation

$$y_i - \hat{y}_i$$

cost function is the summation of all error function from all predictions

$$y_i - \hat{y}_i$$

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

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

Mean Squared Error (MSE) is the mean of the squared errors:

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

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

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

Do it yourself approach to solve the above equation

In [14]:

```
#Slicing out the Actual outout(y) and Predicted output (yhat)
result = dataset[['y', 'yhat']]
```

In [15]:

```
#creating a column in the result to store mean errors from all observations
n = len(dataset)

#Error from each prediction
result['error'] = dataset['y'].subtract(dataset['yhat'])

print(result)

#summation of all error function from all predictions
cost_function = abs(result['error']).sum()

Mean_Absolute_Error = cost_function / n




print()

print(cost_function)
print(Mean_Absolute_Error)
```

```
   y      yhat     error
0 -1 -0.551601 -0.448399
1  2  1.313167  0.686833
2  3  2.245552  0.754448
3  3  4.110320 -1.110320
4  5  5.975089 -0.975089
5  8  6.907473  1.092527

5.06761565836299
0.8446026097271649
```

```
## Using SKLEARN regression metrics
```

In [16]:

```
from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y, y_pred)))
```

```
Mean Absolute Error: 0.8446026097271649
Mean Squared Error: 0.7698695136417557
Root Mean Squared Error: 0.8774220840859636
```

Conclusion

The disparity between the actual y and predicted yhat and shows that the algorithm is not very accurate.

Many factors may have contributed to this inaccuracy:

- Relatively small data: We need to have a huge amount of data to get the best possible prediction.
- Bad assumptions: We made the assumption that this data has a linear relationship, but that might not be the case. Visualizing the data may help you determine that.
- Poor feature: The feature (X) may not have had a high enough correlation to the value (y) we were trying to predict.

Assignment

Download the **"Salary data"** attached in the repository The dataset contains:

- feature (X) representing years of experience
- Target (y) representing Salary

Task:

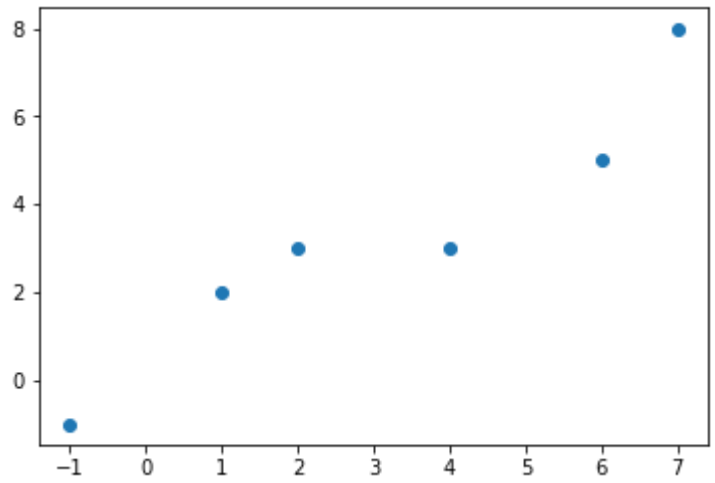
1. What linear regression equation best estimates Salary, based on Experience?
2. what new value of y will the linear regression equation suggest in line with 20% split of the previous Salary?
3. How well does the regression equation fit the data?

Lets explore some of the factors listed out under conclusion

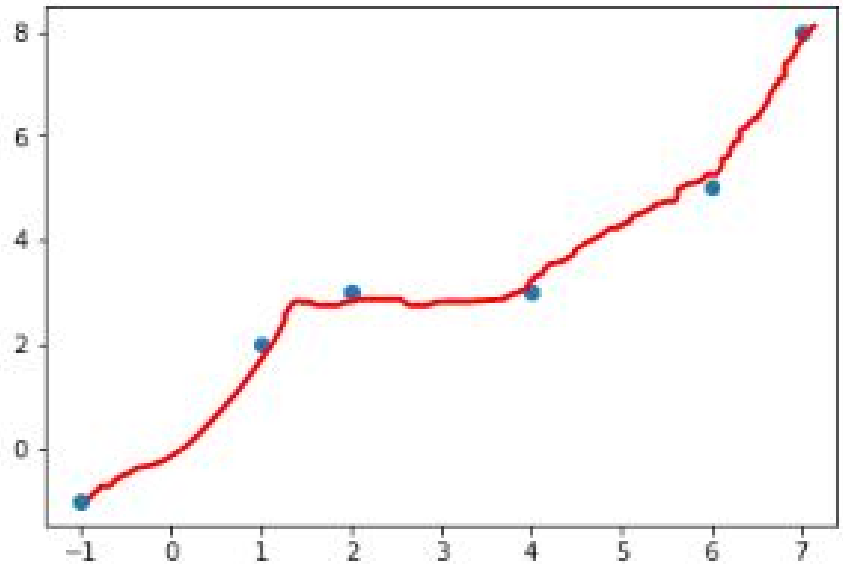
VISUALIZING THE DATA ON SCATTER PLOT

In [17]:

```
#VISUALIZING THE DATA ON SCATTER PLOT AS SUGGESTED UNDER THE SECOND FACTOR **BAD ASSUMPT

plt.scatter(X,y,)
plt.show()
```



As shown in the above figure, the relationship between X and Y can not be represented by staright line. it rather depict something like ths:



In this case, we try using **polynomial regression** which is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an nth degree polynomial in x.

In [18]:

```
#Importing Polynomial Feature class from SKLearn

from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 4) #Degree depicts the Curves
X_poly = poly_reg.fit_transform(X)
poly_reg.fit(X_poly, y) #Transform X as polynomial feature
```

Out[18]:

```
PolynomialFeatures(degree=4, include_bias=True, interaction_only=False)
```

In [19]:

```
regressor_2 = LinearRegression()
regressor_2.fit(X_poly, y)
```

Out[19]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fals
e)
```

In [20]:

```
regressor_2.score(X_poly,y)
```

Out[20]:

```
0.99828352180937
```

In [21]:

```
# Predicting a new result with Polynomial Regression
Poly_pred = regressor_2.predict(poly_reg.fit_transform(X))
```

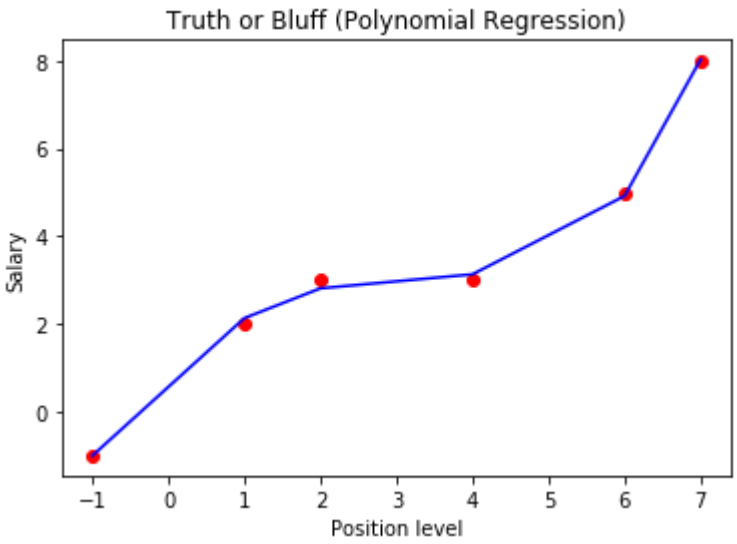
In [22]:

```
#print('regression coefficient= ', regressor_2.coef_)

print('regression intercept= ', regressor_2.intercept_)
```

```
regression intercept=  0.8507269789983924
```

In [23]:

```
# Visualising the Polynomial Regression results
plt.scatter(X, y, color = 'red')
plt.plot(X, regressor_2.predict(poly_reg.fit_transform(X)), color = 'blue')
plt.title('Truth or Bluff (Polynomial Regression)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```



In [24]:

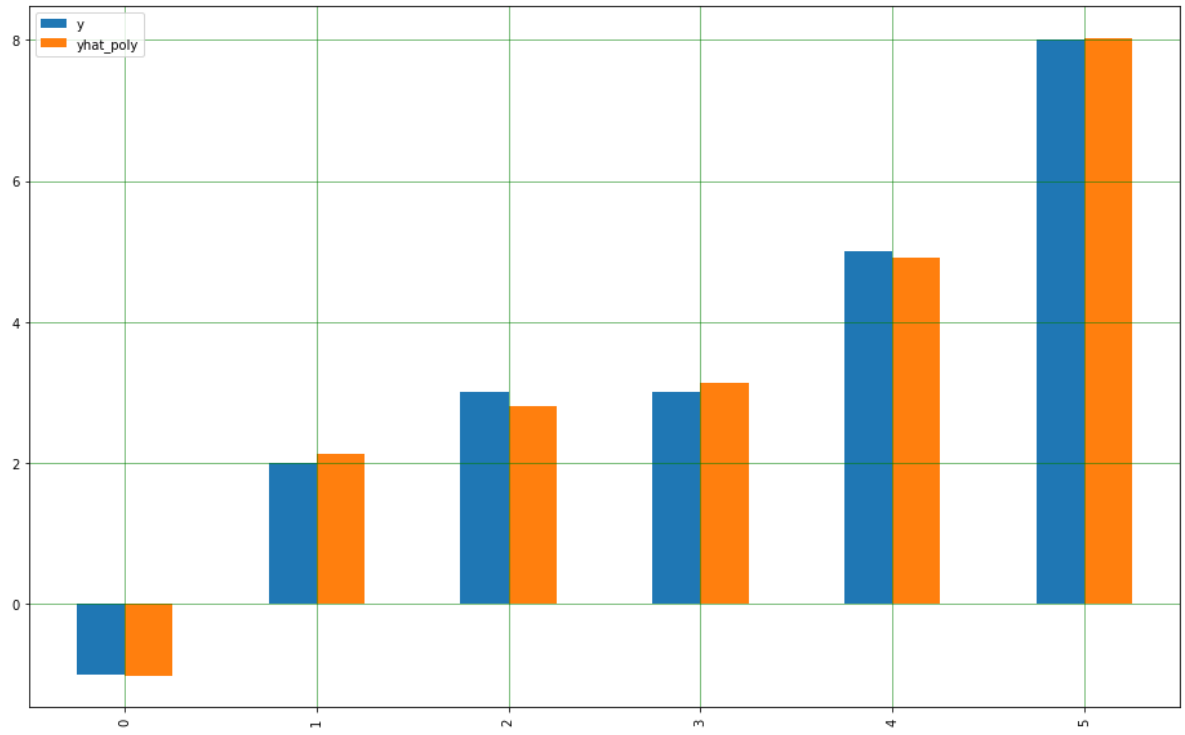
```
dataset['yhat_poly'] = Poly_pred

dataset
```

Out[24]:

	X	y	yhat	yhat_poly
0	-1	-1	-0.551601	-1.013732
1	1	2	1.313167	2.128164
2	2	3	2.245552	2.807754
3	4	3	4.110320	3.128164
4	6	5	5.975089	4.917609
5	7	8	6.907473	8.032041

In [26]:

```
#A bar plot showing the diffrence between actual value (y) and predicted value using pol
dataset[['y', 'yhat_poly']].plot(kind='bar',figsize=(16,10))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



In [ ]: