Lab07-Linear Regression :-

Lab Enhancement made by: ADAM DANIEL BIN SAIFUL AZLY (1211104378)

Linear Regression :-

- A type of Supervised Machine Learning algorithm.
- A fundamental statistical technique
- Used to **model the relationship** between a dependent variable and one or more independent variables

AKA **describes the relationship** between one or more predictor variables and a response variable.

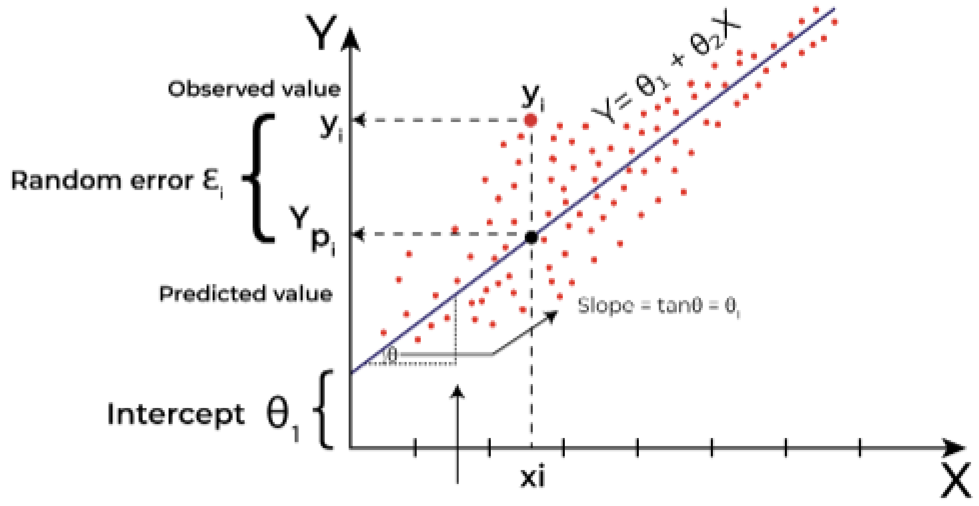
Goal of Linear Regression algorithm:

1. To **find the best Fit Line equation** that can predict the values based on the independent variables.
2. To **describe** how 1 Variable (the dependent variable) changes in relation to another Variable (the independent variable).

[Best Fit Line]:

Objective: To locate the **best-fit line** (**The error between the Predicted & Actual values** should be kept to a minimum.) There will be the least error in the best-fit line.

- Best Fit Line equation: Provides a **straight line** (Represents the relationship between the dependent and independent variables.)
- Slope of the line: Indicates "how much the dependent variable-Y changes for a unit change" in the independent variable(s)-X.


Y
Observed value
y_i
Random error ε_i
Y_{p_i}
Predicted value
Intercept θ_1
y_i
$Y = \theta_1 + \theta_2 X$
Slope = tanθ = θ_i
θ
xi
X

- Y : Dependent/Target variable
- X : Independent variable AKA Predictor of Y.

Note(s):-

1. *Here, X may be a single feature or multiple features representing the problem.*
2. *In the figure above, X (input) is the work experience and Y (output) is the salary of a person.*
3. *The regression line is the best-fit line for our model.*
4. *We utilize the cost function to compute the best values in order to get the best fit line since different values for weights or the coefficient of lines result in different regression lines.*
5. *There are many types of functions or modules that can be used for regression.*

[Linear function]

1. The **simplest** type of function.
2. Performs the task to **predict** a dependent variable value (y) based on a given independent variable (x).

Linear Regression From Scratch (Without Libraries)

```
!pip install latexify-py==0.3.1
```

```
import latexify
```

```
latexify.__version__
```

```
from IPython.display import display, Latex
```

```
def array_to_latex(X, precision = 8, suppress_small = True):
  latex_code = np.array2string(X, precision = precision,
                               separator=' & ',
                               suppress_small = suppress_small)
  latex_code = latex_code.replace("[", "\\begin{bmatrix}\n ", 1)
  latex_code = latex_code.rsplit("]", 1)
  latex_code = "\n\\end{bmatrix}".join(latex_code)

  latex_code = latex_code.replace("[", "")
  latex_code = latex_code.replace("] &", r" \\")
  latex_code = latex_code.replace("]", "")

  return latex_code

def print_latex(latex):
  display(Latex(latex))

def println_latex(latex):
  display(Latex(latex))
  print()
```

```
Collecting latexify-py==0.3.1
  Downloading latexify_py-0.3.1-py3-none-any.whl.metadata (3.9 kB)
Collecting dill>=0.3.2 (from latexify-py==0.3.1)
  Downloading dill-0.3.9-py3-none-any.whl.metadata (10 kB)
Downloading latexify_py-0.3.1-py3-none-any.whl (36 kB)
Downloading dill-0.3.9-py3-none-any.whl (119 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 119.4/119.4 kB 4.2 MB/s eta 0:00:00
Installing collected packages: dill, latexify-py
Successfully installed dill-0.3.9 latexify-py-0.3.1
```

```
# Code to generate LaTeX output

latex1 = r"""
\text{Given $m \times n$ data matrix $\mathbf{X}$ for our independant variables
and $y$ for our dependant variable}
"""
println_latex(latex1)

latex1 = r"""
\mathbf{X} =
  \begin{bmatrix}
    x_{1}^{(1)} & x_{2}^{(1)} & \cdots & x_{n}^{(1)} \\
    x_{1}^{(2)} & x_{2}^{(2)} & \cdots & x_{n}^{(2)} \\
    \vdots & \vdots & \ddots & \vdots \\
    x_{1}^{(m)} & x_{2}^{(m)} & \cdots & x_{n}^{(m)}
  \end{bmatrix}, \ \ \ \
\mathbf{y} =
  \begin{bmatrix}
    y^{(1)} \\
    y^{(2)} \\
    \vdots \\
    y^{(m)}
  \end{bmatrix}
"""
println_latex("$$" + latex1 + "$$")

latex1 = r"""
\text{where $m$ is the data size (number of points = number of observations)
 and $n$ is the data dimension (number of features),}
"""
println_latex(latex1)

latex1 = r"""
\text{we want to find the coefficient $c_0$, $c_1$, \dots, $c_n$ such that}
"""
println_latex(latex1)

latex1 = r"""
y^{(1)} = c_0 + c_1 x_{1}^{(1)} + c_2 x_{2}^{(1)} + \dots + c_n x_{n}^{(1)} + \varepsilon^{(
y^{(2)} = c_0 + c_1 x_{1}^{(2)} + c_2 x_{2}^{(2)} + \dots + c_n x_{n}^{(2)} + \varepsilon^{(

\vdots \\

y^{(m)} = c_0 + c_1 x_{1}^{(m)} + c_2 x_{2}^{(m)} + \dots + c_n x_{n}^{(m)} + \varepsilon^{(
"""
println_latex("$$" + latex1 + "$$")

latex1 = r"""
\text{minimizes}
"""
println_latex(latex1)

latex1 = r"""
\Epsilon = (\varepsilon^{(1)})^2 + (\varepsilon^{(2)})^2 + \dots + (\varepsilon^{(m)})^2
"""
println_latex("$$" + latex1 + "$$")

latex1 = r"""
\text{where $\varepsilon^{(1)}$, $\varepsilon^{(2)}$, \dots, $\varepsilon^{(m)}$ are gaussia
"""
println_latex(latex1)
```

```
latex1 = r"""
\text{We rewrite the above in matrix and vector form as below.}
"""
println_latex(latex1)

latex1 = r"""
\text{Given}
"""
println_latex(latex1)

latex1 = r"""
\mathbf{y} =
  \begin{bmatrix}
    y^{(1)} \\
    y^{(2)} \\
    \vdots \\
    y^{(m)}
  \end{bmatrix}, \ \ \ \
\mathbf{A} =
  \begin{bmatrix}
    1 & x_{1}^{(1)} & x_{2}^{(1)} & \cdots & x_{n}^{(1)} \\
    1 & x_{1}^{(2)} & x_{2}^{(2)} & \cdots & x_{n}^{(2)} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    1 & x_{1}^{(m)} & x_{2}^{(m)} & \cdots & x_{n}^{(m)}
  \end{bmatrix}, \ \ \ \
\mathbf{c} =
  \begin{bmatrix}
    c_{0} \\
    c_{1} \\
    \vdots \\
    c_{n}
  \end{bmatrix}, \ \ \ \
\mathbf{\varepsilon} =
  \begin{bmatrix}
```

```
    \varepsilon^{(1)} \\
    \varepsilon^{(2)} \\
    \vdots \\
    \varepsilon^{(m)}
  \end{bmatrix},
"""
println_latex("$$" + latex1 + "$$")

latex1 = r"""
  \begin{bmatrix}
    y^{(1)} \\
    y^{(2)} \\
    \vdots \\
    y^{(m)}
  \end{bmatrix} =
  \begin{bmatrix}
    1 & x_{1}^{(1)} & x_{2}^{(1)} & \cdots & x_{n}^{(1)} \\
    1 & x_{1}^{(2)} & x_{2}^{(2)} & \cdots & x_{n}^{(2)} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    1 & x_{1}^{(m)} & x_{2}^{(m)} & \cdots & x_{n}^{(m)}
  \end{bmatrix}
  \begin{bmatrix}
    c_{0} \\
    c_{1} \\
    \vdots \\
    c_{n}
  \end{bmatrix} +
  \begin{bmatrix}
    \varepsilon^{(1)} \\
    \varepsilon^{(2)} \\
    \vdots \\
    \varepsilon^{(m)}
  \end{bmatrix}
"""
println_latex("$$" + latex1 + "$$")

latex1 = r"""
\mathbf{y} = \mathbf{A} \mathbf{c} + \mathbf{\varepsilon}
"""
println_latex("$$" + latex1 + "$$")

latex1 = r"""
\text{and,}
"""
println_latex(latex1)

latex1 = r"""
\Epsilon = (\varepsilon^{(1)})^2 + (\varepsilon^{(2)})^2 + \dots + (\varepsilon^{(n)})^2 \\
\Epsilon = \sum_{j=1}^{n} (\varepsilon^{(j)})^2 \\
\Epsilon = \mathbf{\varepsilon}^T \mathbf{\varepsilon} \\
\Epsilon =  (\mathbf{y} - \mathbf{A} \mathbf{c})^T (\mathbf{y} - \mathbf{A} \mathbf{c}) \\
\Epsilon =   \mathbf{y}^T \mathbf{y}
           - \mathbf{y}^T (\mathbf{A} \mathbf{c}) - (\mathbf{A} \mathbf{c})^T \mathbf{y}
           + (\mathbf{A} \mathbf{c})^T (\mathbf{A} \mathbf{c}) \\
\Epsilon =   \mathbf{y}^T \mathbf{y}
           - \mathbf{y}^T \mathbf{A} \mathbf{c} - \mathbf{c}^T \mathbf{A}^T \mathbf{y}
           + \mathbf{c}^T \mathbf{A}^T \mathbf{A} \mathbf{c} \\
\Epsilon =   \mathbf{y}^T \mathbf{y}
           - 2 \mathbf{c}^T \mathbf{A}^T \mathbf{y}
           + \mathbf{c}^T \mathbf{A}^T \mathbf{A} \mathbf{c}
"""
println_latex("$$" + latex1 + "$$")

latex1 = r"""
\text{To Minimze $\Epsilon$, differentiating with respect of $\mathbf{c}$
 and set equal to $\mathbf{0}$ gives the "normal equations"}
"""
println_latex(latex1)

latex1 = r"""
\frac{\partial \Epsilon}{\partial \mathbf{c}}
 = - 2 \mathbf{A}^T \mathbf{y} + 2 \mathbf{A}^T \mathbf{A} \mathbf{c} = \mathbf{0} \\
\mathbf{A}^T \mathbf{A} \mathbf{c} = \mathbf{A}^T \mathbf{y}
"""
println_latex("$$" + latex1 + "$$")

latex1 = r"""
\text{The solution is thus}
"""
println_latex(latex1)

latex1 = r"""
\mathbf{c} = (\mathbf{A}^T \mathbf{A})^{-1}  \mathbf{A}^T \mathbf{y}
"""
println_latex("$$" + latex1 + "$$")
```

```
X = np.array([1, 3, 5, 6, 8, 9]).reshape((-1, 1))
y = np.array([4, 6, 9, 13, 14, 17]).reshape((-1, 1))
print(X)
print(y)
print(X.shape)
print(y.shape)
```

```
[[1]
 [3]
 [5]
 [6]
 [8]
 [9]]
[[ 4]
 [ 6]
 [ 9]
 [13]
 [14]
 [17]]
(6, 1)
(6, 1)
```

```
A = np.append(np.ones((X.shape[0], 1)), X, axis = 1)
print(A)
```

```
[[1. 1.]
 [1. 3.]
 [1. 5.]
```

```
      [1. 6.]
      [1. 8.]
      [1. 9.]]
```

```
AT = A.T
C = AT @ A
print(C)
```

```
[[  6.  32.]
 [ 32. 216.]]
```

```
det = C[0, 0] * C[1, 1] - C[0, 1] * C[1, 0]
print(det)
```

```
272.0
```

```
C_inv = [ [ C[1, 1], -C[0, 1] ],
          [-C[1, 0],  C[0, 0] ] ] / det
print(C_inv)
```

```
[[ 0.79411765 -0.11764706]
 [-0.11764706  0.02205882]]
```

```
# verify using the numpy inv function
C_inv = np.linalg.inv(C)
print(C_inv)
```

```
[[ 0.79411765 -0.11764706]
 [-0.11764706  0.02205882]]
```

```
coef = C_inv @ AT @ y
print(coef)
```

```
[[1.79411765]
 [1.63235294]]
```

```
import numpy as np

class MyLinearRegression:

  def __init__(self):
    pass

  def fit(self, X, y):
    A = np.append(np.ones((X.shape[0], 1)), X, axis = 1)
    AT = A.T
    C = AT @ A
    C_inv = np.linalg.inv(C)
    coef = C_inv @ AT @ y
    coef = coef.reshape(-1)
    self.intercept_ = coef[0]
    self.coef_ = coef[1:]
    return self

  def predict(self, X):
    y_pred = self.intercept_ + np.sum(self.coef_ * X, axis = 1).reshape((-1, 1))
    return y_pred

  def score(self, X, y):
    ssr = np.sum((y - self.predict(X))**2)
    sst = np.sum((y - np.mean(y))**2)
    r_sq = 1 - ssr / sst
    return r_sq

  def fit_predict(self, X, y):
    self.fit(X, y)
    return self.predict(X)

# calculate r_sq


X = np.array([1, 3, 5, 6, 8, 9]).reshape((-1, 1))
y = np.array([4, 6, 9, 13, 14, 17]).reshape((-1, 1))
print(X)
print(y)
print(X.shape)
print(y.shape)

model = MyLinearRegression()

model.fit(X, y)

r_sq = model.score(X, y)

print(f"coefficient of determination: {r_sq}")
print(f"intercept: {model.intercept_}")
print(f"slope: {model.coef_}")

y_pred = model.predict(X)
print(f"predicted response:\n{y_pred}")
```

```
[[1]
 [3]
 [5]
 [6]
 [8]
 [9]]
[[ 4]
 [ 6]
 [ 9]
 [13]
 [14]
 [17]]
(6, 1)
(6, 1)
coefficient of determination: 0.962502929458636
intercept: 1.7941176470588207
slope: [1.63235294]
predicted response:
[[ 3.42647059]
 [ 6.69117647]
 [ 9.95588235]
 [11.58823529]
 [14.85294118]
 [16.48529412]]
```

```
X = np.array( [ [  5, 35],
                [ 10, 50],
                [ 25, 50],
                [ 40, 15],
                [ 55, 90],
                [ 60, 45],
                [ 75, 75],
                [ 80, 15],
                [ 90, 60],
                [100, 40]
              ] )

y = np.array( [40.0, 30.0, 28.0, 40.0, 15.0, 25.0, 10.0, 30.0, 16.0, 15.0] ).reshape((-1, 1)

print(X)
print(y)

model = MyLinearRegression()

model.fit(X, y)

r_sq = model.score(X, y)

print(f"coefficient of determination: {r_sq}")
print(f"intercept: {model.intercept_}")
print(f"slope: {model.coef_}")

y_pred = model.predict(X)
print(f"predicted response:\n{y_pred}")
```

```
[[  5  35]
 [ 10  50]
 [ 25  50]
 [ 40  15]
 [ 55  90]
 [ 60  45]
 [ 75  75]
 [ 80  15]
 [ 90  60]
 [100  40]]
[[40.]
 [30.]
 [28.]
 [40.]
 [15.]
 [25.]
 [10.]
 [30.]
 [16.]
 [15.]]
coefficient of determination: 0.9506431728973666
intercept: 50.242032808300074
slope: [-0.20147336 -0.30447308]
predicted response:
[[38.57810812]
 [33.00364509]
 [29.9815447 ]
 [37.61600221]
 [11.75842063]
 [24.45234255]
 [12.29604969]
 [29.55706785]
 [13.84104554]
 [17.91577361]]
```

Types of Linear Regression:

There are several types of linear regression, each suited for different scenarios and data structures:-

(1) Simple Linear Regression

---

1. It helps to **understand the relationship between a single predictor variable and a response variable**. It involves a single independent variable and a single dependent variable and model the relationship between these 2 variables using a **Straight Line**, represented by the equation:

y=mx+b

- y : Predicted value (dependent variable).
- m : Slope of the line (signifies Slope of the Line: "How much y changes for a 1-unit increase in x. [1] Positive-m = Direct Relationship. [2] Negative-m = Inverse Relationship)
- x : Independent variable.
- b : y-intercept (the value of y when x is 0).

2. A more Statistical Context using the Linear Regression formula:

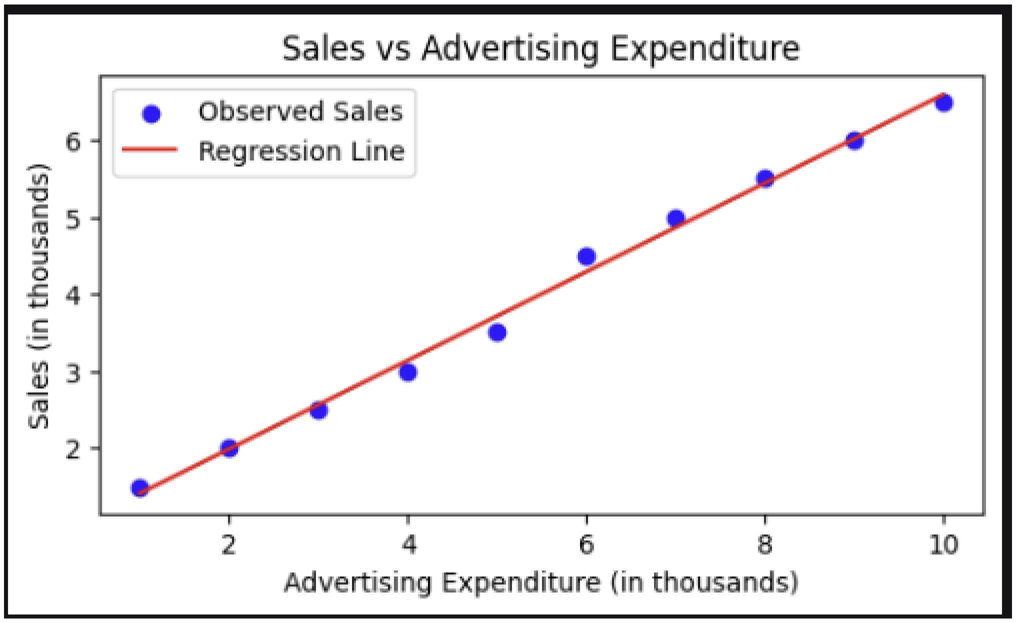
Y = β0 + β1x

- Y : Dependent variable
- X : Independent variable
- β0 : Intercept
- β1 : Slope

Note:- Parameters β0 & β1: Define the relationship between the independent variable X and the dependent variable Y in the regression equation.

Example: Consider a scenario where a company wants to predict sales based on advertising expenditure. By using simple linear regression, the company can determine if an increase in advertising leads to higher sales, and if so, to what extent.

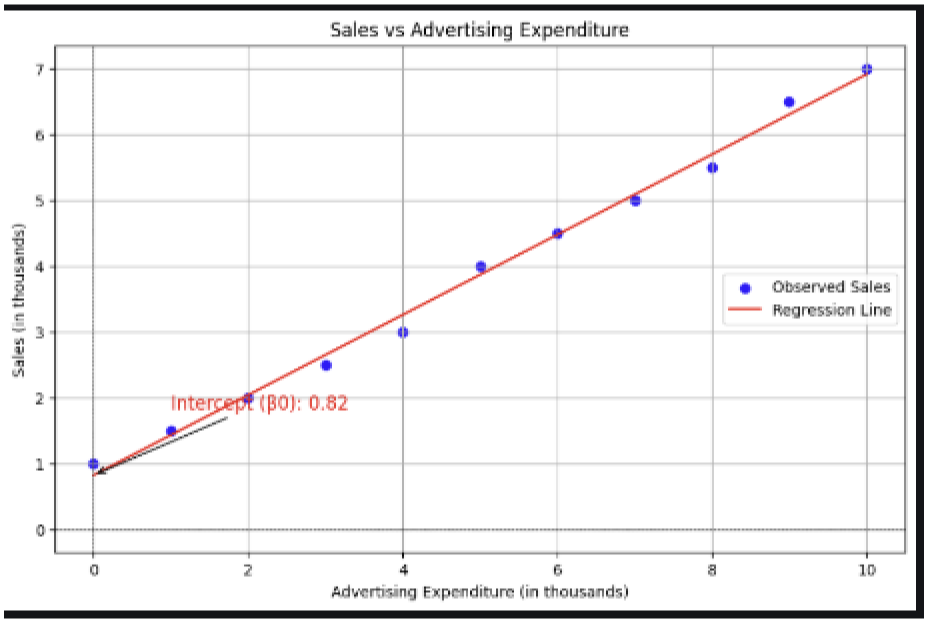
A graph shows a Relationship between advertising expenditure and sales using simple linear regression:



1. Intercept β0: Predicted value of the dependent variable Y when the independent variable X is zero AKA "The point where the regression line crosses the y-axis":

- Provides a baseline value for Y.
- Helps us understand the expected outcome when there is no influence from X.

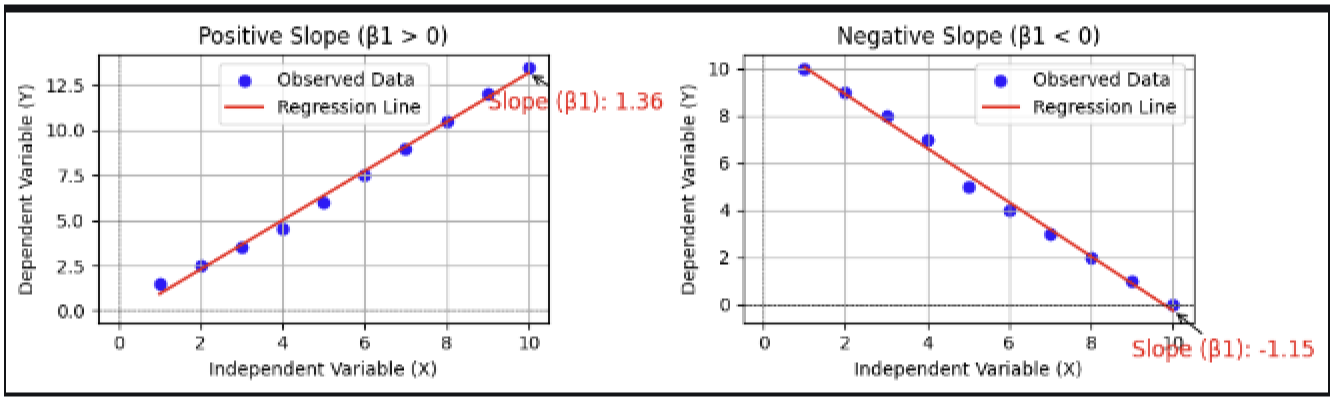
For example, if you were predicting sales based on advertising expenditure, it would indicate the estimated sales when no money is spent on advertising.



2. Slope β1:

- Positive-β1 value: X increases & Y increases = Direct relationship.
- Negative-β1 value: X increases leads to a Y decreases = Inverse relationship.

For instance, if β1=2, this would mean that for every additional unit spent on advertising, sales are expected to increase by 2 units.



3. Dependent variable: The dependent variable/response variable that you're *interested in understanding or predicting*.

4. Independent variables: The independent variables/Explanatory variables that *you think might affect* your dependent variable.

5. Regression equation: The formula that tries to *find* the best values for 'a' and 'b' to make the line of best fit AKA *express* how your independent variables(like studying, sleep, etc.) relate to your dependent variable (the test score). It provides a way to *predict* future outcomes based on the information you currently have.

[Model Creation]

1. To create our model, we must “learn” or estimate the values of regression coefficients β0 and β1.
2. Then, we can use the model to predict responses.
3. Use the principle of Least Squares.
4. Now consider:

- yi=β0+β1xi+εi=h(xi)+εi⇒εi=yi−h(xi)

5. Here, ei is a residual error in ith observation. So, our aim is to minimize the total residual error. We define the squared error or cost function, J as:

- J(β0,β1)=1/2n∑n,i=1 ε^2 i

6. And our task is to find the value of b0 and b1 for which J(b0, b1) is minimum! Without going into the mathematical details, we present the result here:

- β1=SSxy/SSxx
- β0=y¯−β1x¯

7. Where SSxy is the sum of cross-deviations of y and x:

- SSxy=∑n,i=1 (xi−x¯)(yi−y¯)=∑i=1nyixi−nx¯y¯

8. And SSxx is the sum of squared deviations of x:

- SSxx=∑n,i=1 (xi−x¯)^2=∑n,i=1 x^2i−n(x¯)^2

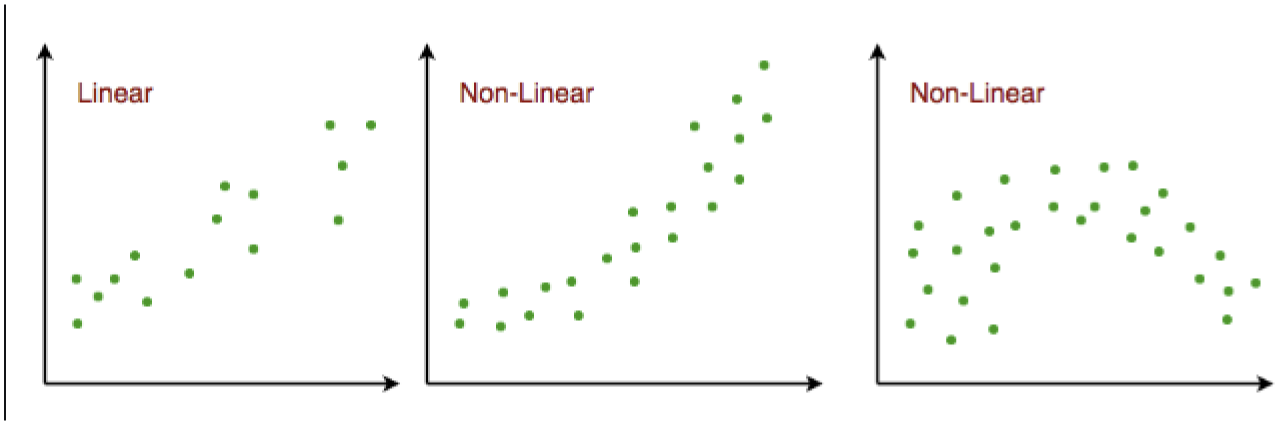
[Assumptions We Make in a Linear Regression Model]:

This technique assumes a linear relationship between the two variables, allowing us to predict the dependent variable based on the independent variable's value.

Given below are the basic assumptions that a linear regression model makes regarding a dataset on which it is applied

1. Linear relationship:

- Relationship between Response & Feature Bariables should be LINEAR.
- The linearity assumption can be tested using scatter plots.



1st figure represents linearly related variables whereas variables in the 2nd and 3rd figures are most likely non-linear. So, 1st figure will give better predictions using linear regression.

2. Little/No Multi-collinearity: Assumed that there is little / No Multicollinearity in the data. Multicollinearity occurs when the features (or independent variables) are not independent of each other.
3. Little/Autocorrelation: Autocorrelation occurs when the residual errors are not independent of each other.
4. No outliers:

- Assume that there are NO outliers in the data.
- Outliers are data points that are far away from the rest of the data.
- Outliers can affect the results of the analysis.

5. Homoscedasticity: Describe a situation in which the error term (that is, the "noise" or random disturbance in the relationship between the independent variables and the dependent variable) is the same across all values of the independent variables.

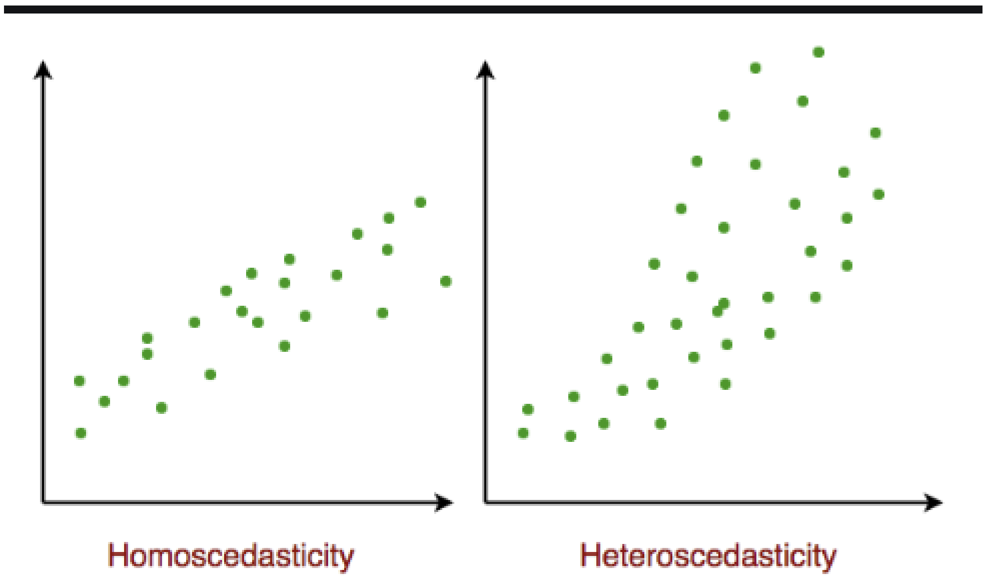


Figure 1 has homoscedasticity. Figure 2 has heteroscedasticity.

How to perform Simple Linear Regression

- Its implementation is done using Python and its libraries such as NumPy, Pandas, and scikit-learn.

Scenario/Example: Predicting a person's salary based on years of experience.

Step 1: Import the Required Libraries

Throughout the internet, self-study, redo the assignment lab enhancement(which is you enhance the lab by yourself by leanr and understand it propoerly and clearly and then you can make the lab that you enhance thta you and others can understrand) and do it yourself , I realize that I got to learn these labs these subjects effectively if I do learn it by myyself by referring to online resources that I prefer and easy to read and understand(cus the resources is infinite and varies) and follow it and practice it and study it by myself*

```
import numpy as np
```

Step 2: Import the Dataset using pandas and explore it.

```
import pandas as pd

dataset = pd.read_csv('Salary_Data.csv')

# Display the 1st few Rows of the Dataset
dataset.head()
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

- df.head() method: To RETRIEVE the 1st 5 Rows of the dataframe.
- df.columns attribute: To RETURN the name of the columns.

Note:- If we have more than 2 Attributes do 'target' Variable code from the Dataset for the y-Variable that we want to predict. For example:

```
# Add the target variable (house prices) to the DataFrame
# df['PRICE'] = boston.target
```

```
# Check for missing values
print(dataset.isnull().sum())
```

```
YearsExperience    0
Salary             0
dtype: int64
```

Step 3: Do Data Pre-Processing

**Note: The difference between the array and vector in these variables:

- Dependent variable must be in vector
- Independent variable must be an array itself

```
X = dataset.iloc[:,:-1].values  #independent variable array
y = dataset.iloc[:,1].values  #dependent variable vector
```

Step 4: Split the Dataset into Testing Data and Training Data

Reason(s):

1. To split the dataset into the test and train set, following the **20-80 / 30-70** policy respectively.
2. To train our Model according to **2 different variables**
3. To test our Model on the Test set.
4. To check whether the predictions made by the model on the test set data matches what was given in the dataset

The code:

1. sklearn.model_selection: sklearn's linear model library
2. train_test_split: [1] X_train (training data of matrix of features) [2] y_train (target values)

Note: If it matches, it implies that our model is accurate and is making the right predictions. We don't need to apply feature scaling for linear regression as libraries take care of it.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=1/3,random_state=0)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(20, 1)
(10, 1)
(20,)
(10,)
```

Note(s):-

- X_train : Portion of the Feature Data (Training the Model)
- X_test : Portion of the Feature Data (Testing the Model)
- y_train : Portion of the Target Data correspond to X_train(Training the Model)
- y_test : Portion of the Target Data correspond to X_test(Testingg the Model)
- X : Feature Matrix (Input Data) containing Independent Variable
- y : Target Vector (Output Data) containing Dependent Variable
- test_size=1/3 : Proportion of the Data for Test set:- [1] 1/3 AKA One-Third of the Data to allocate to Test set. [2] 2/3 for Training Set.
- random_state=0 : Seed for Random Number Generator to ENSURE that the Data is split the same way every time the code is run.

Step 5: Fit Linear Regression Model into the Training set

- From sklearn's linear model library, import **linear regression** class.
- Create an object for a linear regression class: **regressor**.
- To fit the regressor into the training set, call the **fit method function** to fit the regressor into the training set. Fit X_train (training data of matrix of features) into the target values y_train.
- Thus the model **learns** the correlation and "how to predict the dependent variables" based on the independent variable.

```
# Linear Regression from scikit-learn
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()

# Train the Model on the Training Data
regressor.fit(X_train, y_train) #actually produces the linear eqn for the data

# Print the intercept and coefficient
```

```
print(f"Intercept: {regressor.intercept_}")
print(f"Coefficient: {regressor.coef_}")
```

```
Intercept: 26816.192244031183
Coefficient: [9345.94244312]
```

Step 6: Predict the test set results

- LinearRegression() class: To create a Simple Regression Model. It is imported from sklearn.linear_model package.
- Create a vector containing all the predictions of the test set salaries. The predicted salaries are then put into the vector called y_pred.(contains prediction for all observations in the test set)
- 'predict' method makes the predictions for the test set. Hence, the input is the test set. The parameter for predict must be an array or sparse matrix, hence input is X_test.
- y_test : Real salary of the test set.
- y_pred : Predicted salaries.

```
y_test
```

```
array([ 37731., 122391.,  57081.,  63218., 116969., 109431., 112635.,
        55794.,  83088., 101302.])
```

```
y_pred = regressor.predict(X_test)
y_pred
```

```
array([ 40835.10590871, 123079.39940819,  65134.55626083,  63265.36777221,
       115602.64545369, 108125.8914992 , 116537.23969801,  64199.96201652,
        76349.68719258, 100649.1375447 ])
```

```
# Manually compute prediction using equation
y_pred_eq = regressor.intercept_ + regressor.coef_ * X_test
print(f"predicted response:\n{y_pred_eq}")

print(np.abs(y_pred - y_pred_eq) < 0.0000001)
```

```
predicted response:
[[ 40835.10590871]
 [123079.39940819]
 [ 65134.55626083]
 [ 63265.36777221]
 [115602.64545369]
 [108125.8914992 ]
 [116537.23969801]
 [ 64199.96201652]
 [ 76349.68719258]
 [100649.1375447 ]]
[[ True False False False False False False False False False]
 [False  True False False False False False False False False]
 [False False  True False False False False False False False]
 [False False False  True False False False False False False]
 [False False False False  True False False False False False]
 [False False False False False  True False False False False]
 [False False False False False False  True False False False]
 [False False False False False False False  True False False]
 [False False False False False False False False  True False]
 [False False False False False False False False False  True]]
```

```
r_sq = regressor.score(X_train, y_train)

print(f"coefficient of determination: {r_sq}")
print(f"intercept: {regressor.intercept_}")
print(f"slope: {regressor.coef_}")
```

```
coefficient of determination: 0.9381900012894278
intercept: 26816.192244031183
slope: [9345.94244312]
```

```
# Display the first few predictions alongside the actual values
Thepredictions = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(Thepredictions.head())
```

```
     Actual      Predicted
0   37731.0   40835.105909
1  122391.0  123079.399408
2   57081.0   65134.556261
3   63218.0   63265.367772
4  116969.0  115602.645454
```

Step 7: Visualizing the results

1. Plotting the points (observations)

Use 'matplotlib' to visualize the data & to plot real observation points ie plotting the real given values.

- X-axis : Years of experience
- Y-axis : Predicted salaries.

plt.scatter : Plots a scatter plot of the data. Parameters include :

- X – coordinate (X_train: number of years)
- Y – coordinate (y_train: real salaries of the employees)
- Color (Regression line in red and observation line in blue)

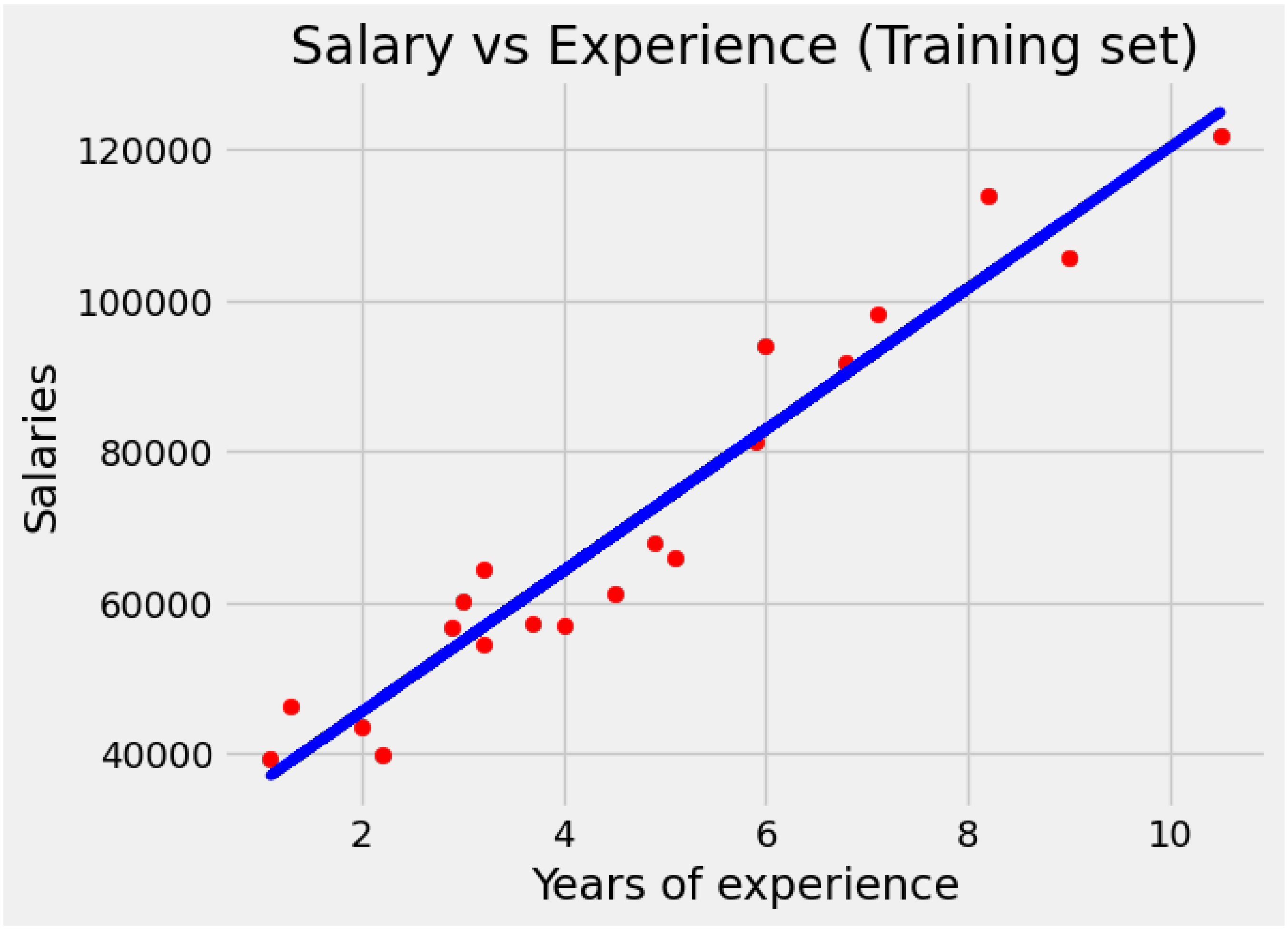
2. Plotting the regression line plt.plot have the following parameters :

- X – coordinates (X_train) – number of years
- Y – coordinates (predict on X_train) – prediction of X-train (based on a number of years).

Note: The y-coordinate is not y_pred because y_pred is predicted salaries of the test set observations

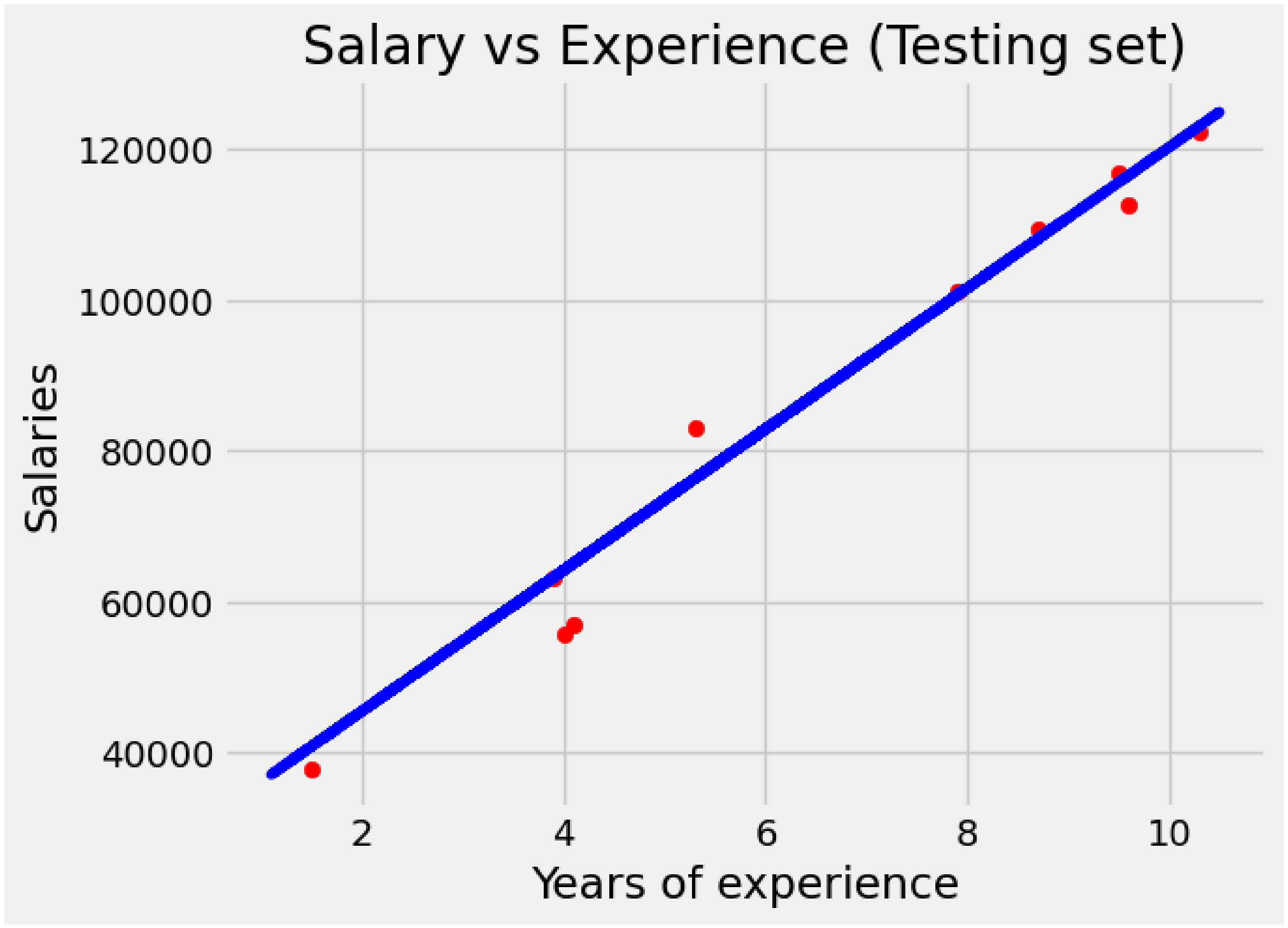
The below code generates a plot for the train set shown below:

```
import matplotlib.pyplot as plt
%matplotlib inline

#plot for the TRAIN

plt.scatter(X_train, y_train, color='red') # plotting the observation line

plt.plot(X_train, regressor.predict(X_train), color='blue') # plotting the regression line

plt.title("Salary vs Experience (Training set)") # stating the title of the graph

plt.xlabel("Years of experience") # adding the name of x-axis
plt.ylabel("Salaries") # adding the name of y-axis
plt.show() # specifies end of graph
```



The below code snippet generates a plot as shown below:

```
#plot for the TEST

plt.scatter(X_test, y_test, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue') # plotting the regression line

plt.title("Salary vs Experience (Testing set)")

plt.xlabel("Years of experience")
plt.ylabel("Salaries")
plt.show()
```



```
from sklearn.metrics import mean_squared_error, r2_score
# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Calculate R-squared score
r2 = r2_score(y_test, y_pred)
print(f"R-squared score: {r2}")
```

```
Mean Squared Error: 21026037.329511296
R-squared score: 0.9749154407708353
```

(2) Multiple Linear Regression

It helps to **understand the relationship between multiple predictor variables and a response variable**. It extends simple linear regression by incorporating multiple independent variables. The equation becomes:

Y = β0 + β1X1 + β2X2 + … + βpXp + ε

- Y: Response variable
- βj: Average effect on Y of a one unit increase in Xj, holding all other predictors fixed
- Xj: The jth predictor variable
- ε: Error term

Values for β0, β1, B2, … , βp are chosen using the **least square method** to minimizes the sum of squared residuals (RSS):

RSS = Σ(yi – ŷi)2

where:

- Σ: A sum
- yi: Actual response value for the ith observation
- ŷi: Predicted response value based on the multiple linear regression model

Reason: To find these coefficient estimates relies on matrix algebra.

Consider a dataset with p features(or independent variables) and one response(or dependent variable). Also, the dataset contains n rows/observations

We define:

- X (feature matrix) = a matrix of size n X p where xij denotes the values of the jth feature for ith observation.

  So,

(X11 ⋯ X1p

X21 ⋯ X2p

⋮ ⋱ ⋮

Xn1 ⋮ Xnp)

and

y (response vector) = a vector of size n where y_{i} denotes the value of response for ith observation.

$y=[y1$

$y2$

.

.

$yn]$

- The regression line for p features is represented as:

$h(Xi)=β0+β1Xi1+β2Xi2+…+βpXip$

where $h(Xi)$ is predicted response value for ith observation and $β0, β1, …, βp$ are the regression coefficients. Also, we can write:

$yi=β0+β1xi1+β2xi2+….+βpxip+εi$

or

$yi=(xi)+εi⇒εi=yi−h(xi)$

where $εi$ represents a residual error in ith observation. We can generalize our linear model a little bit more by representing feature matrix X as:

$X=(1\ x11 \cdots x1p$

$1\ x21 \cdots x2p$

$\vdots \vdots \ddots \vdots$

$1\ xn1 \cdots xnp)$

- So now, the linear model can be expressed in terms of matrices as:

$y=Xβ+ε$

where,

$β=[β0$

$β1$

.

.

$βp]$

and

ε=[ε1

ε2

.

.

εn]

- Now, we determine an estimate of b, i.e. b’ using the Least Squares method. As already explained, the Least Squares method tends to determine b’ for which total residual error is minimized. We present the result directly here:

β^=(X’X)−1X’y

where ‘ represents the transpose of the matrix while -1 represents the matrix inverse. Knowing the least square estimates, b’, the multiple linear regression model can now be estimated as:

y^=Xβ^

where y’ is the estimated response vector.

How to perform Multiple Linear Regression

Scenario/Example: Finding a correlation between House Price of Unit Area and Number of Convenience Stores.

Step 1: Import the Required Libraries

```
import pandas as pd
import numpy as np
```

Step 2: Import the CSV File

```
#Importing Data
df = pd.read_csv('Real_estate.csv')

df.drop('No', inplace = True, axis = 1)
# 'No' column: As an Index

print(df.head())
print(df.columns)
```

```
   X1 transaction date  X2 house age  X3 distance to the nearest MRT station  \
0             2012.917          32.0                                84.87882   
1             2012.917          19.5                               306.59470   
2             2013.583          13.3                               561.98450   
3             2013.500          13.3                               561.98450   
4             2012.833           5.0                               390.56840   

   X4 number of convenience stores  X5 latitude  X6 longitude  \
0                               10     24.98298     121.54024   
1                                9     24.98034     121.53951   
2                                5     24.98746     121.54391   
3                                5     24.98746     121.54391   
4                                5     24.97937     121.54245   

   Y house price of unit area  
0                        37.9  
1                        42.2  
2                        47.3  
3                        54.8  
4                        43.1  
Index(['X1 transaction date', 'X2 house age',
       'X3 distance to the nearest MRT station',
       'X4 number of convenience stores', 'X5 latitude', 'X6 longitude',
       'Y house price of unit area'],
      dtype='object')
```

Note(s):

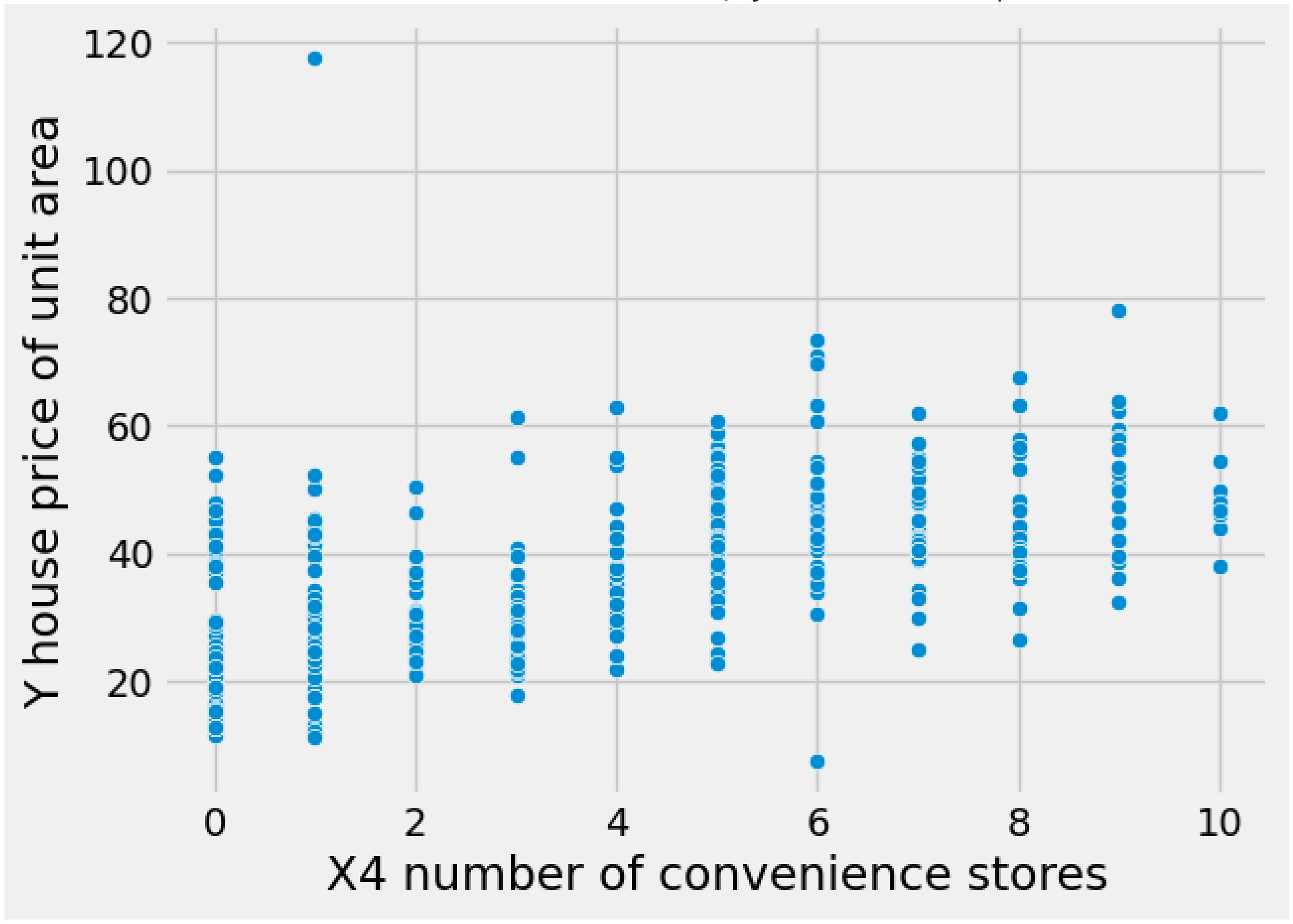
In the Dataset:-

- The column names starting with ‘X’ are the independent features in our dataset.
- The column ‘Y house price of unit area’ is the dependent variable column.
- As the number of independent or exploratory variables (X) is more than one, it is a Multilinear regression.

Step 3: Create a scatterplot to visualize the data.

```
# Plotting a scatterplot
import seaborn as sns
import matplotlib.pyplot as plt
sns.scatterplot(x='X4 number of convenience stores',
                y='Y house price of unit area', data=df)
```

```
<Axes: xlabel='X4 number of convenience stores', ylabel='Y house price of unit area'>
```



Reason: To visualize the relation between the ‘X4 number of convenience stores’ independent variable and the ‘Y house price of unit area’ dependent feature.

Step 4: Create Feature Variables

```
X = df.drop('Y house price of unit area', axis = 1)
y = df['Y house price of unit area']

print(X)
print(y)
```

```
     X1 transaction date  X2 house age  \
0                2012.917          32.0
1                2012.917          19.5
2                2013.583          13.3
3                2013.500          13.3
4                2012.833           5.0
..                    ...           ...
409              2013.000          13.7
410              2012.667           5.6
411              2013.250          18.8
412              2013.000           8.1
413              2013.500           6.5
```

```
     X3 distance to the nearest MRT station  X4 number of convenience stores  \
0                                  84.87882                               10
1                                 306.59470                                9
2                                 561.98450                                5
3                                 561.98450                                5
4                                 390.56840                                5
..                                      ...                              ...
409                              4082.01500                                0
410                                90.45606                                9
411                               390.96960                                7
412                               104.81010                                5
413                                90.45606                                9

     X5 latitude  X6 longitude
0       24.98298     121.54024
1       24.98034     121.53951
2       24.98746     121.54391
3       24.98746     121.54391
4       24.97937     121.54245
..           ...           ...
409     24.94155     121.50381
410     24.97433     121.54310
411     24.97923     121.53986
412     24.96674     121.54067
413     24.97433     121.54310

[414 rows x 6 columns]
0      37.9
1      42.2
2      47.3
3      54.8
4      43.1
       ...
409    15.4
410    50.0
411    40.6
412    52.5
413    63.9
Name: Y house price of unit area, Length: 414, dtype: float64
```

Reason: To model the data for the Feature Variables Creation containing:-

- X-variable: Independent Variables
- y-variable: Dependent Variable

Step 5: Split data into train and test sets

```
# Create Train & Test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X , y, test_size = 0.3, random_state = 101)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(289, 6)
(125, 6)
(289,)
(125,)
```

Note(s):

- train_test_split() method: To create 'train' & 'test' sets, then the feature variables are passed in the method.
- test size: 0.3, which means 30% of the data goes into test sets.
- train set: 0.7, which means 70%.
- random state: given for data reproducibility

Step 6: Create a Linear Regression Model

```
# Create a Regression Model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

Step 7: Fit the Model with Training Data

- fit() method: To fit the Data.
- Fit the Model with the Training Model.
- The Model gains information and knowlegde about the Statistics of the Training Model.

```
# Fit the Model
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Step 8: Make Predictions on the Test Data Set.

- model.predict() method: To make Predictions on the X_test Data
- Test Data: an UNSEEN Data.
- The Model has NO Knowledge about the Statistics of the Test Data.

```
# Make Predictions
predictions = model.predict(X_test)


r_sq = model.score(X, y)

print(f"coefficient of determination: {r_sq}")
print(f"intercept: {model.intercept_}")
print(f"slope: {model.coef_}")

y_pred = model.predict(X)
print(f"predicted response:\n{y_pred}")
```

```
coefficient of determination: 0.5802223765661338
intercept: -18595.05503451969
slope: [ 4.83926101e+00 -2.74749120e-01 -4.18860818e-03  1.18123112e+00
  2.42384317e+02  2.33991349e+01]
predicted response:
[48.05954278 48.72701875 49.68749135 49.28583269 47.06140631 31.18866295
 39.38911349 47.54691381  8.5339677  34.5789675  33.66793688 53.89943294
 41.27851262 26.95553667 46.37025359 38.74415932 51.62507832 37.39582323
 47.42677323 48.26293154 35.10776133 50.28337977 29.53209137 48.77696624
 35.30473459 31.90812422 47.20109264 42.31456083 43.14702697 46.6386477
 12.71500084 41.31454145 30.87406656 47.15700315 48.10837957 34.45503565
 30.37857322 31.48372925 45.50409246 46.6934425  14.06742397 16.0075392
 36.73328773 41.23026871 48.4721038  40.54478865 47.56814504 37.7602422
 12.0736566   9.17202039 42.21116333 24.68926603 34.65362371 41.99940522
 46.31441949 23.1083508  43.9553153  46.2014402  12.95585061 40.79239057
 33.78284173 51.14653971 29.61511665 48.49957871 33.80668812 44.1928161
 47.62740646 47.42153665 43.74411552 46.27784489 55.02682345 35.45051134
 45.61060308 14.82063073 53.79669429 33.01277843 38.26216993 28.30527298
 33.10068415 28.75517451 45.57022814 38.29001222 46.44841272 24.46567309
 45.02702847 50.38641657 32.0791477  16.38172296 37.42879996 15.95586215
 42.81321638 38.99458517 28.11040209 23.41914151 35.31913556 44.73762204
 54.27845595 37.84503374 47.84830793 54.27845595 45.33972278 39.47728995
 48.00159021 44.26307337 36.09178068 49.60710648 48.76235412 32.23209675
 36.44365782 36.58145675 45.51346446 45.51335899 33.76896382 43.57904854
 44.30595729 39.47782591  0.75066854 12.93036536 33.57091525 48.19501677
 42.40590314 43.93495232 41.20147144 43.95831177 48.02859874 48.40808813
 35.7041623  49.83005743 37.48194231 44.11534274 45.4942412  36.17979809
 41.0459903  48.81959723 37.25030623 31.48690954 44.90135328 45.9193634
 38.42798244 40.09617228 47.09510117 39.82608171 42.63806242 43.93495232
 31.10470624 47.20907598 40.73052468 48.11033139 15.08329623 41.04584022
 40.95375462 55.38508698 32.28704658 43.67864674 15.98006429 14.82063073
 26.35762375 42.7603896  46.0562187  40.10394991 47.64473233 42.04716292
 12.59789274 47.42153665 41.13218335 36.48827849 52.4332349  45.9734211
 37.5875205  31.94846605 12.19893826 49.07685066 55.02682345 38.06239771
 52.43323969 36.27028988 12.63830383 45.60290525 46.54336092 39.44063357
  8.88658151 52.50729872 33.56968145 15.55290746 23.46649922 30.44889865
 29.81168651 24.19775376 43.12220556 13.3236246  44.43963558 40.16857297
 38.8563499  49.26999501 15.60855454 40.23479801 38.61072377 43.05448789
 40.4574746  47.20160222 34.24066096 48.93912385 33.85952596 44.43866147
 28.75517451 37.16973014 51.17703175 34.37356579 29.60313197 41.89388746
 50.23424358 44.02303164 37.82572669 52.7170926  35.32733501 47.41992126
```

```
32.44263851 42.13562343 43.53329366 43.20582592 45.65126043 39.42371875
49.72698264 38.99458517 42.51637052 48.8097468  13.27065152 44.24427856
35.13949937 26.77886036 36.17979809 12.76370021 11.69242248 45.03000164
34.07247013 44.22386707 55.85946097 38.08375879 37.73221069 35.14759051
33.2425385  47.63351353 33.94576569 47.22333423 38.69869119 46.64483477
44.24994361 32.17950939 31.86109329  7.09423749 40.19268114 23.50249808
51.58970208 25.91813223 48.38687081  8.18725886 37.841461   39.33938416
52.4332349  33.23951832 37.32689765 29.64139827 45.56605199 35.80561434
40.74141983 40.85231741 33.79828583 37.20420925 46.53559547 32.86533457
42.30380873 44.91934627 40.47035604 42.16604382 43.25327973 48.26293154
37.55180253 30.93907702 44.26307337 40.4930635  48.54234695 54.63188022
32.4306958  33.08611108 44.25118605 36.86830777 51.99620001 37.52432761
48.74061653 48.12868281 31.03656991 45.82241719 39.13258948 43.70076869
45.85251825 30.52446712 43.07876464 34.74054504 14.82717561 48.50264837
45.69116631 29.48529942 31.17044745 32.95481188 34.43839965 46.28694458
42.23260833 22.98807375 47.84830793 24.85604948 32.82052814 41.05953639
```

```
y_pred = model.predict(X)
print(f"predicted response:\n{y_pred}")

# Manually compute prediction using equation
y_pred_eq = model.intercept_ + np.sum(model.coef_ * X, axis = 1).values.reshape((-1, 1))
# Access the underlying NumPy array with .values before reshaping
print(f"predicted response:\n{y_pred_eq}")

print(np.abs(y_pred - y_pred_eq) < 0.0000001)
```

```
predicted response:
[48.05954278 48.72701875 49.68749135 49.28583269 47.06140631 31.18866295
 39.38911349 47.54691381  8.5339677  34.5789675  33.66793688 53.89943294
 41.27851262 26.95553667 46.37025359 38.74415932 51.62507832 37.39582323
 47.42677323 48.26293154 35.10776133 50.28337977 29.53209137 48.77696624
 35.30473459 31.90812422 47.20109264 42.31456083 43.14702697 46.6386477
 12.71500084 41.31454145 30.87406656 47.15700315 48.10837957 34.45503565
 30.37857322 31.48372925 45.50409246 46.6934425  14.06742397 16.0075392
 36.73328773 41.23026871 48.4721038  40.54478865 47.56814504 37.7602422
 12.0736566   9.17202039 42.21116333 24.68926603 34.65362371 41.99940522
 46.31441949 23.1083508  43.9553153  46.2014402  12.95585061 40.79239057
 33.78284173 51.14653971 29.61511665 48.49957871 33.80668812 44.1928161
 47.62740646 47.42153665 43.74411552 46.27784489 55.02682345 35.45051134
 45.61060308 14.82063073 53.79669429 33.01277843 38.26216993 28.30527298
 33.10068415 28.75517451 45.57022814 38.29001222 46.44841272 24.46567309
 45.02702847 50.38641657 32.0791477  16.38172296 37.42879996 15.95586215
 42.81321638 38.99458517 28.11040209 23.41914151 35.31913556 44.73762204
 54.27845595 37.84503374 47.84830793 54.27845595 45.33972278 39.47728995
 48.00159021 44.26307337 36.09178068 49.60710648 48.76235412 32.23209675
 36.44365782 36.58145675 45.51346446 45.51335899 33.76896382 43.57904854
 44.30595729 39.47782591  0.75066854 12.93036536 33.57091525 48.19501677
 42.40590314 43.93495232 41.20147144 43.95831177 48.02859874 48.40808813
 35.7041623  49.83005743 37.48194231 44.11534274 45.4942412  36.17979809
 41.0459903  48.81959723 37.25030623 31.48690954 44.90135328 45.9193634
 38.42798244 40.09617228 47.09510117 39.82608171 42.63806242 43.93495232
 31.10470624 47.20907598 40.73052468 48.11033139 15.08329623 41.04584022
 40.95375462 55.38508698 32.28704658 43.67864674 15.98006429 14.82063073
 26.35762375 42.7603896  46.0562187  40.10394991 47.64473233 42.04716292
 12.59789274 47.42153665 41.13218335 36.48827849 52.4332349  45.9734211
 37.5875205  31.94846605 12.19893826 49.07685066 55.02682345 38.06239771
 52.43323969 36.27028988 12.63830383 45.60290525 46.54336092 39.44063357
  8.88658151 52.50729872 33.56968145 15.55290746 23.46649922 30.44889865
 29.81168651 24.19775376 43.12220556 13.3236246  44.43963558 40.16857297
 38.8563499  49.26999501 15.60855454 40.23479801 38.61072377 43.05448789
 40.4574746  47.20160222 34.24066096 48.93912385 33.85952596 44.43866147
 28.75517451 37.16973014 51.17703175 34.37356579 29.60313197 41.89388746
 50.23424358 44.02303164 37.82572669 52.7170926  35.32733501 47.41992126
 32.44263851 42.13562343 43.53329366 43.20582592 45.65126043 39.42371875
 49.72698264 38.99458517 42.51637052 48.8097468  13.27065152 44.24427856
 35.13949937 26.77886036 36.17979809 12.76370021 11.69242248 45.03000164
 34.07247013 44.22386707 55.85946097 38.08375879 37.73221069 35.14759051
 33.2425385  47.63351353 33.94576569 47.22333423 38.69869119 46.64483477
 44.24994361 32.17950939 31.86109329  7.09423749 40.19268114 23.50249808
 51.58970208 25.91813223 48.38687081  8.18725886 37.841461   39.33938416
 52.4332349  33.23951832 37.32689765 29.64139827 45.56605199 35.80561434
 40.74141983 40.85231741 33.79828583 37.20420925 46.53559547 32.86533457
 42.30380873 44.91934627 40.47035604 42.16604382 43.25327973 48.26293154
 37.55180253 30.93907702 44.26307337 40.4930635  48.54234695 54.63188022
 32.4306958  33.08611108 44.25118605 36.86830777 51.99620001 37.52432761
 48.74061653 48.12868281 31.03656991 45.82241719 39.13258948 43.70076869
 45.85251825 30.52446712 43.07876464 34.74054504 14.82717561 48.50264837
 45.69116631 29.48529942 31.17044745 32.95481188 34.43839965 46.28694458
 42.23260833 22.98807375 47.84830793 24.85604948 32.82052814 41.05953639
 45.22491663 46.66832989 48.16053216 36.45323901 46.53359793 38.15663114
 55.72740213 35.59669332 11.74802502 42.68328309 40.19831349 44.69790471
 30.84475715 40.54478865 48.8578799  45.60370757 34.20647025 12.93036536
 24.18055926 11.98926899 33.56857844 43.98441528 40.39803096 42.27414074
```

Step 9: Evaluate the Model with certain Metrics.

Multi-Linear Regresion Model is evaluated using these Metrics:-

- mean_squared_error: Mean of the sum of residuals
- mean_absolute_error: Mean of the absolute errors of the model.

Mean Square Error formula (MSE):

$$1/n \sum_{i=0}^{n} (y - \bar{y})\, 2$$

Mean Absolute Error formula (MAE):

$$1/n \sum_{i=0}^{n} |y - \overline{y}|$$

- y = actual value
- y bar = predictions

Reason(s):-

1. To compare with the Mean of the target variable so that we can understand "how well our model is predicting".
2. The less the error, the better the model performance is.

```
# Model Evaluation
from sklearn.metrics import mean_squared_error, mean_absolute_error
print('Mean Absolute Error:', mean_absolute_error(y_test, predictions))
print('Mean Squared Error:', mean_squared_error(y_test, predictions))
```

```
Mean Absolute Error: 5.392293684756193
Mean Squared Error: 46.21179783492909
```

(3) Polynomial Linear Regression

- A form of linear regression in which the *relationship between the independent variable x and dependent variable y* is modeled as an "nth-degree polynomial".
- It fits a Non-Linear Relationship between the value of x and the corresponding conditional mean of y, denoted E(y | x).

Choosing a Degree for Polynomial Regression:-

- The choice of degree for polynomial regression is a trade-off between (1) bias and (2) variance.
- (1) Bias : The Tendency of a model to consistently **predict the same value**, regardless of the true value of the dependent variable.
- (2) Variance : The Tendency of a model to **make different predictions for the same data point**, depending on the specific training data used.
- [1] Higher-degree polynomial : Reduce bias but increase variance = overfitting. [2] Lower-degree polynomial : Reduce variance but increase bias.

A number of methods for choosing a degree for polynomial regression:-

1. Cross-validation and using information criteria such as Akaike information criterion (AIC).
2. Bayesian information criterion (BIC).

How to perform Polynomial Linear Regression

Scenario/Example: Finding a relationship between Position Level and Salary.

Step 1: Import all Neccesary Libraries for Data Analysis and Machine Learning Task, Load the Dataset using Pandas, prepares the Data for Modelling by handling missing Values and encoding Categorial Data splits the data into training and testing sets and standardizes the numerical features using StandardScaler.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import warnings

from sklearn.preprocessing import LabelEncoder
from sklearn.impute import KNNImputer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score

warnings.filterwarnings('ignore')

df = pd.read_csv('Position_Salaries.csv')

X = df.iloc[:,1:2].values
y = df.iloc[:,2].values
print(X)
```

```
[[ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]]
```

Output:

array([[1, 45000, 0], [2, 50000, 4], [3, 60000, 8], [4, 80000, 5], [5, 110000, 3], [6, 150000, 7], [7, 200000, 6], [8, 300000, 9], [9, 500000, 1], [10, 1000000, 2]], dtype=int64)

Step 2: Create a Linear Regression Model, Fit the Model to the provided data and establish a Linear Relationship between the Independent & Dependent variables.

```
from sklearn.linear_model import LinearRegression
lin_reg=LinearRegression()
lin_reg.fit(X,y)
```

```
▾ LinearRegression
LinearRegression()
```

Step 3: Perform Quadratic & Cubic Regression by generating Polynomial Features from the Original Data, adn Fit Linear Regression Models to these Features.

```
from sklearn.preprocessing import PolynomialFeatures
poly_reg2=PolynomialFeatures(degree=2)
X_poly=poly_reg2.fit_transform(X)
lin_reg_2=LinearRegression()
lin_reg_2.fit(X_poly,y)

poly_reg3=PolynomialFeatures(degree=3)
X_poly3=poly_reg3.fit_transform(X)
lin_reg_3=LinearRegression()
lin_reg_3.fit(X_poly3,y)
```

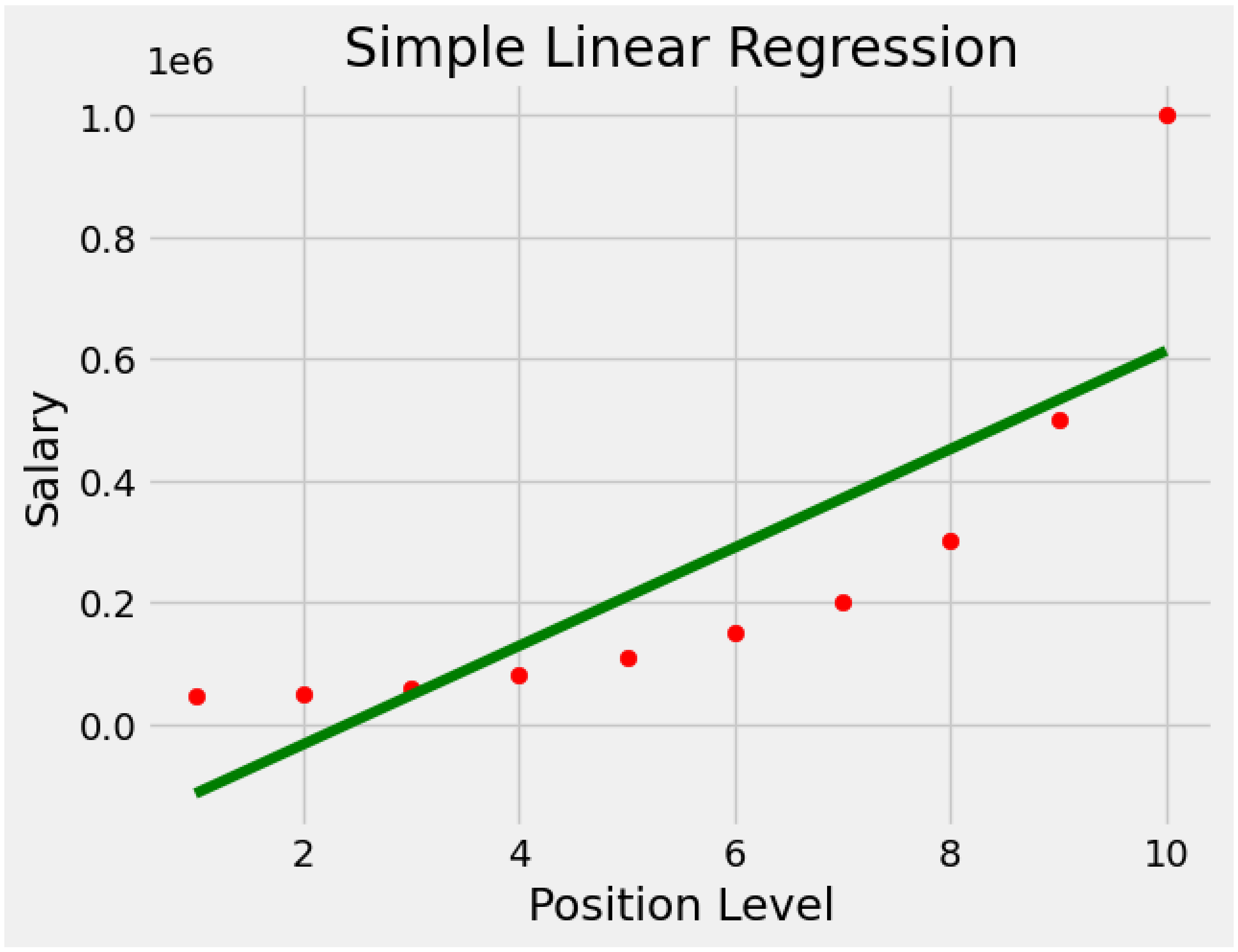
```
▾ LinearRegression
LinearRegression()
```

Code Output:

1. Enables Non-Linear Relationship between Independent & Dependent variables.

Step 4: Create a Scatter Plot of the Data Point for Simple Linear Regression.

```
plt.scatter(X,y,color='red')
plt.plot(X,lin_reg.predict(X),color='green')
plt.title('Simple Linear Regression')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```

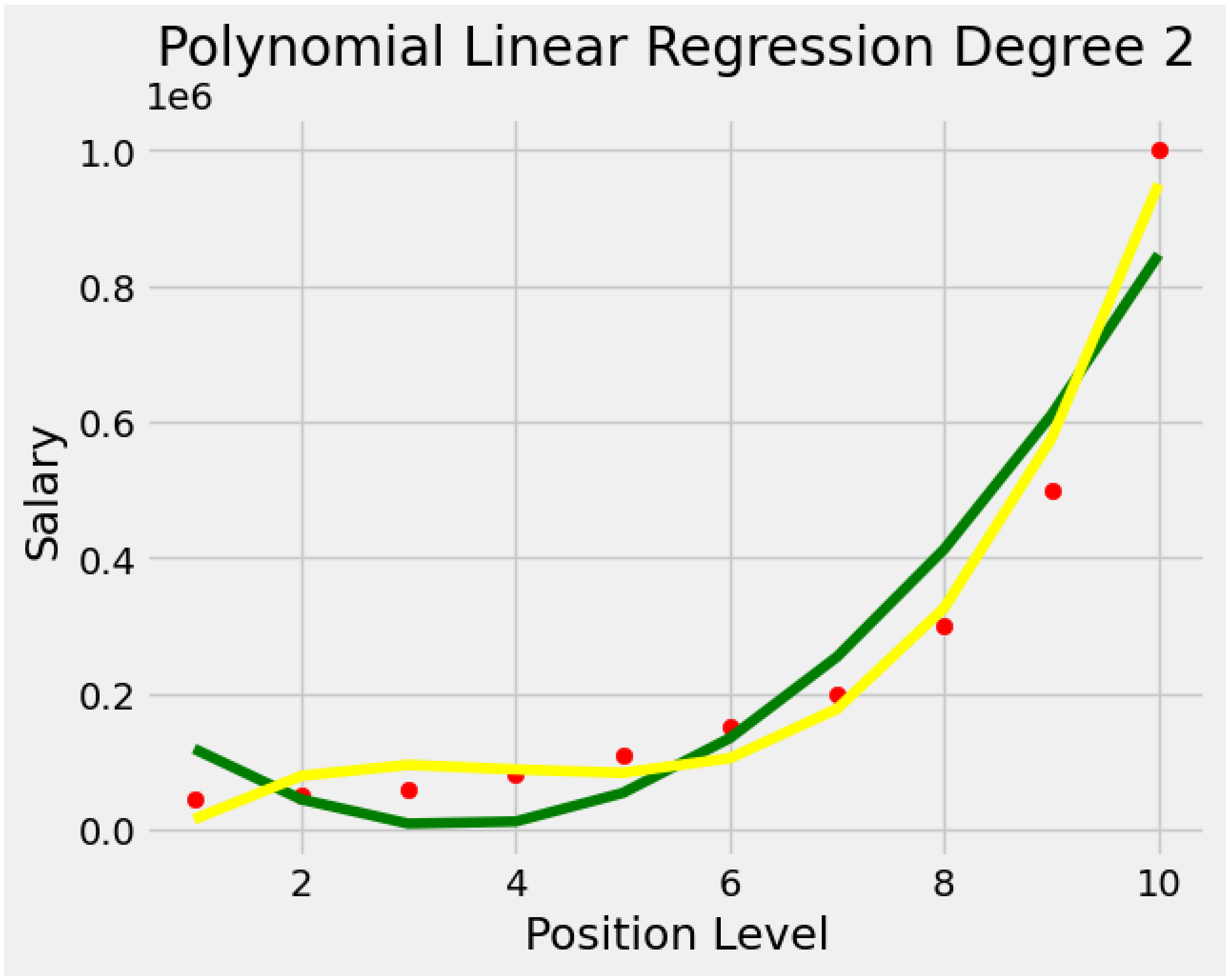


Reason:

To *visualize* the Linear Relationship between Position Level & Salary.

Step 5: Create a Scatter Plot of the Data Points for Polynomial Linear Regression and Overlay the Predicted Quadratic & Cubic Regression Lines.

```
plt.style.use('fivethirtyeight')
plt.scatter(X,y,color='red')
plt.plot(X,lin_reg_2.predict(poly_reg2.fit_transform(X)),color='green')
plt.plot(X,lin_reg_3.predict(poly_reg3.fit_transform(X)),color='yellow')
plt.title('Polynomial Linear Regression Degree 2')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```

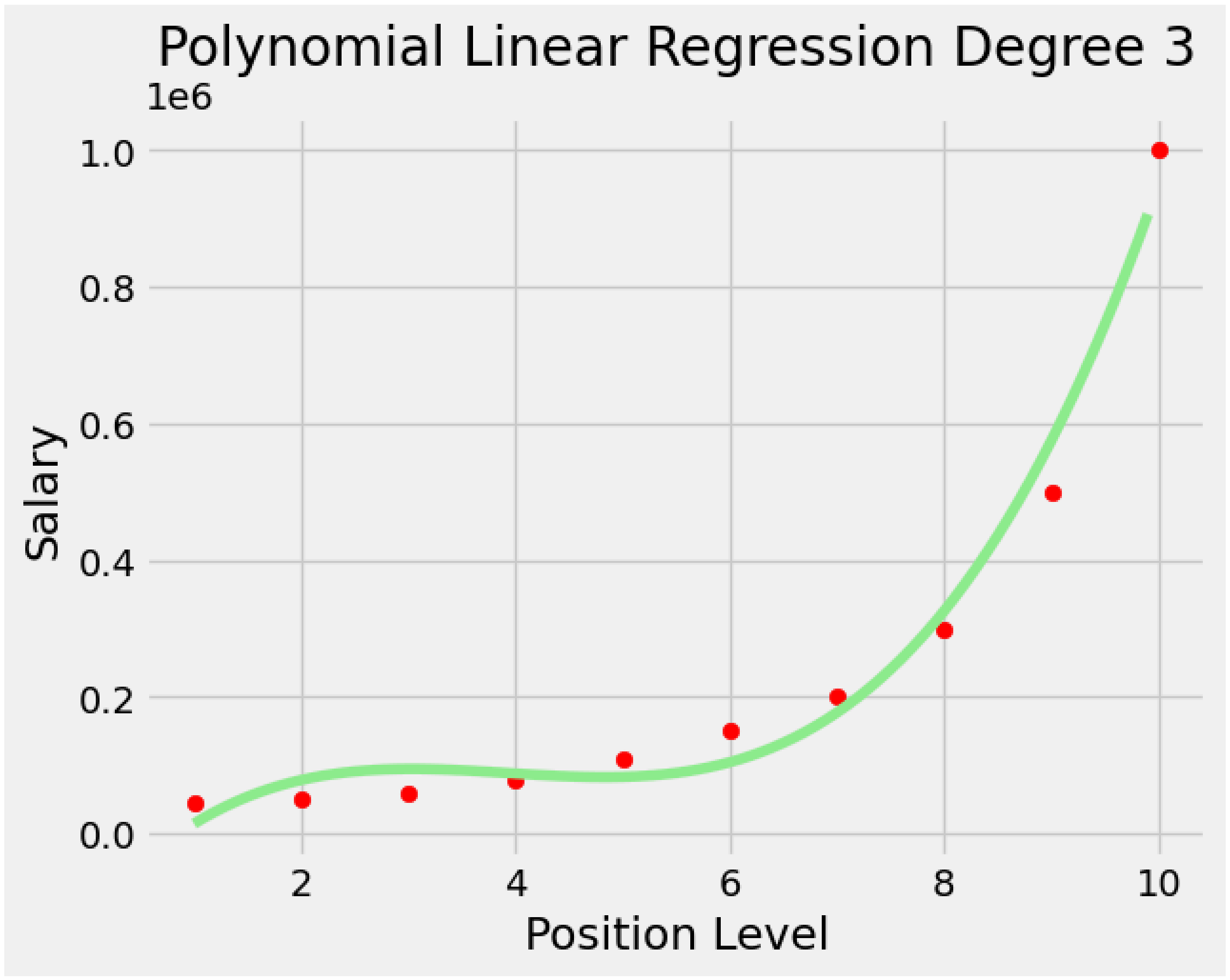


Reason(s):

1. To visualizes the nonlinear relationship between position level and salary.
2. To compares the fits of Quadratic & Cubic Regression Models.

Step 6: Visualizes the relationship between Position Level & Salary using Cubic Regression and generates a Continuous Prediction Line for a broader range of position levels.

```
plt.style.use('fivethirtyeight')
X_grid=np.arange(min(X),max(X),0.1) # This will give us a vector.We will have to convert thi
X_grid=X_grid.reshape((len(X_grid),1))
plt.scatter(X,y,color='red')
plt.plot(X_grid,lin_reg_3.predict(poly_reg3.fit_transform(X_grid)),color='lightgreen')

#plt.plot(X,lin_reg_3.predict(poly_reg3.fit_transform(X)),color='green')
plt.title('Polynomial Linear Regression Degree 3')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```



Another Scenarion of Polynomial Linear Regression:

(1) Method 1: Polynomial Features: include_bias = False

(2) Method 2: Polynomial Features: include_bias = True

```
import numpy as np
from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures


X = np.array([1, 2, 4, 6, 8, 9]).reshape((-1, 1))
y = np.array([8, 6, 1, 3, 13, 17]).reshape((-1, 1))

print(X)
print(y)
```

```
[[1]
 [2]
 [4]
 [6]
 [8]
 [9]]
[[ 8]
 [ 6]
 [ 1]
 [ 3]
 [13]
 [17]]
```

(1) Method 1: Polynomial Features: include_bias = False

```
transformer = PolynomialFeatures(degree = 2, include_bias = False)

X_ = transformer.fit_transform(X)

print(X_)
```

```
[[ 1.  1.]
 [ 2.  4.]
 [ 4. 16.]
 [ 6. 36.]
 [ 8. 64.]
 [ 9. 81.]]
```

```
# NOTE: fit_intercept is default to be True
model = LinearRegression()

model.fit(X_, y)
```

```
▾ LinearRegression
LinearRegression()
```

```
r_sq = model.score(X_, y)

print(f"coefficient of determination: {r_sq}")
print(f"intercept: {model.intercept_}")
print(f"slope: {model.coef_}")

y_pred = model.predict(X_)
print(f"predicted response:\n{y_pred}")

# Manually compute prediction using equation
y_pred_eq = model.intercept_ + np.sum(model.coef_ * X_, axis = 1).reshape((-1, 1))
print(f"predicted response:\n{y_pred}")

print(np.abs(y_pred - y_pred_eq) < 0.0000001)
```

```
coefficient of determination: 0.973565731926936
intercept: [13.85207101]
```

```
slope: [[-5.92159763  0.7056213 ]]
predicted response:
[[ 8.63609467]
 [ 4.83136095]
 [ 1.4556213 ]
 [ 3.72485207]
 [11.63905325]
 [17.71301775]]
predicted response:
[[ 8.63609467]
 [ 4.83136095]
 [ 1.4556213 ]
 [ 3.72485207]
 [11.63905325]
 [17.71301775]]
[[ True]
 [ True]
 [ True]
 [ True]
 [ True]
 [ True]]
```

(2) Method 2: Polynomial Features: include_bias = True

```
X_ = PolynomialFeatures(degree=2, include_bias = True).fit_transform(X)

print(X_)
```

```
[[ 1.  1.  1.]
 [ 1.  2.  4.]
 [ 1.  4. 16.]
 [ 1.  6. 36.]
 [ 1.  8. 64.]
 [ 1.  9. 81.]]
```

```
# NOTE: fit_intercept is now equals to False
model = LinearRegression(fit_intercept = False)

model.fit(X_, y)
```

```
LinearRegression

LinearRegression(fit_intercept=False)
```

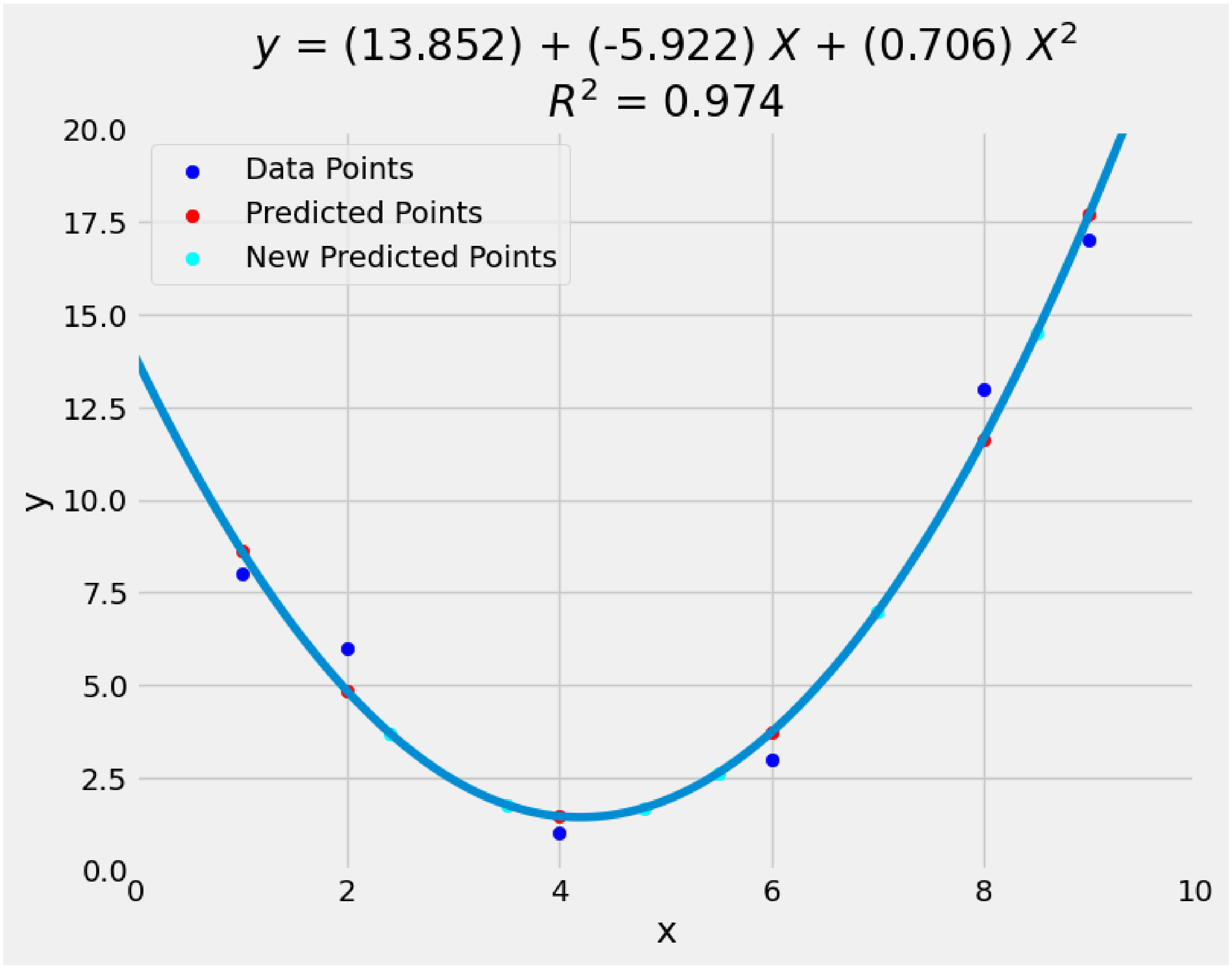
```
r_sq = model.score(X_, y)

print(f"coefficient of determination: {r_sq}")
print(f"intercept: {model.intercept_}")
print(f"slope: {model.coef_}")
```

```
y_pred = model.predict(X_)
print(f"predicted response:\n{y_pred}")

# Manually compute prediction using equation
y_pred_eq = model.intercept_ + np.sum(model.coef_ * X_, axis = 1).reshape((-1, 1))
print(f"predicted response:\n{y_pred}")

print(np.abs(y_pred - y_pred_eq) < 0.0000001)
```

```
coefficient of determination: 0.9735657319269361
intercept: 0.0
slope: [[13.85207101 -5.92159763  0.7056213 ]]
predicted response:
[[ 8.63609467]
 [ 4.83136095]
 [ 1.4556213 ]
 [ 3.72485207]
 [11.63905325]
 [17.71301775]]
predicted response:
[[ 8.63609467]
 [ 4.83136095]
 [ 1.4556213 ]
 [ 3.72485207]
 [11.63905325]
 [17.71301775]]
[[ True]
 [ True]
 [ True]
 [ True]
 [ True]
 [ True]]
```

```
X_new = np.array([2.4, 3.5, 4.8, 5.5, 7, 8.5, 10]).reshape((-1, 1))

# Do NOT forget to transform using PolynomialFeatures !!!
X_new_ = PolynomialFeatures(degree=2, include_bias = True).fit_transform(X_new)

y_new_pred = model.predict(X_new_)

plt.figure(figsize=(8, 6))

plt.xlim(0, 10)
plt.ylim(0, 20)
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(X, y, color = "blue", label = "Data Points")
plt.scatter(X, y_pred, color = "red", label = "Predicted Points")
plt.scatter(X_new, y_new_pred, color = "cyan", label = "New Predicted Points")

line_Xs = np.linspace(0, 10, 100).reshape(-1, 1)
# Do NOT forget to transform using PolynomialFeatures !!!
```

```
line_Xs_ = PolynomialFeatures(degree=2, include_bias = True).fit_transform(line_Xs)

line_ys = model.predict(line_Xs_)
plt.plot(line_Xs, line_ys)

coef0 = model.coef_.squeeze()[0]
coef1 = model.coef_.squeeze()[1]
coef2 = model.coef_.squeeze()[2]

title_text = "$y$ = (%0.3f) + (%0.3f) $X$ + (%0.3f) $X^2$\n$R^2$ = %0.3f" % (coef0, coef1, c
plt.title(title_text)

plt.legend()

plt.show()
```



Pros and cons of linear regression

Pros:-

1. Ease of use: Linear regression is generally considered to be a straightforward and manageable algorithm that can be used on many types of computational systems.
2. Simplicity and efficiency: The underlying linear regression technique is relatively simple to understand compared to other machine-learning techniques
3. Modeling linear relationships: Linear regression can effectively model datasets that are linearly separable, which makes it useful when determining relationships between variables.
4. Making informed insights: With linear regression, you can use your data to explore the relationships between different variables and make predictions based on different values. This helps to inform decision-making, such as optimizing a marketing strategy or allocating the right volume of resources for a project.
5. Easy to interpret: The coefficients of a linear regression model represent the change in the dependent variable for a one-unit change in the independent variable, making it simple to comprehend the relationship between the variables.
6. Robust to outliers: Linear regression is relatively robust to outliers meaning it is less affected by extreme values of the independent variable compared to other statistical methods.
7. Can handle both linear and nonlinear relationships: Linear regression can be used to model both linear and nonlinear relationships between variables. This is because the independent variable can be transformed before it is used in the model.
8. No need for feature scaling or transformation: Unlike some machine learning algorithms, linear regression does not require feature scaling or transformation. This can be a significant advantage, especially when dealing with large datasets.

Cons:-

1. Causation vs. correlation: Regression analysis only shows correlation, not causation. Just because two things seem to move together doesn't mean one is directly affecting the other. There might be other hidden factors at play, or it could be a coincidence. It’s always important to use other forms of research and critical thinking to back up your findings from regression analysis.
2. Risk of underfitting: Linear regression might lead to underfitting, which is when the machine learning model fails to represent the data accurately.
3. Restricted to linear relationships: When measuring the relationship between naturally occurring variables, the underlying shape may be nonlinear Since linear regression assumes a linear relationship between input and output variables, this type of analysis would fail to fit complex datasets accurately.

4. Sensitivity to outliers: Outliers, or extreme values, can significantly impact linear regression by pulling the line of best fit toward them. This may lead to models that don't represent the data well.
5. Assumes linearity: Linear regression assumes that the relationship between the independent variable and the dependent variable is linear. This assumption may not be valid for all data sets. In cases where the relationship is nonlinear, linear regression may not be a good choice.
6. Sensitive to multicollinearity: Linear regression is sensitive to multicollinearity. This occurs when there is a high correlation between the independent variables.
7. Multicollinearity can make it difficult to interpret the coefficients of the model and can lead to overfitting.
8. May not be suitable for highly complex relationships: Linear regression may not be suitable for modeling highly complex relationships between variables. For example, it may not be able to model relationships that include interactions between the independent variables.
9. Not suitable for classification tasks: Linear regression is a regression algorithm and is not suitable for classification tasks, which involve predicting a categorical variable rather than a continuous variable.

Application of Linear Regression

1. Politics: The relationship between state spending and public support.
2. Business: The relationship between revenue and employee pay.
3. Economics: Linear regression is the predominant empirical tool in economics. For example, it is used to predict consumer spending, fixed investment spending, inventory investment, purchases of a country's exports, spending on imports, the demand to hold liquid assets, labor demand, and labor supply.
4. Finance: The capital price asset model uses linear regression to analyze and quantify the systematic risks of an investment.
5. Environment: The relationship between carbon emissions and taxes.
6. Sociology: The relationship between professional pay and applicant qualifications.
7. Psychology: The relationship between culture and inclusive behaviors.
8. Health: The relationship between patient demographics and body weight.
9. Biology: Linear regression is used to model causal relationships between parameters in biological systems.

10. Education: The relationship between academic grades and geographic location.
11. Trend lines: A trend line represents the variation in quantitative data with the passage of time (like GDP, oil prices, etc.). These trends usually follow a linear relationship. Hence, linear