Importing Libraries and Dataset

Regression analysis is the most widely used method of prediction. Linear regression is used when the dataset has a linear correlation and as the name suggests, simple linear regression has one independent variable (predictor) and one dependent variable(response).

The simple linear regression equation is represented as y = a+bx where x is the explanatory variable, y is the dependent variable, b is coefficient and a is the intercept.

For regression analysis, first we have to import libraries.

In [2]:

```
#Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
%matplotlib inline

import plotly.express as px
import plotly.graph_objs as go
from plotly.offline import iplot
```

In [3]:

```
from google.colab import files


uploaded = files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Salary_Data.csv to Salary_Data.csv
```

In [5]:

```
# read the dataset using pandas
import pandas as pd
import io
dataset = pd.read_csv(io.BytesIO(uploaded['Salary_Data.csv']))
```

Data Analysis

In [6]:

```
#To see first 5 rows of the dataset
dataset.head().style.background_gradient(cmap='pink_r')
```

Out[6]:

	YearsExperience	Salary
0	1.100000	39343.000000
1	1.300000	46205.000000
2	1.500000	37731.000000
3	2.000000	43525.000000
4	2.200000	39891.000000

In [7]:

```
# To see information of dataset
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     float64
dtypes: float64(2)
memory usage: 608.0 bytes
```

In [8]:

```
#Statistical Analysis
dataset.describe().style.background_gradient(cmap='pink_r')
```

Out[8]:

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

Both are numerical column

Visualization

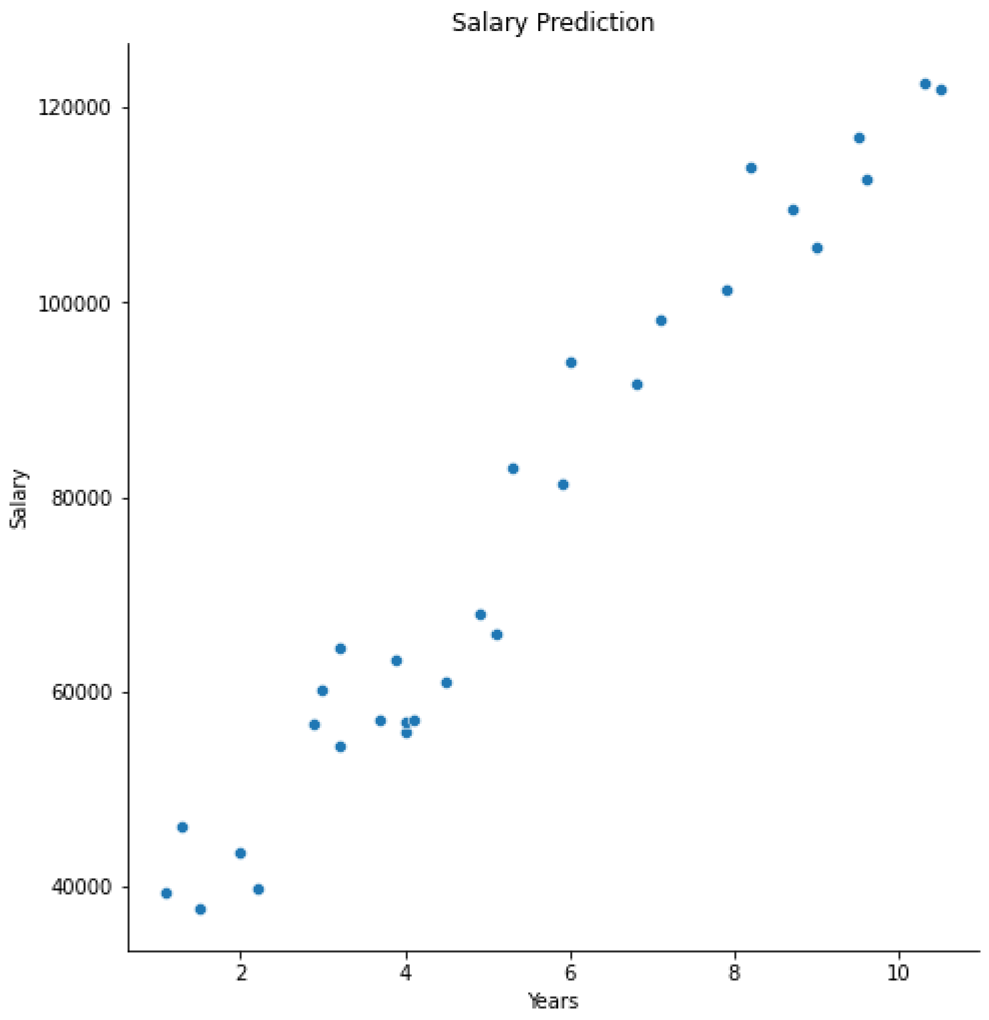
Scatter Plot

In [9]:

```
# These Plots help to explain the values and how they are scattered

plt.figure(figsize=(12,6))
sns.pairplot(dataset,x_vars=['YearsExperience'],y_vars=['Salary'],height=7,kind=
'scatter')
plt.xlabel('Years')
plt.ylabel('Salary')
plt.title('Salary Prediction')
plt.show()
```

```
<Figure size 864x432 with 0 Axes>
```



Heat Map

In [10]:

```
fig = px.imshow(dataset.corr())
fig.show()
```

In [14]:

```
# Cooking the dataset
X = dataset['YearsExperience']
X.head()
```

Out[14]:

```
0    1.1
1    1.3
2    1.5
3    2.0
4    2.2
Name: YearsExperience, dtype: float64
```

In [15]:

```
# Cooking the dataset
y = dataset['Salary']
y.head()
```

Out[15]:

```
0    39343.0
1    46205.0
2    37731.0
3    43525.0
4    39891.0
Name: Salary, dtype: float64
```

In [16]:

```
# Import Segregating dataset from scikit learn
from sklearn.model_selection import train_test_split
```

In [17]:

```
# Split the dataset for train and test
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=0.7,random_state
=100)
```

In [18]:

```
# Create new axis for x column
X_train = X_train[:,np.newaxis]
X_test = X_test[:,np.newaxis]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: Futu
reWarning:

Support for multi-dimensional indexing (e.g. `obj[:, None]`) is depr
ecated and will be removed in a future version.  Convert to a numpy
array before indexing instead.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: Futu
reWarning:

Support for multi-dimensional indexing (e.g. `obj[:, None]`) is depr
ecated and will be removed in a future version.  Convert to a numpy
array before indexing instead.
```

In [19]:

```
# Importing Linear Regression model from scikit learn
from sklearn.linear_model import LinearRegression
```

In [20]:

```
# Fitting the model
lr = LinearRegression()
lr.fit(X_train,y_train)
```

Out[20]:

```
LinearRegression()
```

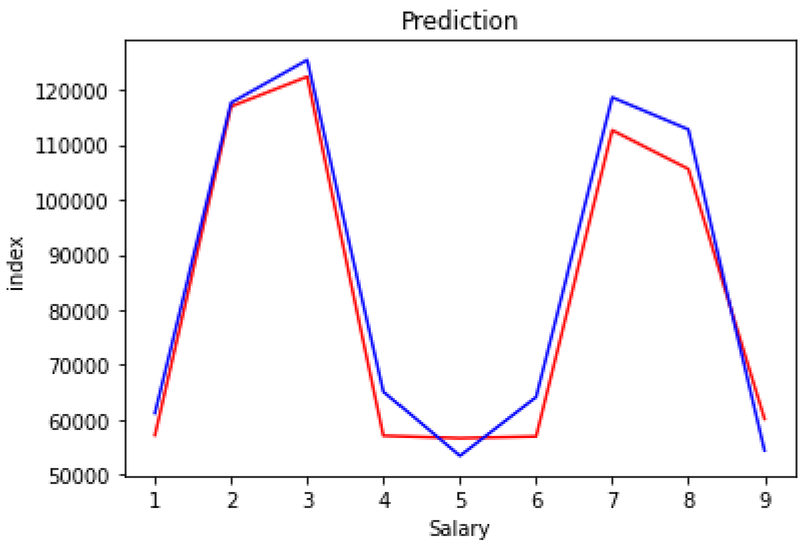
In [21]:

```
# Predicting the Salary for the Test values
y_pred = lr.predict(X_test)
```

In [22]:

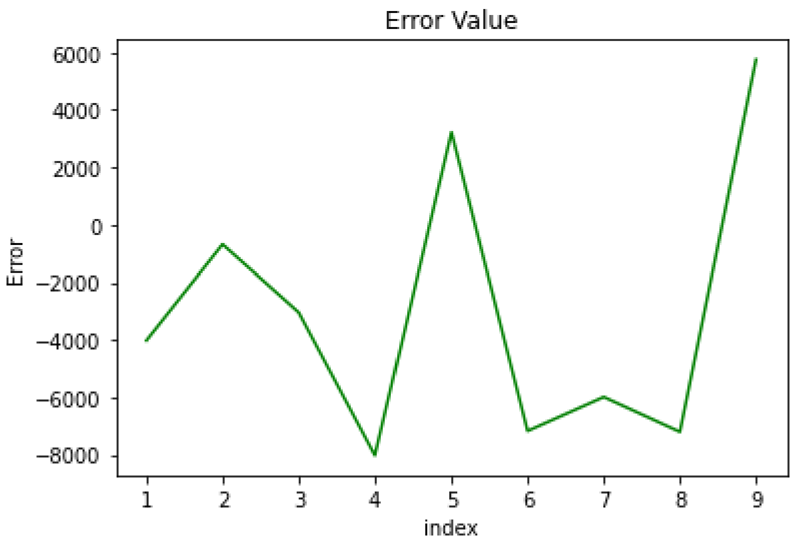
```
# Plotting the actual and predicted values

c = [i for i in range (1,len(y_test)+1,1)]
plt.plot(c,y_test,color='r',linestyle='-')
plt.plot(c,y_pred,color='b',linestyle='-')
plt.xlabel('Salary')
plt.ylabel('index')
plt.title('Prediction')
plt.show()
```



In [23]:

```
# plotting the error
c = [i for i in range(1,len(y_test)+1,1)]
plt.plot(c,y_test-y_pred,color='green',linestyle='-')
plt.xlabel('index')
plt.ylabel('Error')
plt.title('Error Value')
plt.show()
```



In [24]:

```
# Importing metrics for the evaluation of the model
from sklearn.metrics import r2_score,mean_squared_error
```

In [25]:

```
# calculate Mean square error
mse = mean_squared_error(y_test,y_pred)
```

In [26]:

```
# Calculate R square vale
rsq = r2_score(y_test,y_pred)
```

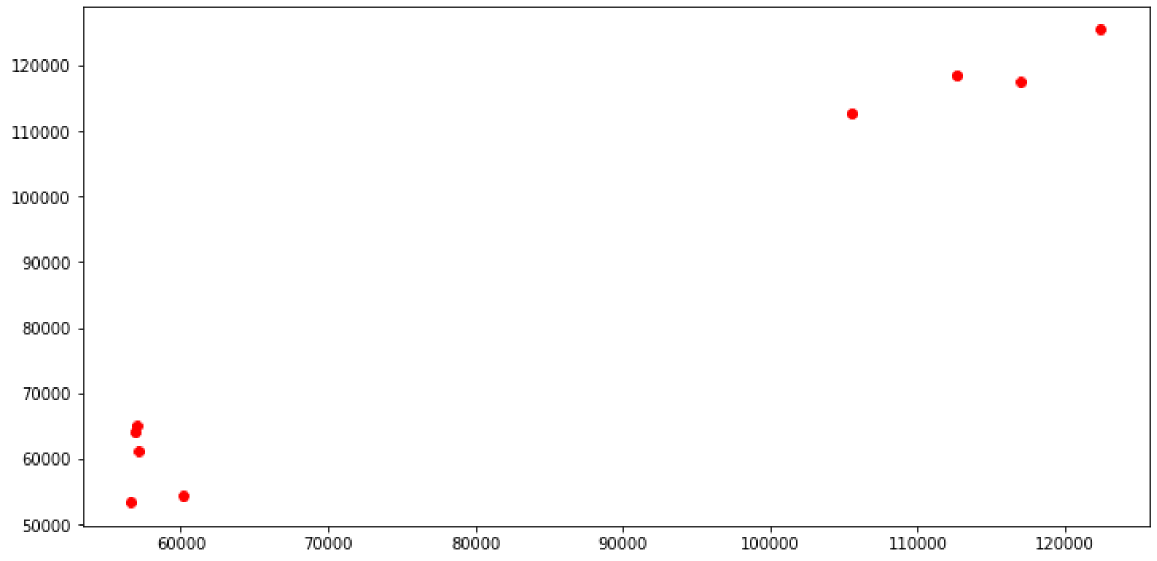
In [27]:

```
print('mean squared error :',mse)
print('r square :',rsq)
```

```
mean squared error : 30310299.043402452
r square : 0.9627668685473267
```

In [28]:

```
# Just plot actual and predicted values for more insights
plt.figure(figsize=(12,6))
plt.scatter(y_test,y_pred,color='r',linestyle='-')
plt.show()
```



In [29]:

```
# Intecept and coeff of the line
print('Intercept of the model:',lr.intercept_)
print('Coefficient of the line:',lr.coef_)
```

```
Intercept of the model: 25202.887786154883
Coefficient of the line: [9731.20383825]
```

Then it is said to form a line with

y = 25202.8 + 9731.2x