EXPERIMENT 2

AIM: Simple Linear Regression in Python/R.

THEORY :

- Linear Regression is a machine learning algorithm based on supervised learning.
- Linear regression is used for finding linear relationship between target and one or more predictors.
- Linear regression is one of the easiest and most popular Machine Learning algorithms.
- Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (x) variables, hence called as linear regression.
- Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.
- It is a statistical method that is used for predictive analysis.
- Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.
- There are two types of linear regression- Simple Regression and Multiple Regression.

❖ SIMPLE LINEAR REGRESSION:

- Simple linear regression is a statistical method for establishing the relationship between two variables using a straight line.
- The line is drawn by finding the slope and intercept, which define the line and minimize regression errors.
- The simplest form of simple linear regression has only one x variable and one y variable.
- The x variable is the independent variable because it is independent of what you try to predict the dependent variable.
- The y variable is the dependent variable because it depends on what you try to predict.
- **y = B0 +B1x+e** is the formula used for simple linear regression.
- y is the predicted value of the dependent variable (y) for any given value of the independent variable (x).

- B0 is the intercept, the predicted value of y when the x is 0.
- B1 is the regression coefficient – how much we expect y to change as x increases.
- x is the independent variable (the variable we expect is influencing y).
- e is the error of the estimate, or how much variation there is in our regression coefficient estimate.

- Simple linear regression establishes a line that fits your data, but it does not guarantee that the line is good enough.
- For example, if your data points have an upward trend and are very far apart, then simple linear regression will give you a downward-sloping line, which will not match your data.
- A simple linear regression can accurately capture the relationship between two variables in simple relationships.
- But when dealing with more complex interactions that require more thought, you need to switch from simple to multiple regression.

CODE OUTPUT (PYTHON)

Dataset used:- Student_Score

```
[1]  #Import Libraries
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import math
     from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_squared_error, r2_score
```

```
df = pd.read_csv("student_scores.csv")
df.head()
```

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

```
[4]  x = df["Hours"]
     y = df["Scores"]
```

```
[5]  df.describe()
```

colab.research.google.com/drive/1CO-dYmbqdBJ7ni259jK5L3731zg1c3bD#scrollTo=OibDCzQdMsie

+ Code + Text

min	1.100000	17.000000
25%	2.700000	30.000000
50%	4.800000	47.000000
75%	7.400000	75.000000
max	9.200000	95.000000

```
df.dtypes
```

```
Hours      float64
Scores       int64
dtype: object
```

```
[7] df.info()
```

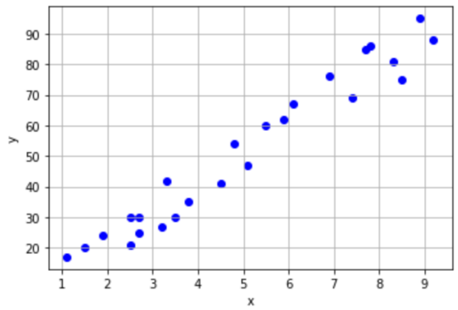
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Hours   25 non-null     float64
 1   Scores  25 non-null     int64
dtypes: float64(1), int64(1)
memory usage: 528.0 bytes
```

```
[8] plt.scatter(x, y, color = 'Blue')
    #plt.plot(x, y_predicted, color = 'green')
    plt.xlabel('x')
```

colab.research.google.com/drive/1CO-dYmbqdBJ7ni259jK5L3731zg1c3bD#scrollTo=OibDCzQdMsie

+ Code + Text

```
[8] plt.scatter(x, y, color = 'Blue')
    #plt.plot(x, y_predicted, color = 'green')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid()
    plt.show()
```




```
[9] df=df.drop_duplicates()
```

```
[10] x.head()
```

```
0    2.5
```

+ Code + Text

[10]
```
x.head()
```

```
0    2.5
1    5.1
2    3.2
3    8.5
4    3.5
Name: Hours, dtype: float64
```

[11]
```
x = np.array(x)
y = np.array(y)
y
```

```
array([21, 47, 27, 75, 30, 20, 88, 60, 81, 25, 85, 62, 41, 42, 17, 95, 30,
       24, 67, 69, 30, 54, 35, 76, 86])
```

[12]
```
x = x.reshape(-1,1)
y = y.reshape(-1,1)
```

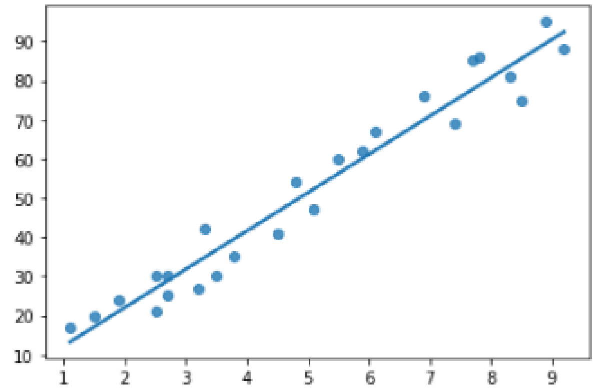
[13]
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

print(len(x_test))
```

✓ 0s completed at 3:34 PM

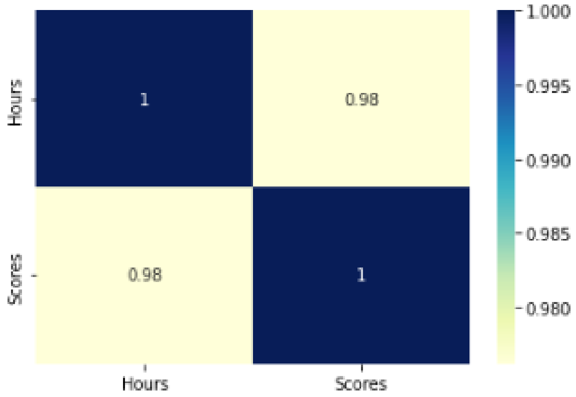
+ Code + Text

[22]
```
import seaborn as sns
sns.regplot(x, y, ci=None)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWa
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7f71fcbac100>
```



[24]
```
sns.heatmap(df.corr(),cmap="YlGnBu", annot=True)
plt.show()
```



[25]
```
import statsmodels.api as sm
```

[26]
```
x_train_sm=sm.add_constant(x_train)
lr=sm.OLS(y_train,x_train_sm).fit()
```

```
lr.summary()
```

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.947
Model:	OLS	**Adj. R-squared:**	0.944
Method:	Least Squares	**F-statistic:**	322.4

```
[27] lr.summary()
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.947
Model:	OLS	Adj. R-squared:	0.944
Method:	Least Squares	F-statistic:	322.4
Date:	Tue, 14 Feb 2023	Prob (F-statistic):	6.14e-13
Time:	09:25:52	Log-Likelihood:	-62.677
No. Observations:	20	AIC:	129.4
Df Residuals:	18	BIC:	131.3
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.9738	3.195	0.618	0.544	-4.738	8.686
x1	9.7897	0.545	17.957	0.000	8.644	10.935

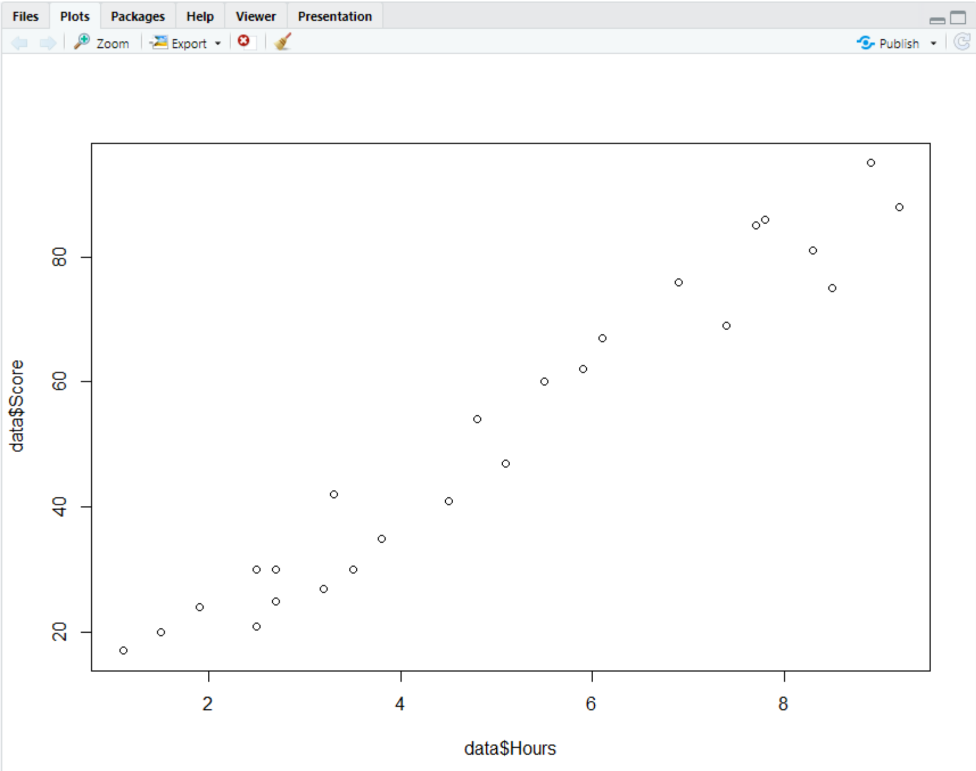
Omnibus:	7.470	Durbin-Watson:	2.620
Prob(Omnibus):	0.024	Jarque-Bera (JB):	1.836
Skew:	-0.090	Prob(JB):	0.399
Kurtosis:	1.527	Cond. No.	14.6

Code and Output: R

data = read.csv("C:/Users/saksh/Downloads/student_scores.csv")
library(dplyr)
# random_Inspection of Dataset
sample_n(data,5)

```
> data = read.csv("C:/Users/saksh/Downloads/student_scores.csv")
> library(dplyr)

  Hours Scores
1   4.5     41
2   4.8     54
3   8.5     75
4   7.8     86
5   6.9     76
> plot(data$Hours,data$Score)
>
```

# plotting data to understand which regression method should be applied
plot(data$Hours,data$Score)



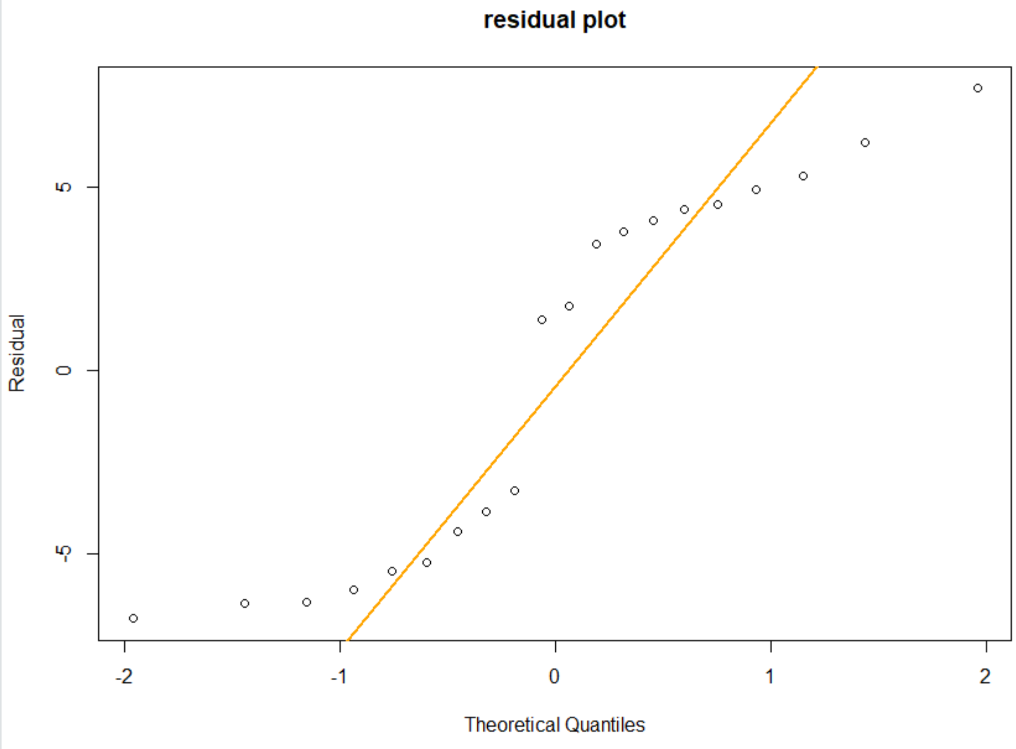
# Pearson's correlation test will also help to check linearity and correlation between data
cor.test(data$Hours,data$Score)

#Since, p-value is less than 0.05 and sample correlation is 0.9782416.
#Hence, High significant correlation (Highly Linearly Related) exists in the population.
library(caret)

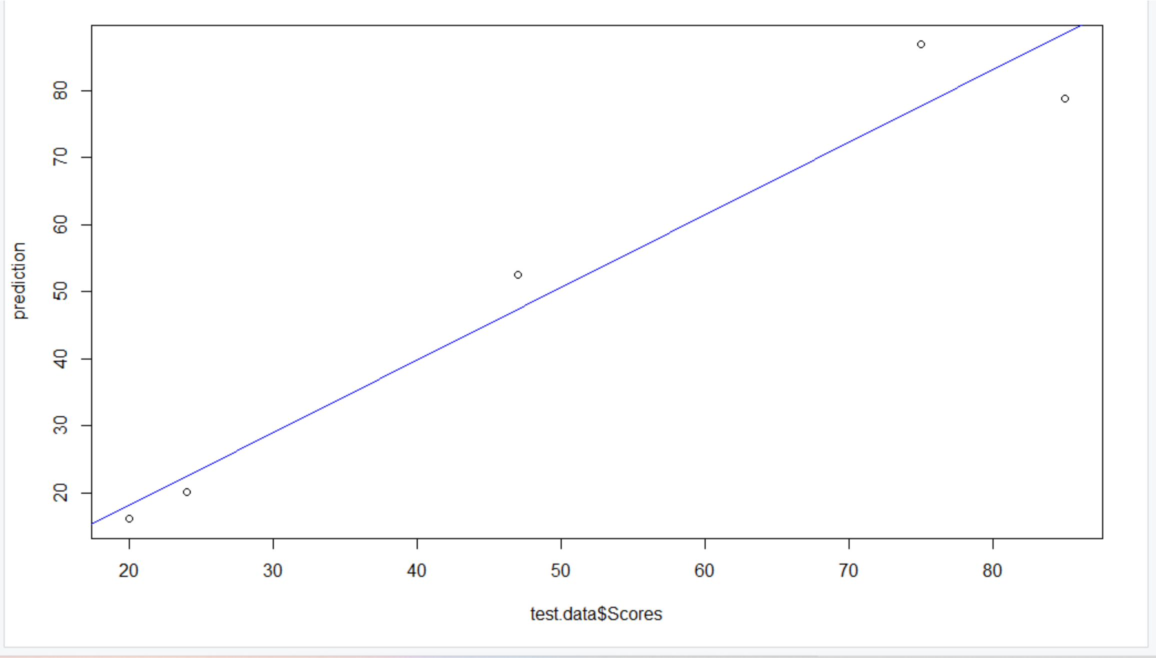
```
> sample_n(data,5)
  Hours Scores
1   4.5     41
2   4.8     54
3   8.5     75
4   7.8     86
5   6.9     76
> plot(data$Hours,data$Score)
> cor.test(data$Hours,data$Score)

        Pearson's product-moment correlation

data:  data$Hours and data$Score
t = 21.583, df = 23, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9459248 0.9896072
sample estimates:
      cor
0.9761907
```

#Split the given dataset into train and test data,
#fit the model and obtain summary of model as follows

```
training.samples <- data$Score %>%
  createDataPartition(p = 0.7, list = FALSE)
train.data <- data[training.samples, ]
test.data <- data[-training.samples, ]

# Fit model
model <- lm(Scores ~ Hours, data = train.data)
summary(model)

# Visualization
qqnorm(model$residuals,ylab="Residual", main="residual plot")
qqline(model$residuals, col = "orange", lwd = 2)
```




```
# Making prediction

prediction <- model %>% predict(test.data)
# Visualization
plot(test.data$Scores , prediction)
abline(lm(prediction ~ Scores, data = test.data), col = "blue")
```




```
# Statistical Measure
data.frame( R2 = R2(prediction, test.data$Scores),
            RMSE = RMSE(prediction, test.data$Scores),
            MAE = MAE(prediction, test.data$Scores))
# Multifold training
# Define training control
train.control <- trainControl(method = "repeatedcv",
                              number = 4, repeats = 3)
# Train the model
model_cv <- train(Scores ~ Hours , data = data, method="lm",
                  trControl = train.control)
# Summarize the results
print(model_cv)
```

```
> training.samples <- data$Score %>%
+   createDataPartition(p = 0.7, list = FALSE)
> train.data <- data[training.samples, ]
> test.data <- data[-training.samples, ]
> model <- lm(Scores ~ Hours, data = train.data)
> summary(model)

Call:
lm(formula = Scores ~ Hours, data = train.data)

Residuals:
   Min     1Q Median     3Q    Max 
-6.755 -5.282  1.579  4.420  7.686 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   0.9591     2.7716   0.346    0.733    
Hours        10.1075     0.4988  20.266 7.67e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 5.269 on 18 degrees of freedom
Multiple R-squared:  0.958,	Adjusted R-squared:  0.9557 
F-statistic: 410.7 on 1 and 18 DF,  p-value: 7.67e-14

> qqnorm(model$residuals,ylab="Residual", main="residual plot")
> qqline(model$residuals, col = "orange", lwd = 2)
> prediction <- model %>% predict(test.data)
> plot(test.data$Scores , prediction)
> abline(lm(prediction ~ Scores, data = test.data), col = "blue")
> # Statistical Measure
> data.frame( R2 = R2(prediction, test.data$Scores),
+             RMSE = RMSE(prediction, test.data$Scores),
+             MAE = MAE(prediction, test.data$Scores))
         R2     RMSE      MAE
1 0.9492178 6.923496 6.261933
> # Multifold training
> # Define training control
> train.control <- trainControl(method = "repeatedcv",
+                               number = 4, repeats = 3)
> # Train the model
> model_cv <- train(Scores ~ Hours , data = data, method="lm",
+                   trControl = train.control)
> # Summarize the results
> print(model_cv)
Linear Regression 

25 samples
 1 predictor

No pre-processing
Resampling: Cross-Validated (4 fold, repeated 3 times) 
Summary of sample sizes: 18, 20, 20, 17, 19, 19, ... 
```

```
+                   trControl = train.control)
> # Summarize the results
> print(model_cv)
Linear Regression 

25 samples
 1 predictor

No pre-processing
Resampling: Cross-Validated (4 fold, repeated 3 times) 
Summary of sample sizes: 18, 20, 20, 17, 19, 19, ... 
Resampling results:

  RMSE      Rsquared   MAE     
  5.845614  0.9641844  5.425861

Tuning parameter 'intercept' was held constant at a value of TRUE
> 
```

Environment | History | Connections | Tutorial

Import Dataset | 388 MiB

R | Global Environment

Data	
data	25 obs. of 2 variables
model	List of 12
model_cv	List of 24
test.data	5 obs. of 2 variables
train.control	List of 27
train.data	20 obs. of 2 variables
training.samples	int [1:20, 1] 1 3 5 7 8 9 10 12 13 14 ...
Values	
prediction	Named num [1:5] 52.5 86.9 16.1 78.8 20.2

Conclusion:

regression analysis is a supervised learning algorithm that uses labeled data to produce continuous variables. The linear regression model comprises a single parameter and a linear connection between the dependent and independent variables. we have studied and implemented the basic concepts of Simple Linear Regression of both Python & R, using students-score dataset.