Name : Salandri Nirusha (Data Science & Business Analytics Intern)

Prediction using Supervised ML

simple linear regression task

> Simple linear regression is a regression model that estimates the relationship between one independent variable and one dependent variable using a straight line. Both variables should be quantitative. In this regression task we will predict the percentage of marks that a student is expected to score based upon the number of hours they studied. This is a simple linear regression task as it involves just two variables.

Importing required libraries

In [56]:

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

Reading data from remote link

In [57]:

```
url = "http://bit.ly/w-data"
data = pd.read_csv(url)
print("Data imported successfully")
```

```
Data imported successfully
```

Exploratory data analysis(EDA)

In [58]:

```
data.head(5)
```

Out[58]:

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

In [59]:

```
data.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
```

In [60]:

```
data.describe(include='all')
```

Out[60]:

	Hours	Scores
count	25.000000	25.000000
mean	5.012000	51.480000
std	2.525094	25.286887
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

In [61]:

```
data.columns
```

Out[61]:

```
Index(['Hours', 'Scores'], dtype='object')
```

In [62]:

```
data.shape
```

Out[62]:

```
(25, 2)
```

checking for null_values

In [63]:

```
data.isnull().sum()
```

Out[63]:

```
Hours     0
Scores    0
dtype: int64
```

In [64]:

```
data.isnull().values.any()
```

Out[64]:

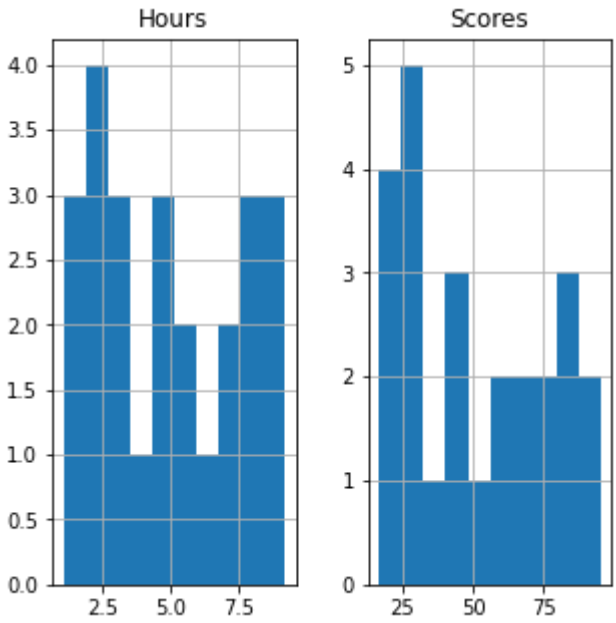
```
False
```

In [65]:

```
data.hist(figsize=(5,5))
```

Out[65]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000011FB66A9DC8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011FB66FE608
>]],
      dtype=object)
```



checking for Outlier's

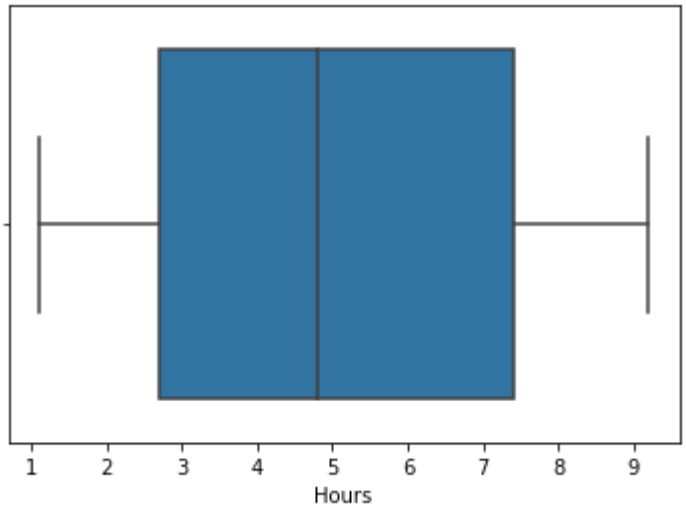
> An outlier is an observation that lies an abnormal distance from other values in a random sample from a population.

In [66]:

```
sns.boxplot(x="Hours",  data=data)
```

Out[66]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x11fb675e788>
```

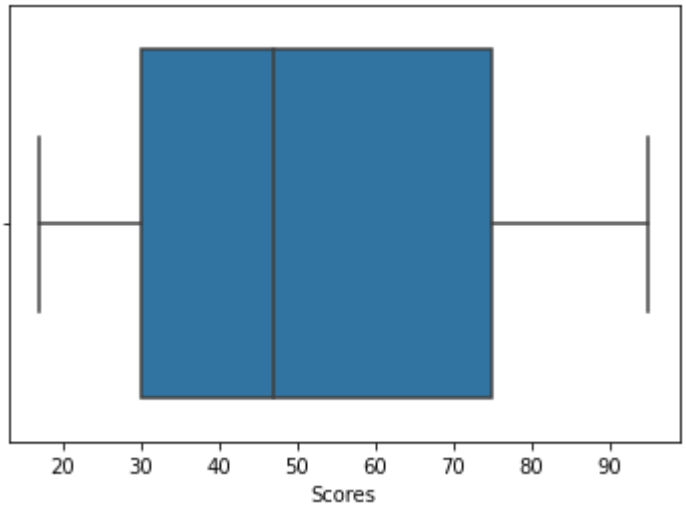


In [67]:

```
sns.boxplot(x="Scores",  data=data)
```

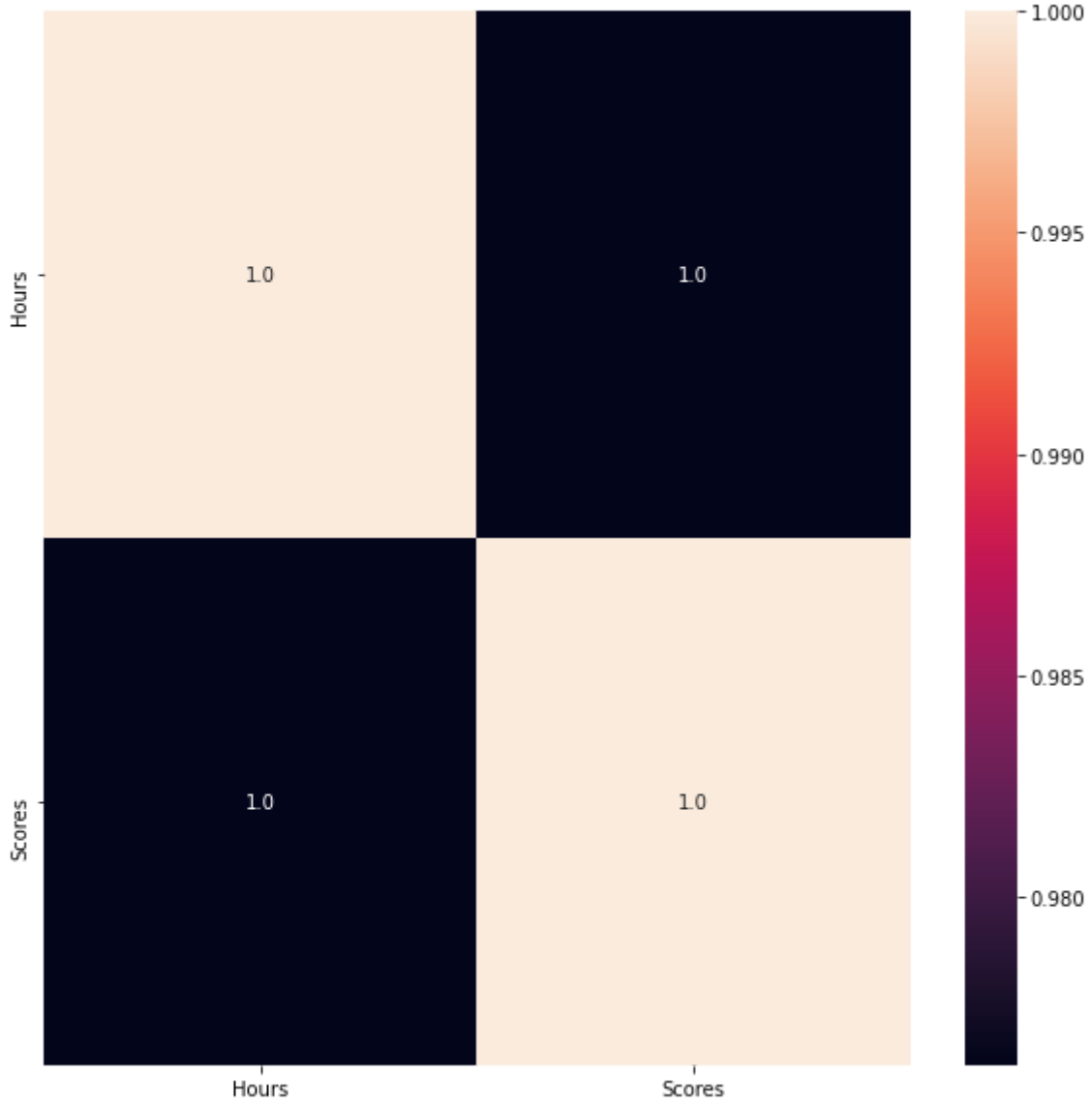
Out[67]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x11fb6680248>
```



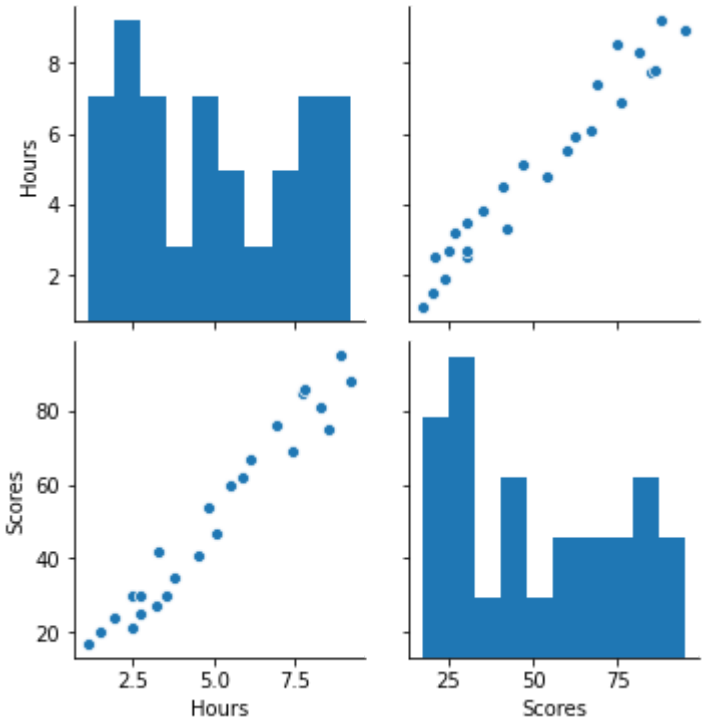
In [68]:

```
plt.figure(figsize=(10,10))
sns.heatmap(data.corr(),annot=True,fmt='.1f')
plt.show()
```



In [70]:

```
sns.pairplot(data)
plt.show()
```

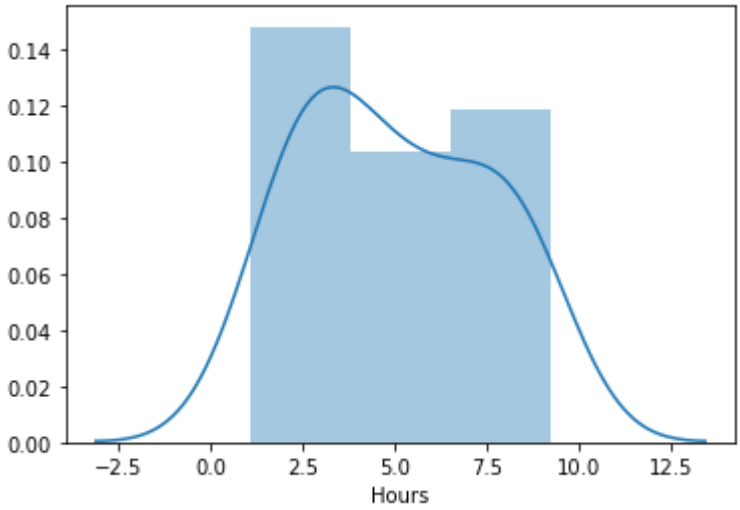


In [71]:

```
sns.distplot(data['Hours'])
```

Out[71]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x11fb6c74e48>
```

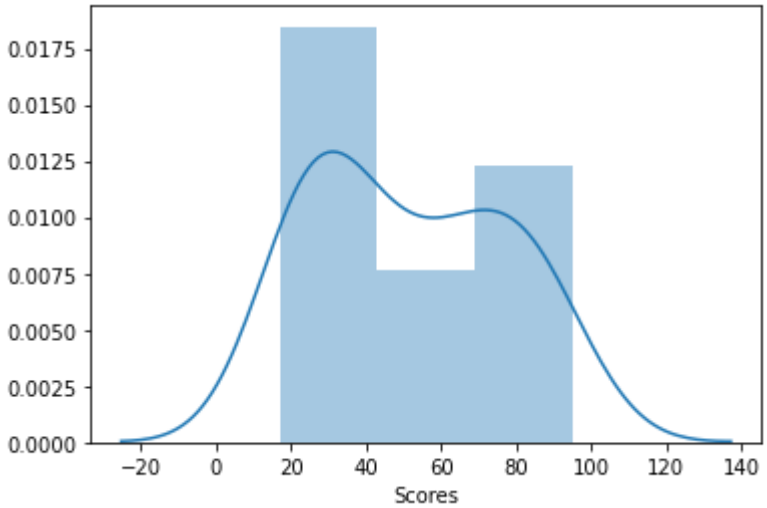


In [72]:

```
sns.distplot(data['Scores'])
```

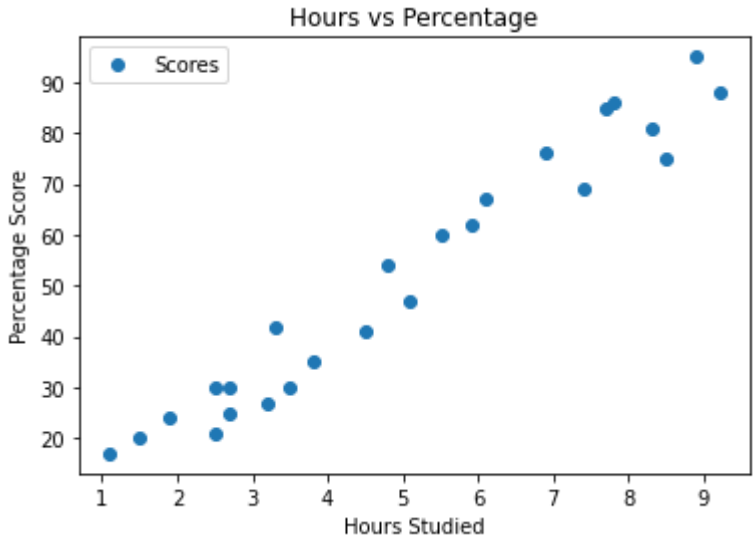
Out[72]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x11fb6cd01c8>
```



In [82]:

```
# Plotting the distribution of scores
data.plot(x='Hours', y='Scores', style='o')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()
```



Defining Independent & Dependent variable

In [74]:

```
X = data.iloc[:, :-1].values
y = data.iloc[:, 1].values
```

In [38]:

```
X
```

Out[38]:

```
array([[2.5],
       [5.1],
       [3.2],
       [8.5],
       [3.5],
       [1.5],
       [9.2],
       [5.5],
       [8.3],
       [2.7],
       [7.7],
       [5.9],
       [4.5],
       [3.3],
       [1.1],
       [8.9],
       [2.5],
       [1.9],
       [6.1],
       [7.4],
       [2.7],
       [4.8],
       [3.8],
       [6.9],
       [7.8]])
```

Splitting the data into train & test with 80% & 20% respectively

In [39]:

```
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, random_state=0)
```

Building and training the model.

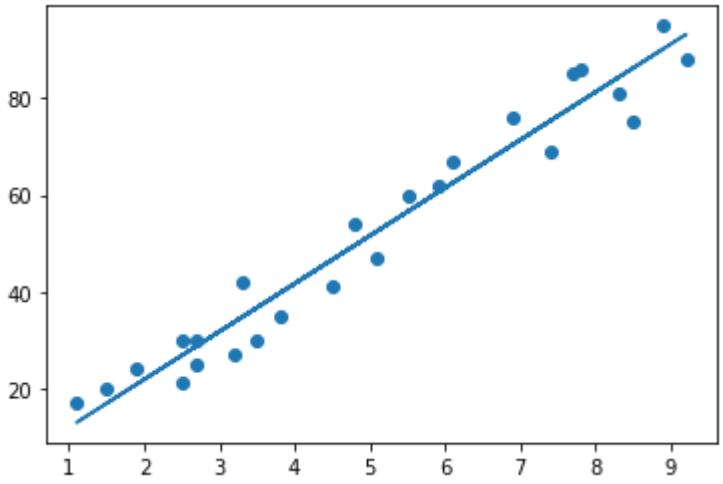
In [40]:

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

print("Training complete.")
```

```
Training complete.
```

In [41]:

```
#Plotting the regression line
line = regressor.coef_*X+regressor.intercept_

# Plotting for the test data
plt.scatter(X, y)
plt.plot(X, line);
plt.show()
```



In [42]:

```
print(X_test) # Testing data - In Hours
y_pred = regressor.predict(X_test) # Predicting the scores
```

```
[[1.5]
 [3.2]
 [7.4]
 [2.5]
 [5.9]]
```

Comparing Actual vs Predicted

In [52]:

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

Out[52]:

	Actual	Predicted
0	20	16.884145
1	27	33.732261
2	69	75.357018
3	30	26.794801
4	62	60.491033

In [46]:

```
regressor.coef_
```

Out[46]:

```
array([9.91065648])
```

In [48]:

```
regressor.intercept_
```

Out[48]:

```
2.018160041434683
```

Linear Regression Equation

> Scores=2.01+9.91*Hours

Example

In [80]:

```
hours=3.2
score=2.01+9.91*(hours)
print(score)
```

```
33.722
```

Evaluating the model

> Mean Absolute Error (MAE) is another loss function used for regression models. MAE is the sum of absolute differences between our target and predicted variables.

In [81]:

```
from sklearn import metrics
print('Mean Absolute Error:',
      metrics.mean_absolute_error(y_test, y_pred))
```

```
Mean Absolute Error: 4.183859899002975
```

Thank you!