The Sparks Foundation

Task 2 - Supervised Machine Learning - Simple Linear Regression

Dataset: consist of 2 variables i.e. Number of Hours and Scores.

In [2]:
```
# importing libraries
import pandas as pd
import numpy as np
```

In [3]:
```
# Reading data from remote link
url = "http://bit.ly/w-data"
data = pd.read_csv(url)
print("Data imported successfully")
data.head(10)
```

```
Data imported successfully
```

Out[3]:

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30
5	1.5	20
6	9.2	88
7	5.5	60
8	8.3	81
9	2.7	25

In [4]:
```
# Summary of the dataframe.
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 [5]:
```
# Computing the Statistics Summary of the dataset
data.describe()
```

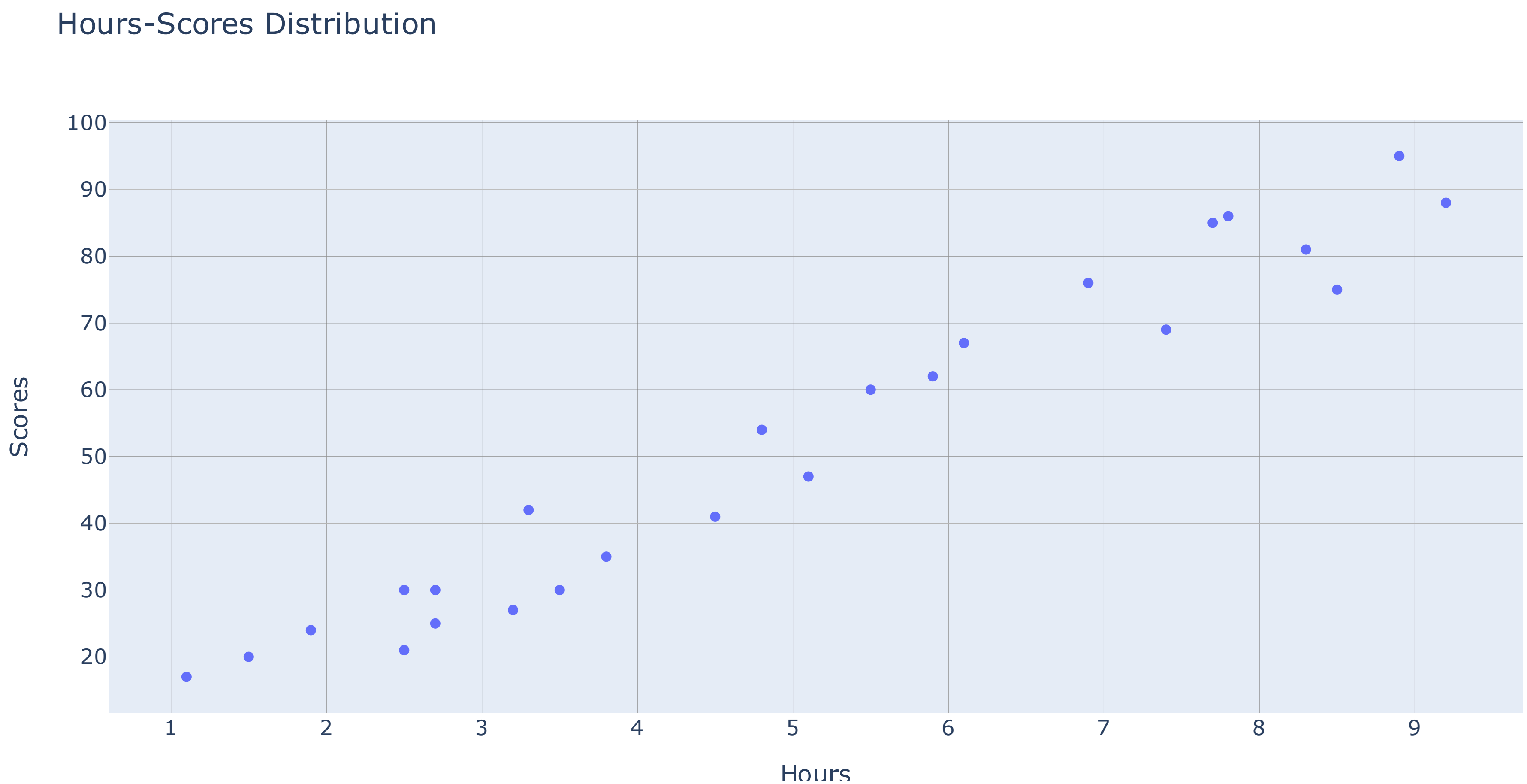
Out[5]:

	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

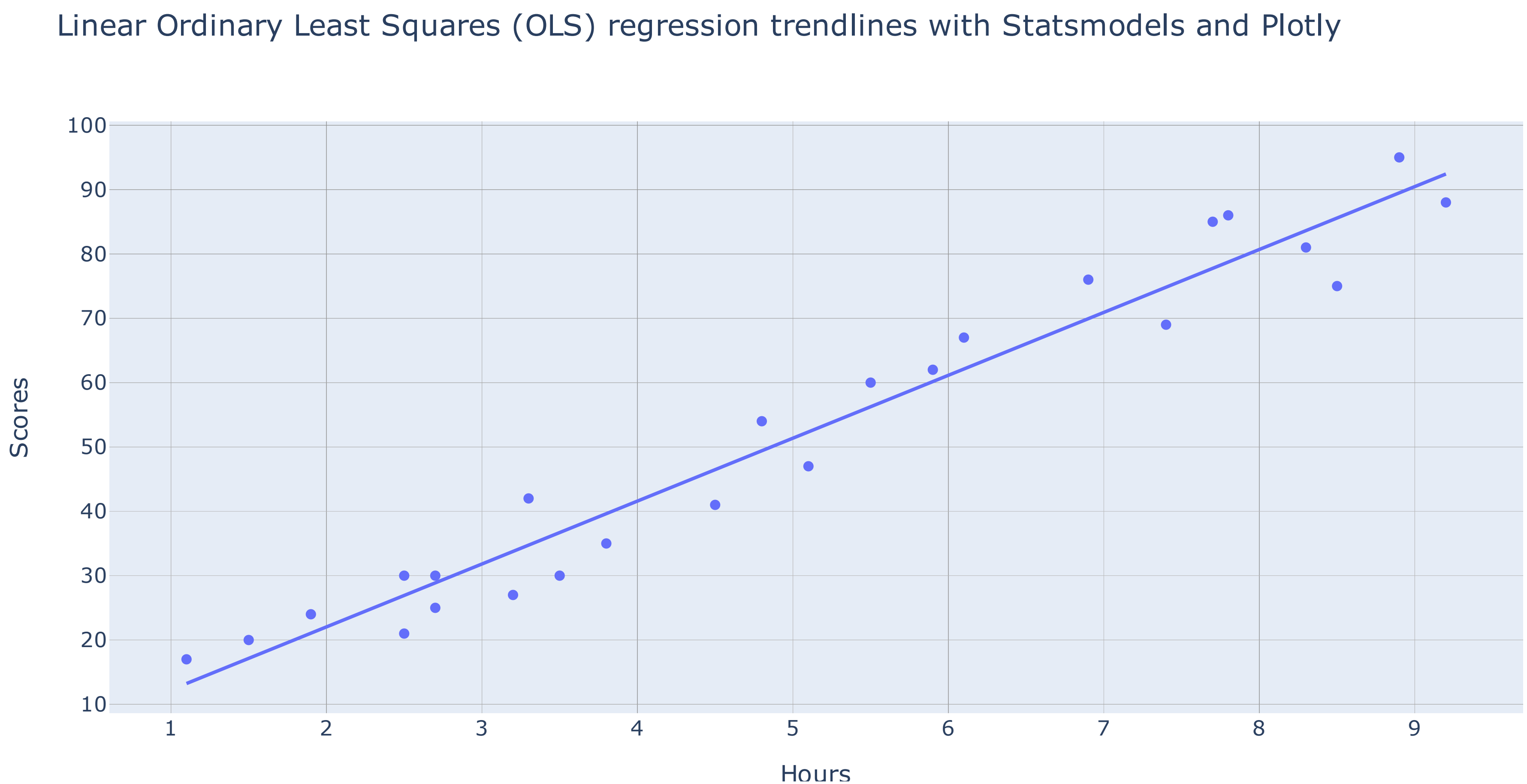
In [6]:
```
# Finding the missing values
data.isnull().sum()
```

Out[6]:
```
Hours     0
Scores    0
dtype: int64
```

In [7]:
```
#Hour-Scores Distribution using Plotly.
import plotly.express as px
fig = px.scatter(data, x="Hours", y="Scores", title = "Hours-Scores Distribution")
fig.show()
```



In [9]:
```python
# Ordinary Least Squares regression trendline
import plotly.express as px
fig = px.scatter(data, x="Hours", y="Scores", trendline="ols",
                 title="Linear Ordinary Least Squares (OLS) regression trendlines with Statsmodels and Plotly")
fig.show()
```



From the graph above, we can clearly see that there is a positive linear relation between the number of hours studied and percentage of score.

Preparing the data

In [10]:
```
X = data.iloc[:, :-1].values
y = data.iloc[:, 1].values
```

Splitting the dataset into the Training set and Test set

For the machine learning models, models need to learn something.

So we have to make two different sets, training set on which we build the machine model. And the test set on which we test the performance of the model.

In [11]:
```
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)
```

In [12]:
```
print('The number of observations in train set:', len(X_train))
print('The number of observations in test set:', len(X_test))
```

```
The number of observations in train set: 20
The number of observations in test set: 5
```

From above statement, the observations in train set and test set divided with 20 and 5.

Fitting the Simple Linear Regression to the Training set

In [13]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

print("Training complete.")
```

```
Training complete.
```

Making Predictions

In [14]:
```
print(X_test) # Testing data - In Hours
y_pred = regressor.predict(X_test) # Predicting the scores
y_pred
```

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

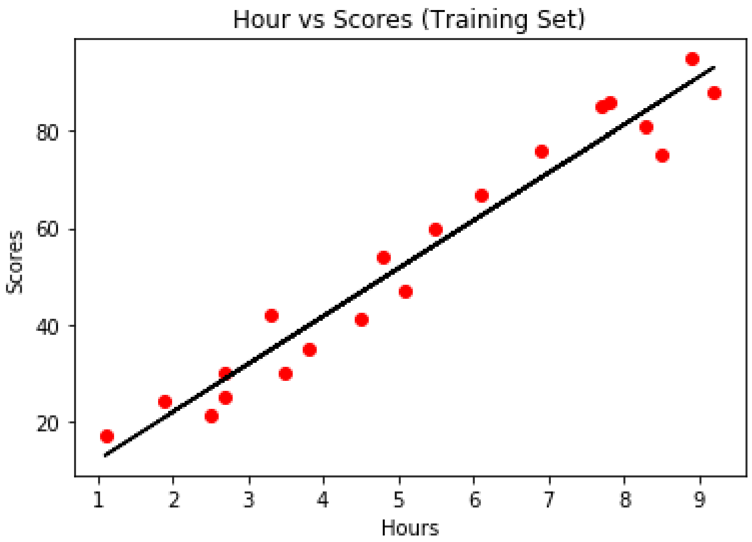
Out[14]:
```
array([16.88414476, 33.73226078, 75.357018  , 26.79480124, 60.49103328])
```

In [15]:
```
# Comparing Actual vs Predicted
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

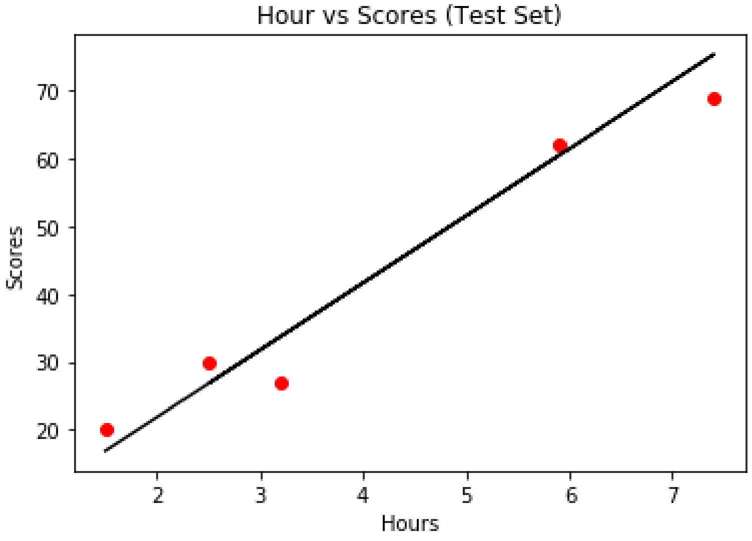
Out[15]:

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

In [16]:
```python
#Visualize the result with Matplotlib
#Training Set
import matplotlib.pyplot as plt
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'black')
plt.title("Hour vs Scores (Training Set)")
plt.xlabel("Hours")
plt.ylabel("Scores")
plt.show()
```



In [17]:
```
#Visualize the result with Matplotlib
#Test Set
import matplotlib.pyplot as plt
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_test, regressor.predict(X_test), color = 'black')
plt.title("Hour vs Scores (Test Set)")
plt.xlabel("Hours")
plt.ylabel("Scores")
plt.show()
```



In [18]:
```
# You can also test with your own data
hours = 9.25
own_pred = regressor.predict([[hours]])
print("No of Hours = {}".format(hours))
print("Predicted Score = {}".format(own_pred[0]))
```

```
No of Hours = 9.25
Predicted Score = 93.69173248737538
```

Evaluating the model

The final step is to evaluate the performance of algorithm. This step is particularly important to compare how well different algorithms perform on a particular dataset. For simplicity here, we have chosen the mean square error. There are many such metrics.

In [19]:

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

```
Mean Absolute Error: 4.183859899002975
```

In [ ]:

```
```