In [1]:
```
""" student scores analysis"""
# Step-1 Importing all libraries required
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
#  Step-2 Load dataset into dataframe
student_score_df = pd.read_csv("students_scores.csv")
print('First 10 entries: \n', student_score_df.head(10))
```

```
First 10 entries:
     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 [3]:
```
print('Finding missing value: \n', student_score_df.isnull().sum())
```

```
Finding missing value:
 Hours     0
Scores    0
dtype: int64
```

In [4]:
```
print('Information of database: \n', student_score_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
Information of database: 
 None
```

In [5]:
```
student_score_df.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]:
```
# Step-3 Creating feature variable(X) and outcome variable(y) for building mod
el
data = student_score_df[["Hours", "Scores"]]
predict = "Scores"
X1 = np.array(data.drop([predict], 1))
X = sm.add_constant(X1)
Y = np.array(data[predict])
print('Values of X: \n', X)
print('Values of Y: \n', Y)
```

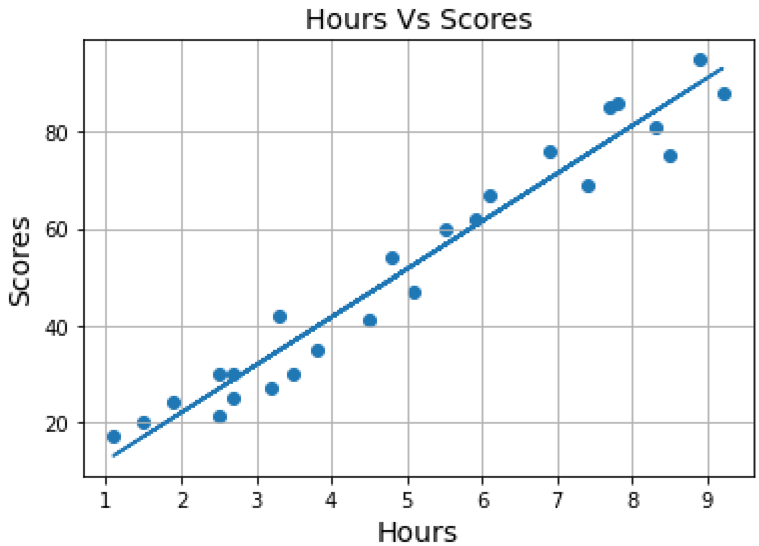
```
Values of X:
 [[1.  2.5]
 [1.  5.1]
 [1.  3.2]
 [1.  8.5]
 [1.  3.5]
 [1.  1.5]
 [1.  9.2]
 [1.  5.5]
 [1.  8.3]
 [1.  2.7]
 [1.  7.7]
 [1.  5.9]
 [1.  4.5]
 [1.  3.3]
 [1.  1.1]
 [1.  8.9]
 [1.  2.5]
 [1.  1.9]
 [1.  6.1]
 [1.  7.4]
 [1.  2.7]
 [1.  4.8]
 [1.  3.8]
 [1.  6.9]
 [1.  7.8]]
Values of Y:
 [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]
```

In [7]:
```
# Step-4 Splitting dataset into Training and Validation Sets
train_X, test_X, train_y, test_y = train_test_split(X, Y, train_size=0.8, rand
om_state=0)
```

In [8]:
```
# Step-5 Fitting of regression model by OLS method
stu_scores_lm = sm.OLS(train_y, train_X).fit()
print('Parameters: ', stu_scores_lm.params)
line = stu_scores_lm.params[1]*X1+stu_scores_lm.params[0]
```

```
Parameters:  [2.01816004 9.91065648]
```

In [9]:
```
# Plotting for the test data
plt.scatter(X1, Y)
plt.plot(X1, line)
plt.title('Hours Vs Scores', fontsize=14)
plt.xlabel('Hours', fontsize=14)
plt.ylabel('Scores', fontsize=14)
plt.grid(True)
plt.show()
```



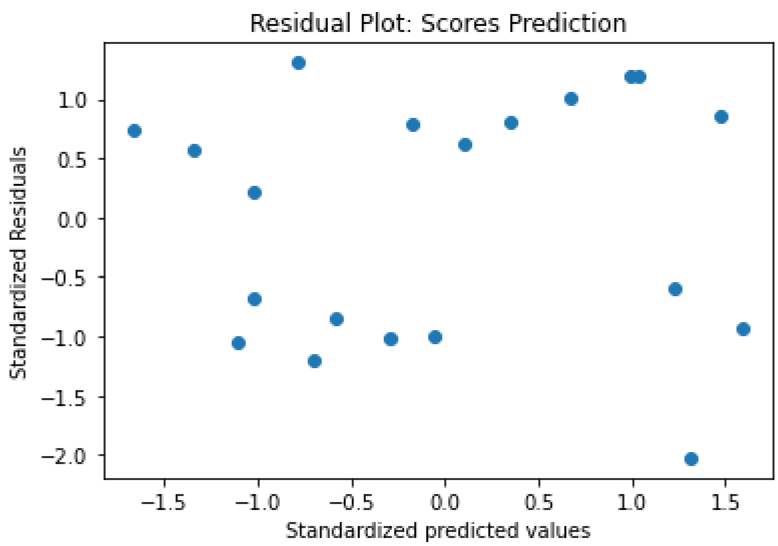
In [10]:
```
#  Statistical data required for diagnosing regression model
print('Summary of model:', stu_scores_lm.summary2())
```

```
Summary of model:                   Results: Ordinary least squares
=================================================================
Model:              OLS              Adj. R-squared:     0.949
Dependent Variable: y                AIC:                129.3715
Date:               2021-07-15 01:47 BIC:                131.3630
No. Observations:   20               Log-Likelihood:     -62.686
Df Model:           1                F-statistic:        353.5
Df Residuals:       18               Prob (F-statistic): 2.79e-13
R-squared:          0.952            Scale:              34.331
-------------------------------------------------------------------
           Coef.    Std.Err.     t      P>|t|     [0.025    0.975]
-------------------------------------------------------------------
const      2.0182     3.0570   0.6602   0.5175   -4.4043    8.4407
x1         9.9107     0.5271  18.8023   0.0000    8.8033   11.0181
-----------------------------------------------------------------
Omnibus:               4.659         Durbin-Watson:         1.813
Prob(Omnibus):         0.097         Jarque-Bera (JB):      1.720
Skew:                  -0.296        Prob(JB):              0.423
Kurtosis:              1.691         Condition No.:         14
=================================================================
```

In [11]:
```
# Step-6 Model Dignostics to validate the data
# Residual analysis-Test of Homoscedasticity
stu_scores_resid = stu_scores_lm.resid
```

In [12]:
```
def get_standardized_values(vals):
    """
    Test of Homoscedasticity
    :param vals: series of variable values
    :return: standardized values of vals
    """
    return (vals - vals.mean())/vals.std()
```

In [13]:
```
plt.scatter(get_standardized_values(stu_scores_lm.fittedvalues), get_standardi
zed_values(stu_scores_resid))
plt.title("Residual Plot: Scores Prediction")
plt.xlabel("Standardized predicted values")
plt.ylabel("Standardized Residuals")
plt.show()
```

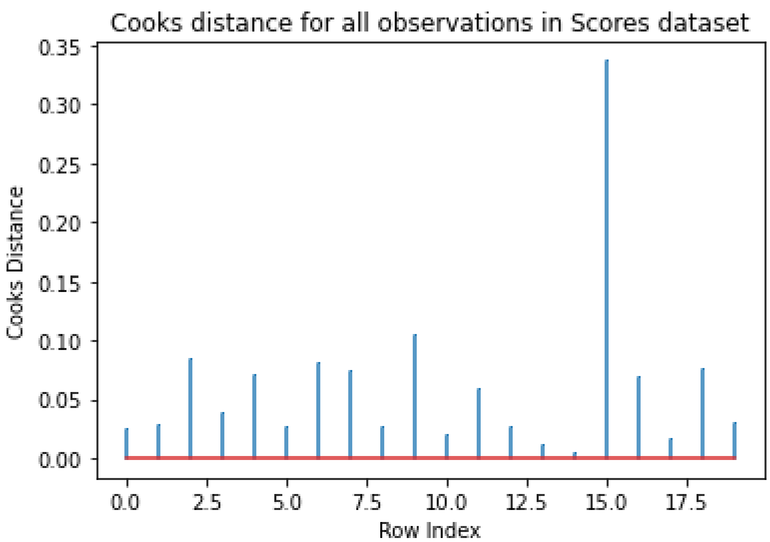


In [14]:
```
# Outlier Analysis(most influential observation)- Z_score
from scipy.stats import zscore
student_score_df['z_score_percent'] = zscore(student_score_df.Scores)
Z_score = student_score_df[(student_score_df.z_score_percent > 3.0) | (student
_score_df.z_score_percent < -3.0)]
print(Z_score)
```

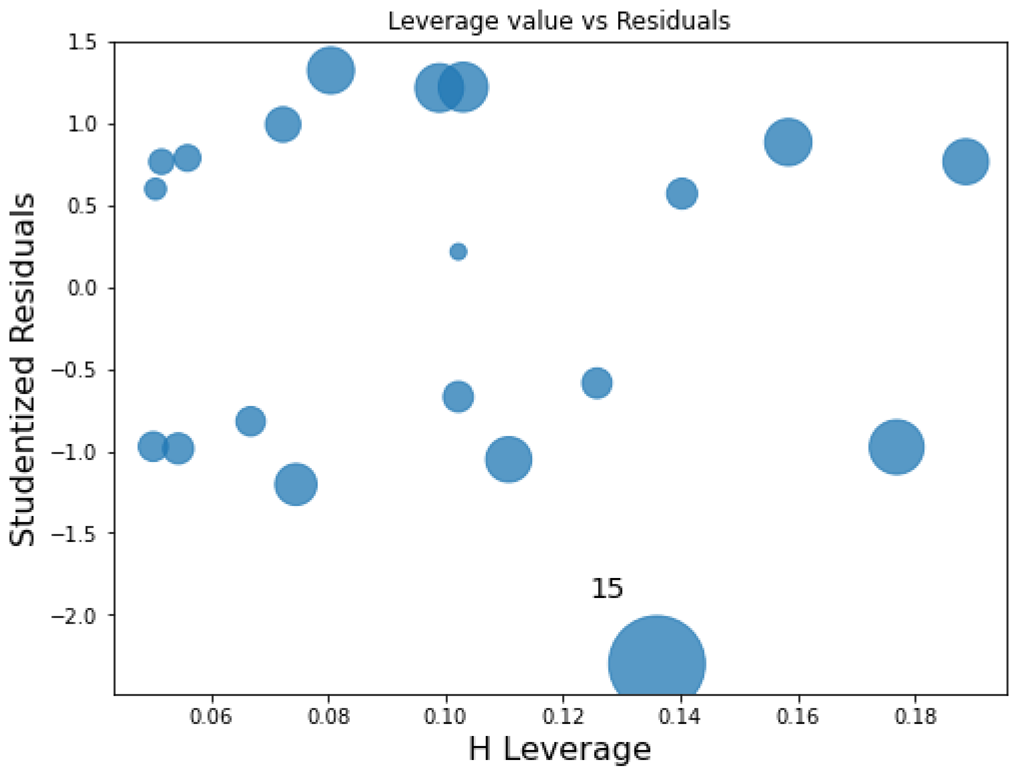
```
Empty DataFrame
Columns: [Hours, Scores, z_score_percent]
Index: []
```

In [15]:
```
# Cook's Distance
score_influence = stu_scores_lm.get_influence()
(c, p) = score_influence.cooks_distance
plt.stem(np.arange(len(train_X)),
         np.round(c, 3),
         markerfmt=",")
plt.title("Cooks distance for all observations in Scores dataset")
plt.xlabel("Row Index")
plt.ylabel("Cooks Distance")
plt.show()
```

```
<ipython-input-15-6238179c4d94>:4: UserWarning: In Matplotlib 3.3 individual
lines on a stem plot will be added as a LineCollection instead of individual
lines. This significantly improves the performance of a stem plot. To remove
this warning and switch to the new behaviour, set the "use_line_collection" k
eyword argument to True.
  plt.stem(np.arange(len(train_X)),
```



In [16]:
```
# leverage values
from statsmodels.graphics.regressionplots import influence_plot
fig, ax = plt.subplots(figsize=(8, 6))
influence_plot(stu_scores_lm, ax=ax)
plt.title("Leverage value vs Residuals")
plt.show()
```



In [17]:
```
# Step-7 Making Prediction
# predicting using validation set
pred_y = stu_scores_lm.predict(test_X)
# print(pred_y)
```

In [18]:
```
df = pd.DataFrame({'Actual': test_y, 'Predicted': pred_y})
print(df)
```

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

In [19]:
```
# you have to create a DataFrame since the Statsmodels formula interface expec
ts it

new_X = pd.DataFrame({'Const': [1], 'Hours': [9.25]})

y_new = stu_scores_lm.predict(new_X)

# Y = np.array(data[predict])

print("Given out of sample input hours : \n", new_X)
print("Prediction of scores : \n", y_new)
```

```
Given out of sample input hours :
    Const  Hours
0      1   9.25
Prediction of scores :
 0    93.691732
dtype: float64
```

In [20]:
```
# Formula for Mean Absolute Percentage Error
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

In [21]:
```
# Step-8 Different measures for accuracy of prediction

from sklearn import metrics
print('Mean Absolute Error: \t \t \t',
      metrics.mean_absolute_error(test_y, pred_y))

from sklearn.metrics import r2_score, mean_squared_error
print('r2_score: \t \t \t \t', np.abs(r2_score(test_y, pred_y)))

print('Mean Squared Error: \t \t \t', np.sqrt(mean_squared_error(test_y, pred_
y)))

print('Mean Absolute Percentage Error: \t', mean_absolute_percentage_error(tes
t_y, pred_y))
```

```
Mean Absolute Error:                    4.183859899002978
r2_score:                               0.9454906892105355
Mean Squared Error:                     4.647447612100368
Mean Absolute Percentage Error:         12.568891617045674
```