In [1]:
```
%cd C:\Users\manoj\Downloads
```

```
C:\Users\manoj\Downloads
```

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [6]:
```
df= pd.read_csv('student_scores.csv')
```

In [7]:
```
df.info
```

Out[7]:
```
<bound method DataFrame.info of     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
10    7.7      85
11    5.9      62
12    4.5      41
13    3.3      42
14    1.1      17
15    8.9      95
16    2.5      30
17    1.9      24
18    6.1      67
19    7.4      69
20    2.7      30
21    4.8      54
22    3.8      35
23    6.9      76
24    7.8      86>
```

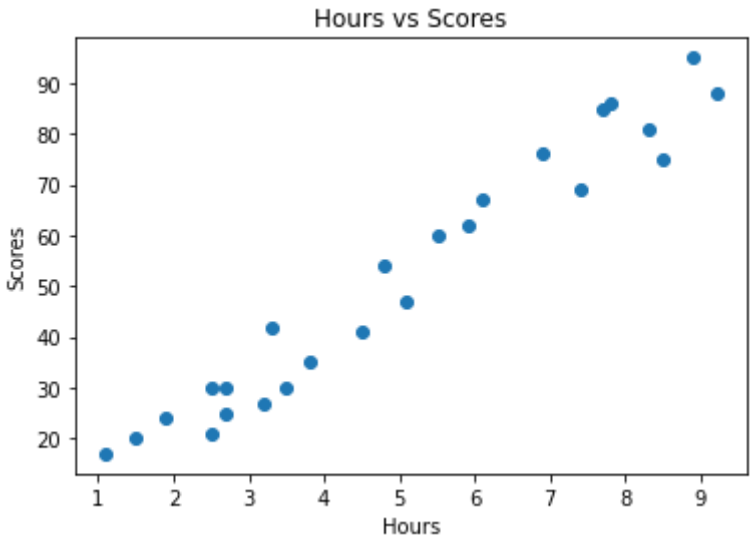
In [8]:
```
df.head()
```

Out[8]:

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

In [9]:
```
plt.scatter(df['Hours'],df['Scores'])
plt.xlabel('Hours')
plt.ylabel('Scores')
plt.title('Hours vs Scores')
```

Out[9]:
```
Text(0.5, 1.0, 'Hours vs Scores')
```



In [19]:
```
X=np.array(df.iloc[:,:-1])
```

In [24]:
```
y=np.array(df.iloc[:,1])
```

In [25]:
```
print(X)
```

```
[[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]]
```

In [26]:
```
print(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 [27]:
```
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 [28]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[28]:
```
LinearRegression()
```

In [30]:
```
print(regressor.intercept_)
```

```
2.018160041434683
```

In [31]:
```
print(regressor.coef_)
```

```
[9.91065648]
```

In [33]:
```
y_pred=regressor.predict(X_test)
```

In [34]:
```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

Out[34]:

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

In [35]:
```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 4.183859899002975
Mean Squared Error: 21.5987693072174
Root Mean Squared Error: 4.6474476121003665
```

In [36]:
```
df.describe()
```

Out[36]:

	Actual	Predicted
count	5.000000	5.000000
mean	41.600000	42.651852
std	22.255336	24.407192
min	20.000000	16.884145
25%	27.000000	26.794801
50%	30.000000	33.732261
75%	62.000000	60.491033
max	69.000000	75.357018

In [38]:
```
print(y_pred)
```

```
[16.88414476 33.73226078 75.357018   26.79480124 60.49103328]
```

In [39]:
```
print(X_test)
```

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

In [ ]:
```
#multilinear regression
```

In [1]:
```
%cd C:\Users\manoj\Downloads
```

```
C:\Users\manoj\Downloads
```

In [11]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [12]:
```
df= pd.read_csv('petrol_consumption.csv')
```

In [13]:
```
df.info
```

Out[13]:
```
<bound method DataFrame.info of     Petrol_tax  Average_income  Paved_Highways  Population_Driver_licence(%)  \
0         9.00            3571            1976                         0.525
1         9.00            4092            1250                         0.572
2         9.00            3865            1586                         0.580
3         7.50            4870            2351                         0.529
4         8.00            4399             431                         0.544
5        10.00            5342            1333                         0.571
6         8.00            5319           11868                         0.451
7         8.00            5126            2138                         0.553
8         8.00            4447            8577                         0.529
9         7.00            4512            8507                         0.552
10        8.00            4391            5939                         0.530
11        7.50            5126           14186                         0.525
12        7.00            4817            6930                         0.574
13        7.00            4207            6580                         0.545
14        7.00            4332            8159                         0.608
15        7.00            4318           10340                         0.586
16        7.00            4206            8508                         0.572
17        7.00            3718            4725                         0.540
18        7.00            4716            5915                         0.724
19        8.50            4341            6010                         0.677
20        7.00            4593            7834                         0.663
21        8.00            4983             602                         0.602
22        9.00            4897            2449                         0.511
23        9.00            4258            4686                         0.517
24        8.50            4574            2619                         0.551
25        9.00            3721            4746                         0.544
26        8.00            3448            5399                         0.548
27        7.50            3846            9061                         0.579
28        8.00            4188            5975                         0.563
29        9.00            3601            4650                         0.493
30        7.00            3640            6905                         0.518
31        7.00            3333            6594                         0.513
32        8.00            3063            6524                         0.578
33        7.50            3357            4121                         0.547
34        8.00            3528            3495                         0.487
35        6.58            3802            7834                         0.629
36        5.00            4045           17782                         0.566
37        7.00            3897            6385                         0.586
```

38	8.50	3635	3274	0.663
39	7.00	4345	3905	0.672
40	7.00	4449	4639	0.626
41	7.00	3656	3985	0.563
42	7.00	4300	3635	0.603
43	7.00	3745	2611	0.508
44	6.00	5215	2302	0.672
45	9.00	4476	3942	0.571
46	7.00	4296	4083	0.623
47	7.00	5002	9794	0.593

	Petrol_Consumption
0	541
1	524
2	561
3	414
4	410
5	457
6	344
7	467
8	464
9	498
10	580
11	471
12	525
13	508
14	566
15	635
16	603
17	714
18	865
19	640
20	649
21	540
22	464
23	547
24	460
25	566
26	577
27	631
28	574
29	534

```
30                  571
31                  554
32                  577
33                  628
34                  487
35                  644
36                  640
37                  704
38                  648
39                  968
40                  587
41                  699
42                  632
43                  591
44                  782
45                  510
46                  610
47                  524  >
```

In [14]:
```
df.head()
```

Out[14]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consumption
0	9.0	3571	1976	0.525	541
1	9.0	4092	1250	0.572	524
2	9.0	3865	1586	0.580	561
3	7.5	4870	2351	0.529	414
4	8.0	4399	431	0.544	410

In [20]:
```
X = df[['Petrol_tax', 'Average_income', 'Paved_Highways',
       'Population_Driver_licence(%)']]
y = df['Petrol_Consumption']
```

In [25]:
```
X=np.array(df.iloc[:,:4])
y=np.array(df.iloc[:,4])
```

In [26]:
```
print(X)
```

```
[[9.0000e+00 3.5710e+03 1.9760e+03 5.2500e-01]
 [9.0000e+00 4.0920e+03 1.2500e+03 5.7200e-01]
 [9.0000e+00 3.8650e+03 1.5860e+03 5.8000e-01]
 [7.5000e+00 4.8700e+03 2.3510e+03 5.2900e-01]
 [8.0000e+00 4.3990e+03 4.3100e+02 5.4400e-01]
 [1.0000e+01 5.3420e+03 1.3330e+03 5.7100e-01]
 [8.0000e+00 5.3190e+03 1.1868e+04 4.5100e-01]
 [8.0000e+00 5.1260e+03 2.1380e+03 5.5300e-01]
 [8.0000e+00 4.4470e+03 8.5770e+03 5.2900e-01]
 [7.0000e+00 4.5120e+03 8.5070e+03 5.5200e-01]
 [8.0000e+00 4.3910e+03 5.9390e+03 5.3000e-01]
 [7.5000e+00 5.1260e+03 1.4186e+04 5.2500e-01]
 [7.0000e+00 4.8170e+03 6.9300e+03 5.7400e-01]
 [7.0000e+00 4.2070e+03 6.5800e+03 5.4500e-01]
 [7.0000e+00 4.3320e+03 8.1590e+03 6.0800e-01]
 [7.0000e+00 4.3180e+03 1.0340e+04 5.8600e-01]
 [7.0000e+00 4.2060e+03 8.5080e+03 5.7200e-01]
 [7.0000e+00 3.7180e+03 4.7250e+03 5.4000e-01]
 [7.0000e+00 4.7160e+03 5.9150e+03 7.2400e-01]
 [8.5000e+00 4.3410e+03 6.0100e+03 6.7700e-01]
 [7.0000e+00 4.5930e+03 7.8340e+03 6.6300e-01]
 [8.0000e+00 4.9830e+03 6.0200e+02 6.0200e-01]
 [9.0000e+00 4.8970e+03 2.4490e+03 5.1100e-01]
 [9.0000e+00 4.2580e+03 4.6860e+03 5.1700e-01]
 [8.5000e+00 4.5740e+03 2.6190e+03 5.5100e-01]
 [9.0000e+00 3.7210e+03 4.7460e+03 5.4400e-01]
 [8.0000e+00 3.4480e+03 5.3990e+03 5.4800e-01]
 [7.5000e+00 3.8460e+03 9.0610e+03 5.7900e-01]
 [8.0000e+00 4.1880e+03 5.9750e+03 5.6300e-01]
 [9.0000e+00 3.6010e+03 4.6500e+03 4.9300e-01]
 [7.0000e+00 3.6400e+03 6.9050e+03 5.1800e-01]
 [7.0000e+00 3.3330e+03 6.5940e+03 5.1300e-01]
 [8.0000e+00 3.0630e+03 6.5240e+03 5.7800e-01]
 [7.5000e+00 3.3570e+03 4.1210e+03 5.4700e-01]
 [8.0000e+00 3.5280e+03 3.4950e+03 4.8700e-01]
 [6.5800e+00 3.8020e+03 7.8340e+03 6.2900e-01]
 [5.0000e+00 4.0450e+03 1.7782e+04 5.6600e-01]
 [7.0000e+00 3.8970e+03 6.3850e+03 5.8600e-01]
 [8.5000e+00 3.6350e+03 3.2740e+03 6.6300e-01]
```

```
[7.0000e+00 4.3450e+03 3.9050e+03 6.7200e-01]
[7.0000e+00 4.4490e+03 4.6390e+03 6.2600e-01]
[7.0000e+00 3.6560e+03 3.9850e+03 5.6300e-01]
[7.0000e+00 4.3000e+03 3.6350e+03 6.0300e-01]
[7.0000e+00 3.7450e+03 2.6110e+03 5.0800e-01]
[6.0000e+00 5.2150e+03 2.3020e+03 6.7200e-01]
[9.0000e+00 4.4760e+03 3.9420e+03 5.7100e-01]
[7.0000e+00 4.2960e+03 4.0830e+03 6.2300e-01]
[7.0000e+00 5.0020e+03 9.7940e+03 5.9300e-01]]
```

In [27]:
```
print(y)
```

```
[541 524 561 414 410 457 344 467 464 498 580 471 525 508 566 635 603 714
 865 640 649 540 464 547 460 566 577 631 574 534 571 554 577 628 487 644
 640 704 648 968 587 699 632 591 782 510 610 524]
```

In [28]:
```
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 [32]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[32]:
```
LinearRegression()
```

In [33]:
```
print(regressor.intercept_)
```

```
425.59933220324103
```

In [34]:
```
print(regressor.coef_)
```

```
[-4.00166602e+01 -6.54126674e-02 -4.74073380e-03  1.34186212e+03]
```

In [36]:
```
y_pred=regressor.predict(X_test)
```

In [37]:
```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

Out[37]:

	Actual	Predicted
0	534	469.391989
1	410	545.645464
2	577	589.668394
3	571	569.730413
4	577	649.774809
5	704	646.631164
6	487	511.608148
7	587	672.475177
8	467	502.074782
9	580	501.270734

In [38]:
```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 56.822247478964734
Mean Squared Error: 4666.344787588368
Root Mean Squared Error: 68.31064915215173
```

In [39]:
```
df.describe()
```

Out[39]:

	Actual	Predicted
count	10.000000	10.000000
mean	549.400000	565.827107
std	80.687601	71.811927
min	410.000000	469.391989
25%	498.750000	504.458123
50%	574.000000	557.687939
75%	579.250000	632.390471
max	704.000000	672.475177

In [40]:
```
print(y_pred)
```

```
[469.39198872 545.64546431 589.66839402 569.7304133  649.77480909
 646.63116356 511.60814841 672.47517717 502.07478157 501.2707342 ]
```