Import Libraries

In [2]:
```
import numpy as np # numpay library is use for Mathematical calculations.
import pandas as pd # pandas is use to Analysis the Data
import matplotlib.pyplot as plt # It is use to Visualize the Data (shows in Diagram format).
```

Load Dataset

In [3]:
```
path = r"C:\Users\Admin\Downloads\archive\student_info.csv"
df = pd.read_csv(path)
```

In [4]:
```
df
```

Out[4]:

	study_hours	student_marks
0	6.83	78.50
1	6.56	76.74
2	NaN	78.68
3	5.67	71.82
4	8.67	84.19
...	...	...
195	7.53	81.67
196	8.56	84.68
197	8.94	86.75
198	6.60	78.05
199	8.35	83.50

200 rows × 2 columns

In [3]:
```
df.head()
```

Out[3]:

	study_hours	student_marks
0	6.83	78.50
1	6.56	76.74
2	NaN	78.68
3	5.67	71.82
4	8.67	84.19

In [4]:
```
df.tail()
```

Out[4]:

	study_hours	student_marks
195	7.53	81.67
196	8.56	84.68
197	8.94	86.75
198	6.60	78.05
199	8.35	83.50

In [5]:
```
df.shape
```

Out[5]:
```
(200, 2)
```

Discover and visualize the data to gain insights

In [6]:
```
df.info() # It gives the information about dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   study_hours    195 non-null    float64
 1   student_marks  200 non-null    float64
dtypes: float64(2)
memory usage: 3.2 KB
```

In [7]:
```
df.describe() # It gives the information about
              # numeirical value in our data
```

Out[7]:

	study_hours	student_marks
count	195.000000	200.00000
mean	6.995949	77.93375
std	1.253060	4.92570
min	5.010000	68.57000
25%	5.775000	73.38500
50%	7.120000	77.71000
75%	8.085000	82.32000
max	8.990000	86.99000

In study_hours colunm (195.0000) students of hours information are present & In study_marks colunm (200.0000) students of marks information are present

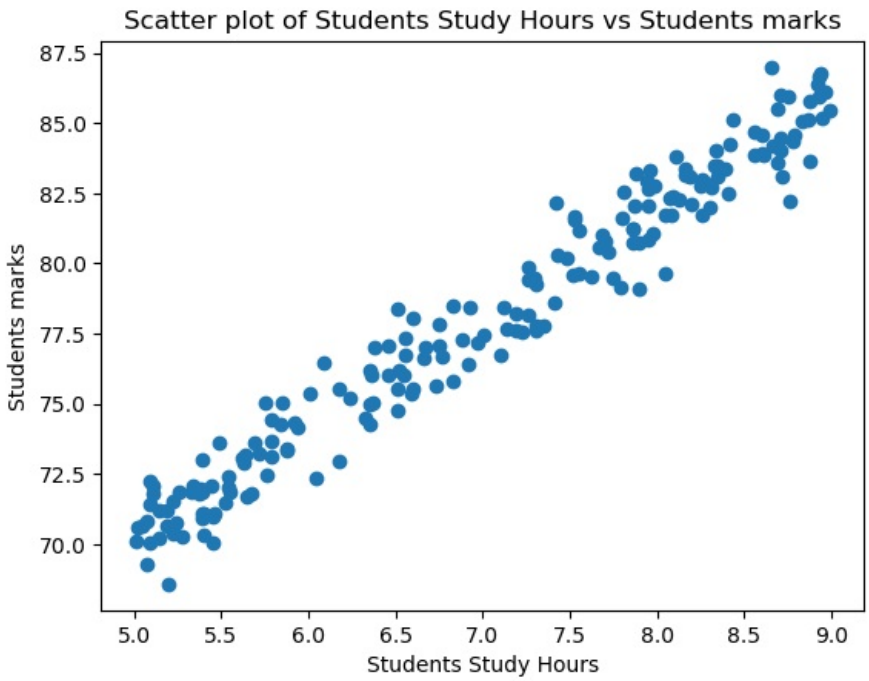
study_hours of mean value is (6.995949)--- means average 7 hours students are studied. study_marks of mean value is (77.93375)--- means average 77.9% marks are getting

Students maximum getting only 86% marks & can't getting above 90% marks so that's why we want to predict the minimum number of hours to study so that we can get 99% marks

Visualize the data

Visualize the data in Scatter plot- (Data visualization is one of the concept here we gives whatever the data or whatever the information as Input the Data visualization & It display the Output in the form of visualiz format of data and that can be in the form of pictorial / Graphical Represention)

In [8]:
```
plt.scatter(x = df.study_hours, y = df.student_marks) # we gives on the x-axis study_hours & on the y-axis stud
plt.xlabel("Students Study Hours")
plt.ylabel("Students marks")
plt.title("Scatter plot of Students Study Hours vs Students marks")
plt.show()
```



Our data is in linear form as students do more study they will get more marks.

Prepare the data for Machine Learning algorithms

Prepare the data for Machine Learning algorithms

In preparation of data we have to clean our data means we have to check missing values in our data

Data Cleaning

Data cleaning refers to the process of identifying and correcting or removing inaccurate, incomplete, or irrelevant data from a dataset.

In [9]:
```
df.isnull() # It display how many data are present or not
            # This function gives the value in (True or False)
```

Out[9]:

	study_hours	student_marks
0	False	False
1	False	False
2	True	False
3	False	False
4	False	False
...	...	...
195	False	False
196	False	False
197	False	False
198	False	False
199	False	False

200 rows × 2 columns

In [10]:
```
df.isnull().sum() # In study_hours there are 5 missing values are present &
                  # In student_marks there are no any missing values.
```

Out[10]:
```
study_hours      5
student_marks    0
dtype: int64
```

5 means we have five missing values in our data which means we have to import data at missing place

In [11]:
```
df.mean() # This function gives the Mean value of all numerical colums
```

Out[11]:
```
study_hours       6.995949
student_marks    77.933750
dtype: float64
```

In [12]:
```
df2 = df.fillna(df.mean()) # It fill the Mean values at the place on NaN values &
                           # We assign the clean value to the New variable (df2)
```

In [13]:
```
df2.isnull().sum() # for checking the NULL value are present or not
```

Out[13]:
```
study_hours      0
student_marks    0
dtype: int64
```

Now we can see there is no missing values in our data which means there is no missing values in our data.

In [14]:
```
df2.head() # Here will display the totaly clean data
```

Out[14]:

	study_hours	student_marks
0	6.830000	78.50
1	6.560000	76.74
2	6.995949	78.68
3	5.670000	71.82
4	8.670000	84.19

split the dataset

study_hours are independunt & study_marks are dependunt on study_hours

Here we gives to x for (student_marks) & y for (study_hours)

In [16]:
```
X = df2.drop("student_marks",axis = "columns") # x means Matrix
y = df2.drop("study_hours",axis = "columns") # y means Vector
```

```
print("shape of X =", X.shape)
print("shape of y =", y.shape)  # Now here we split our data into x & y
X
```

```
shape of X = (200, 1)
shape of y = (200, 1)
```

Out[16]:

	study_hours
0	6.830000
1	6.560000
2	6.995949
3	5.670000
4	8.670000
...	...
195	7.530000
196	8.560000
197	8.940000
198	6.600000
199	8.350000

200 rows × 1 columns

Now here we have to slip our data into training and testing.

In [17]:
```
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=51) # 0.2 means we use t
                                                                                            # for testing
print("shape of X_train =", X_train.shape)                                  # random state=52 means it is pattern
print("shape of y_train =", X_train.shape)                                  # 20% testing data
print("shape of X_test =", X_test.shape)
print("shape of y_test =", X_test.shape)
```

```
shape of X_train = (160, 1)
shape of y_train = (160, 1)
shape of X_test = (40, 1)
shape of y_test = (40, 1)
```

We use 80% data for training and 20% for testing.

When we split our data in training or testing after that we can build machine learning model.

Now for machine learnig model we use the training data.

Select a model and train it

Linear Regression

In [ ]:
```
# y = m * x + c  (we find the m & c)
```

y=mx+c this equation is of straight line here we have x and y but we have not m and c so we have to do work on m and c in machine learning we have algorithm called linear regression (linear means aur data is in linear form and regression means output will be in regression form).

In [19]:
```
from sklearn.linear_model import LinearRegression # our data in linear format that's why we used Linear Regress
lr = LinearRegression()                           # Regression means the output will gives in continuos values
```

In [20]:
```
lr.fit(X_train,y_train) # By using "fit" method we train our LinearRegression model
                        # He took training on (m & c) value
```

Out[20]:
```
▾ LinearRegression
LinearRegression()
```

Now you can see linear regression data is trained.

In [21]:
```
lr.coef_ # coefficient value means m value
```

Out[21]:
```
array([[3.93571802]])
```

In [22]:
```
lr.intercept_ # intercept means c value
```

Out[22]:
```
array([50.44735504])
```

In [23]:
```
# Mathematical formula
m = 3.93
c = 50.44
y = m*4 + c
y
```

Out[23]:
```
66.16
```

If student read four hour then he will get 66.16 % marks

In [24]:
```
lr.predict([[4]])[0][0].round(2) # For get the exact value we used the (round(2)).
```

```
C:\ProgramData\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature name
s, but LinearRegression was fitted with feature names
  warnings.warn(
```

Out[24]:
```
66.19
```

Now the Machine Learning Model to be tested

In [25]:
```
y_pred = lr.predict(X_test) # now we gives the data for testing
y_pred                      # Here we predict the y value
```

Out[25]:
```
array([[83.11381458],
       [78.9025963 ],
       [84.57003024],
       [85.82946001],
       [84.72745896],
       [80.75238377],
       [72.84159055],
       [71.66087515],
       [73.23516235],
       [71.66087515],
       [73.47130543],
       [76.38373677],
       [73.23516235],
       [73.58937697],
       [82.95638585],
       [70.40144538],
       [73.23516235],
       [78.74516758],
       [75.55723598],
       [82.68088559],
       [76.65923703],
       [70.48015974],
       [74.77009238],
       [77.98143645],
       [85.59331693],
       [82.56281405],
       [76.42309395],
       [85.0423164 ],
       [78.39095296],
       [81.38209865],
       [81.73631327],
       [83.15317176],
       [82.20859943],
       [81.10659839],
       [73.58937697],
       [71.1492318 ],
       [71.89701823],
       [81.53952737],
       [72.60544747],
       [71.93637541]])
```

From here we can get how much our machine learning model can predict

In [ ]:
```

```

Now join the Actual value and Predicted value

Our machine learning model gives a accurate prediction

In [26]:
```
pd.DataFrame(np.c_[X_test, y_test, y_pred], columns = ["study_hours", "student_marks_original","student_marks_p
```

Out[26]:

	study_hours	student_marks_original	student_marks_predicted
0	8.300000	82.02	83.113815
1	7.230000	77.55	78.902596
2	8.670000	84.19	84.570030
3	8.990000	85.46	85.829460
4	8.710000	84.03	84.727459
5	7.700000	80.81	80.752384
6	5.690000	73.61	72.841591
7	5.390000	70.90	71.660875
8	5.790000	73.14	73.235162
9	5.390000	73.02	71.660875
10	5.850000	75.02	73.471305
11	6.590000	75.37	76.383737
12	5.790000	74.44	73.235162
13	5.880000	73.40	73.589377
14	8.260000	81.70	82.956386
15	5.070000	69.27	70.401445
16	5.790000	73.64	73.235162
17	7.190000	77.63	78.745168
18	6.380000	77.01	75.557236
19	8.190000	83.08	82.680886
20	6.660000	76.63	76.659237
21	5.090000	72.22	70.480160
22	6.180000	72.96	74.770092
23	6.995949	76.14	77.981436
24	8.930000	85.96	85.593317
25	8.160000	83.36	82.562814
26	6.600000	78.05	76.423094
27	8.790000	84.60	85.042316
28	7.100000	76.76	78.390953
29	7.860000	81.24	81.382099
30	7.950000	80.86	81.736313
31	8.310000	82.69	83.153172
32	8.070000	82.30	82.208599
33	7.790000	79.17	81.106598
34	5.880000	73.34	73.589377
35	5.260000	71.86	71.149232
36	5.450000	70.06	71.897018
37	7.900000	80.76	81.539527
38	5.630000	72.87	72.605447
39	5.460000	71.10	71.936375

Fine-tune our model

Now we test the accuracy

In [27]:
```
lr.score(X_test,y_test)
```

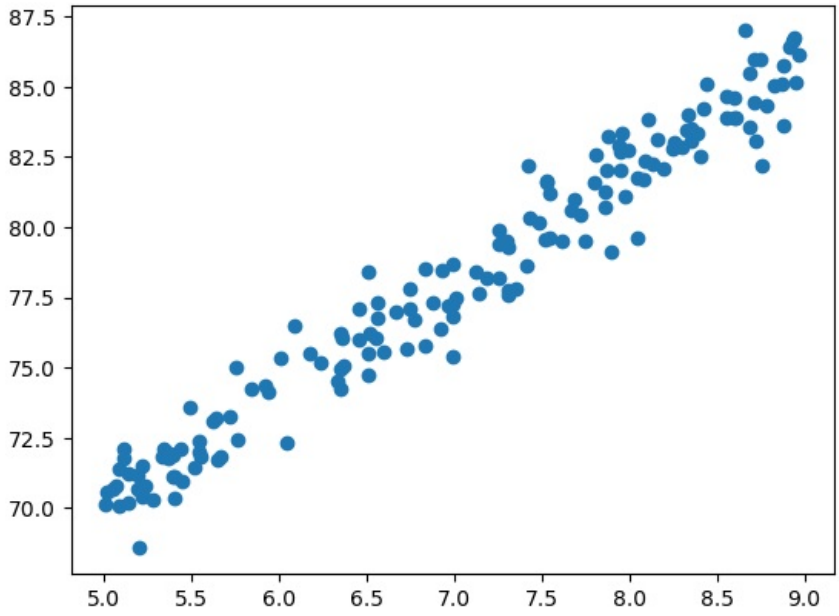
Out[27]:
```
0.9514124242154464
```

This accuracy we are getting from our machine learning model 0.9514124242154464

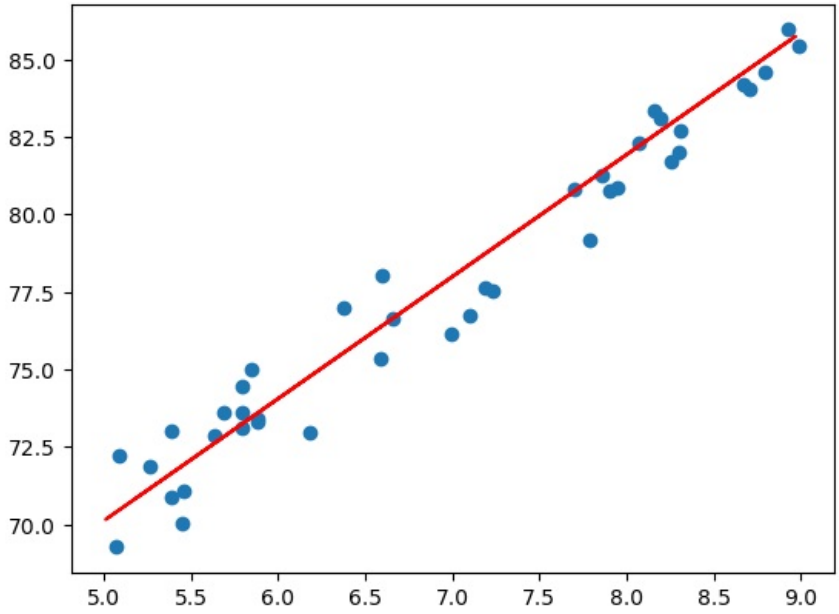
In [ ]:
```
```

If we want to see our model how to predict the line (m * x + c)

In [29]:
```
plt.scatter(X_train,y_train)
plt.show()
```



In [30]:
```
plt.scatter(X_test, y_test)
plt.plot(X_train, lr.predict(X_train), color = "r")
plt.show()
```



Distance between the line and these two points has an error of 5% so that's why giving 95% accuracy

Support Vector Regression

Feactur Scalling

In [31]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)
```

Support Vector Regression - ML Model Training

In [32]:
```
from sklearn.svm import SVR # Here we used the "sklearn" Library to train the SVR
```

In [33]:
```
svr_rbf=SVR(kernel='rbf') # "rbf" is bydefault kernal
svr_rbf.fit(X_train, y_train)
svr_rbf.score(X_test, y_test)
```

```
C:\ProgramData\anaconda3\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-ve
ctor y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example usin
g ravel().
  y = column_or_1d(y, warn=True)
```

Out[33]:
```
0.9529910461937098
```

In [34]:
```
svr_linear = SVR (kernel='linear')
svr_linear.fit(X_train, y_train)
svr_linear.score(X_test, y_test)
```

```
C:\ProgramData\anaconda3\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-ve
ctor y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example usin
g ravel().
  y = column_or_1d(y, warn=True)
```

Out[34]:
```
0.9521482056094251
```

Predict the Student Marks and Test

In [35]:
```
svr_linear.predict([X_test[0]]) # It predict the first row of Student marks
```

Out[35]:
```
array([83.04243449])
```

In [36]:
```
y_pred = svr_linear.predict(X_test) # Predict all of the student marks
y_pred
```

Out[36]:
```
array([83.04243449, 78.9267416 , 84.46561802, 85.69647944, 84.6194757 ,
       80.73456932, 73.00322098, 71.84928839, 73.38786517, 71.84928839,
       73.61865169, 76.46501874, 73.38786517, 73.73404495, 82.88857681,
       70.61842696, 73.38786517, 78.77288392, 75.65726593, 82.61932588,
       76.73426968, 70.6953558 , 74.88797754, 78.02647692, 85.46569293,
       82.50393262, 76.50348316, 84.92719105, 78.42670414, 81.35000003,
       81.69617981, 83.08089891, 82.15775284, 81.08074909, 73.73404495,
       71.34925094, 72.08007491, 81.50385771, 72.77243446, 72.11853933])
```

In [37]:
```
y_test # Compare with actual value
```

Out[37]:

	student_marks
148	82.02
104	77.55
4	84.19
7	85.46
192	84.03
160	80.81
118	73.61
58	70.90
190	73.14
174	73.02
23	75.02
10	75.37
115	74.44
86	73.40
67	81.70
68	69.27
177	73.64
171	77.63
128	77.01
14	83.08
82	76.63
50	72.22
45	72.96
31	76.14
176	85.96
21	83.36
198	78.05
89	84.60
35	76.76
36	81.24
113	80.86
121	82.69
99	82.30
162	79.17
79	73.34
131	71.86
65	70.06
13	80.76
85	72.87
42	71.10

In [ ]:

Model Evaluation

Root Mean Square error

In [38]:
```
from sklearn.metrics import mean_squared_error # To find the error
import numpy as np

mse = mean_squared_error(y_test, y_pred)
rsme=np.sqrt(mse)
print('MSE=',mse)
print('RMSE=',rsme)
```

```
MSE= 1.0912249880452323
RMSE= 1.0446171490288834
```

Ridge and Lasso Regression

In [39]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [40]:
```
rd=Ridge()
rd.fit(X_train,y_train)
rd.score(X_test,y_test)
```

Out[40]:
```
0.9520029324089081
```

In [41]:
```
ls1=Lasso()
ls1.fit(X_train,y_train)
ls1.score(X_test,y_test)
```

Out[41]:
```
0.9267681219261026
```

In [42]:
```
rd2=Ridge(alpha=2)
rd2.fit(X_train,y_train)
rd2.score(X_test,y_test)
```

Out[42]:
```
0.9525052057150583
```

In [43]:
```
ls2=Lasso(alpha=2)
ls2.fit(X_train,y_train)
ls2.score(X_test,y_test)
```

Out[43]:
```
0.8107859673990103
```

In [44]:
```
ls2=Lasso(alpha=3)
ls2.fit(X_train,y_train)
ls2.score(X_test,y_test)
```

Out[44]:
```
0.6034659606341695
```

In [45]:
```
ls2=Lasso(alpha=1)
ls2.fit(X_train,y_train)
ls2.score(X_test,y_test)
```

Out[45]:
```
0.9267681219261026
```

In [46]:
```
ls =Lasso(alpha=2)
rd=Ridge()
rd.fit(X_train,y_train)
rd.score(X_test,y_test)
```

Out[46]:
```
0.9520029324089081
```

In [ ]:
```

```