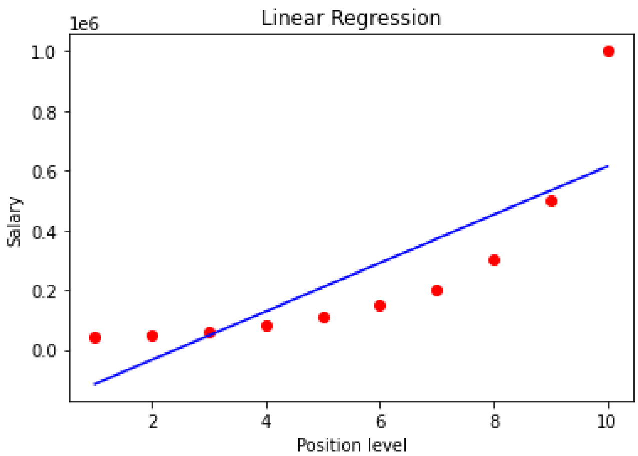
1. Go through position_salaries.csv file and show that it is not following linear regression and can be best modelled with polynomial regression. It consists of only two fields' position and salary.

In [1]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
df = pd.read_csv(r'C:\Users\Nirmalya Majhi\Desktop\Advanced IT Workshop\position_salaries1.csv')
position_type = {
    'Business Analyst': 1,
    'Junior Consultant': 2,
    'Senior Consultant': 3,
    'Manager': 4,
    'Country Manager': 5,
    'Region Manager': 6,
    'Partner': 7,
    'Senior Partner': 8,
    'C-level': 9,
    'CEO': 10
}
df['position_type'] = df['Position'].apply(position_type.get)
X = df[['position_type']]
Y = df['Salary']
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=1/3,random_state=0)
lin_reg = LinearRegression()
lin_reg.fit(X,Y)
plt.scatter(X,Y,color='red')
plt.plot(X, lin_reg.predict(X), color='blue')
plt.title('Linear Regression')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```



Discussion:

from the above plot, we can tell that it is not following linear regression as it makes a curve path(shows in dotted path). So it can be best modelled with polynomial regression.

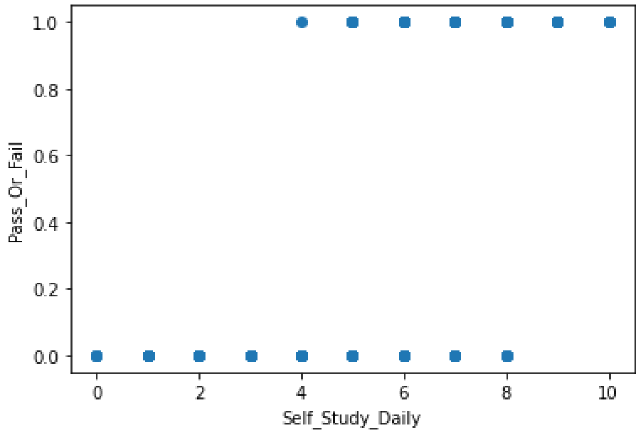
2. Go through "Student-Pass-Fail-Data.csv" where self -study daily and tuition monthly are the two influential factors where 1 is pass and 0 is for fail. Use logistic regression and now reduce the number of rows to half and see the success rate has it influenced by the data.

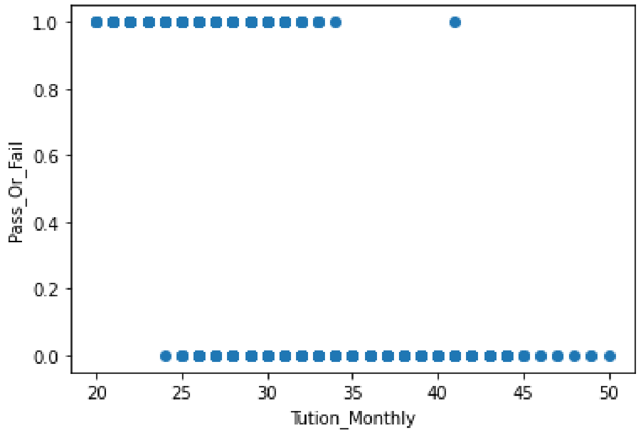
In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
df = pd.read_csv(r"C:\Users\Nirmalya Majhi\Desktop\Advanced IT Workshop\Student-Pass-Fail-Data.csv")
df.size
df1 = df
```

```
plt.scatter(df1['Self_Study_Daily'],df1['Pass_Or_Fail'])
plt.xlabel('Self_Study_Daily')
plt.ylabel('Pass_Or_Fail')
plt.show()
plt.scatter(df1['Tution_Monthly'],df1['Pass_Or_Fail'])
plt.xlabel('Tution_Monthly')
plt.ylabel('Pass_Or_Fail')
plt.show()
X = df1.drop('Pass_Or_Fail',axis = 1)
Y = df1['Pass_Or_Fail']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=4)
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train,Y_train)
LogisticRegression(C=1.0,class_weight=None,dual=False,fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
verbose=0, warm_start=False)
Y_pred = logistic_regression.predict(X_test)

accuracy = metrics.accuracy_score(Y_test,Y_pred)
accuracy_percentage = 100 * accuracy
print("The percentage of accurate prediction = ",accuracy_percentage,"%")
```





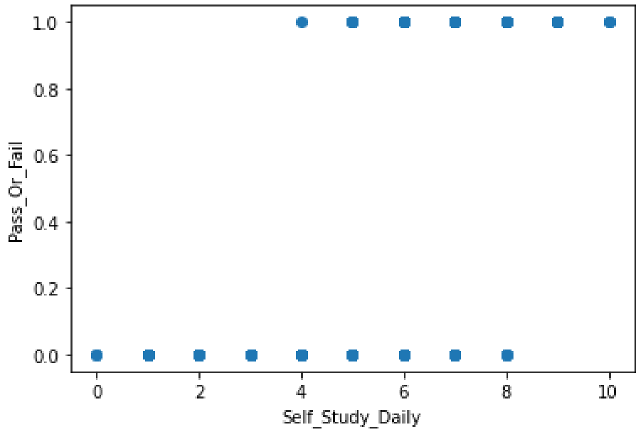

```
The percentage of accurate prediction =  96.8 %
```

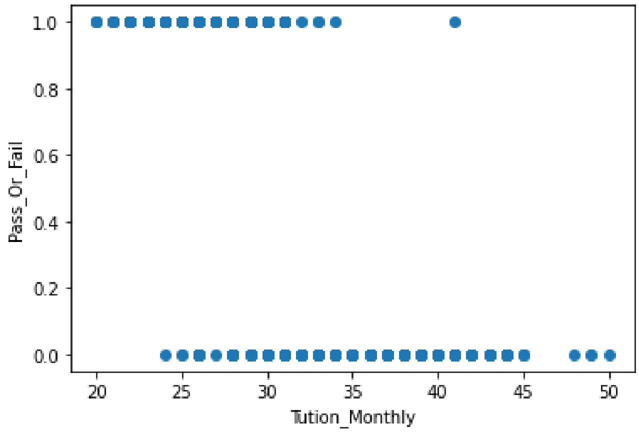
Now, we run the same regression but considering half of the data to see if the percentage of prediction improves.

In [3]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
df = pd.read_csv(r"C:\Users\Nirmalya Majhi\Desktop\Advanced IT Workshop\Student-Pass-Fail-Data.csv")
df.size
df1 = df.head(500)
plt.scatter(df1['Self_Study_Daily'],df1['Pass_Or_Fail'])
plt.xlabel('Self_Study_Daily')
plt.ylabel('Pass_Or_Fail')
plt.show()
plt.scatter(df1['Tution_Monthly'],df1['Pass_Or_Fail'])
plt.xlabel('Tution_Monthly')
plt.ylabel('Pass_Or_Fail')
plt.show()
```

```
X = df1.drop('Pass_Or_Fail',axis = 1)
Y = df1['Pass_Or_Fail']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=4)
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train,Y_train)
LogisticRegression(C=1.0,class_weight=None,dual=False,fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,

penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
verbose=0, warm_start=False)
Y_pred = logistic_regression.predict(X_test)
accuracy = metrics.accuracy_score(Y_test,Y_pred)
accuracy_percentage = 100 * accuracy
print("The percentage of accurate prediction = ",accuracy_percentage,"%")
```

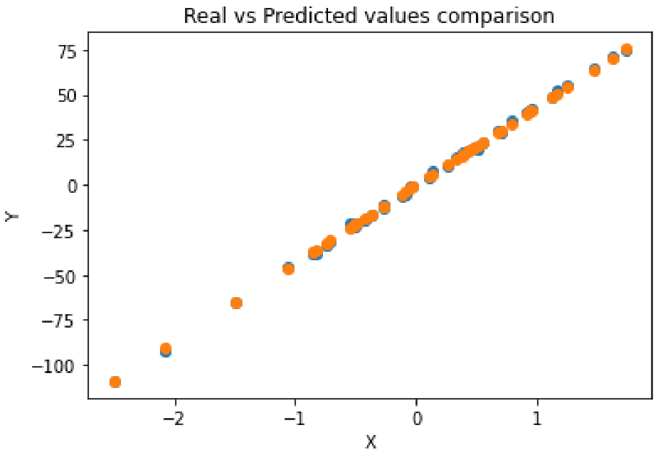





```
The percentage of accurate prediction =  98.4 %
```

4. From sklearn.datasets import make regression and fit the data and perform the linear regression. Use scatter plot.

In [4]:
```
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
X,Y = make_regression(n_samples=150, n_features=1, noise=1)
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.3,random_state=0)
model = LinearRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Real vs Predicted values comparison')
plt.scatter(X_test,Y_test)
plt.scatter(X_test,Y_pred)
plt.show()
```



5. Take Y= [5, 4, 3, 6,7, 8,9, 5,4,,3,1] , X= [yoe, level, leow, city] such that yoe and leow is directly proportional to the data and level is moderately dependent and calculate R2 and equation slope and intercept for yoe, level and leow which is the best parameter.

In [5]:

```
import numpy as np
from sklearn.linear_model import LinearRegression
yoe = np.array([3,2,1,3,4,5,5,3,2,1,1]).reshape(-1, 1)
level = np.array([2,1,0,3,4,4,5,2,1,0,0]).reshape(-1, 1)
leow = np.array([5,3,2,5,5,8,8,4,3,2,0]).reshape(-1, 1)
city = np.array([1,1,2,5,4,3,6,7,4,2,3]).reshape(-1, 1)
Y = np.array([5,4,3,6,7,8,9,5,4,3,1])
print('When X=[ yoe, level, leow, city] :\n')
X = np.array([yoe, level, leow, city]).reshape(11,4)
model = LinearRegression()
model.fit(X, Y)
r_sq = model.score(X,Y)
print('R-squared = ',r_sq)
print('\nWhen X=[yoe] :')
X = yoe
model = LinearRegression()
model.fit(X, Y)
r_sq = model.score(X,Y)
print('\nR-squared = ',r_sq)
print('intercept:', model.intercept_)
print('slope:', model.coef_)
b0 = (model.intercept_).round(4)
b1 = (model.coef_[0]).round(4)
print("The required equation for parameter X=[yoe] is -->  Y = ",b0,"+",b1,"* X ")
print('\nWhen X=[level] :')
X = level
model = LinearRegression()
model.fit(X, Y)
r_sq = model.score(X,Y)
print('\nR-squared = ',r_sq)
print('intercept:', model.intercept_)
print('slope:', model.coef_)
b0 = (model.intercept_).round(4)
b1 = (model.coef_[0]).round(4)
print("The required equation for parameter X=[level] is -->  Y = ",b0,"+",b1,"* X ")
print('\nWhen X=[leow] :')
X = leow
model = LinearRegression()
model.fit(X, Y)
r_sq = model.score(X,Y)
print('\nR-squared = ',r_sq)
print('intercept:', model.intercept_)
print('slope:', model.coef_)
b0 = (model.intercept_).round(4)
b1 = (model.coef_[0]).round(4)
print("The required equation for parameter X=[leow] is -->  Y = ",b0,"+",b1,"* X ")
```

```
When X=[ yoe, level, leow, city] :

R-squared =  0.5004017672661639

When X=[yoe] :

R-squared =  0.9306206088992974
intercept: 0.8196721311475397
slope: [1.53278689]
The required equation for parameter X=[yoe] is -->  Y =  0.8197 + 1.5328 * X

When X=[level] :

R-squared =  0.9380580357142857
intercept: 2.4375000000000004
slope: [1.28125]
The required equation for parameter X=[level] is -->  Y =  2.4375 + 1.2812 * X

When X=[leow] :

R-squared =  0.9525319829424307
intercept: 1.171641791044776
slope: [0.9358209]
The required equation for parameter X=[leow] is -->  Y =  1.1716 + 0.9358 * X
```

6. Take Y= [5,4,3,6,7,8,9,5,4,3] and X= [[3,2], [2,1], [1,0],[3,3],[4,4],[5,4],[5,5],[3,2],[2,1],[1,0]] Where x0= yoe and x1= level. Calculate R2 and equation slope and intercept.

In [6]:

```python
Y_=[ 5,4,3,6,7,8,9,5,4,3]
X_= [[3,2],[2,1],[1,0],[3,3],[4,4],[5,4],[5,5],[3,2],[2,1],[1,0]]
X_ = pd.DataFrame(X_, columns=['yoe', 'level'])
model = LinearRegression().fit(X_, Y_)
print('R-squared = ',model.score(X_, Y_))
print('intercept:', model.intercept_)
print('slope:', model.coef_[0])
```

```
R-squared =  0.9845377604166666
intercept: 2.291666666666665
slope: 0.4791666666666671
```

7. Take the following x = np.arange(10).reshape(-1, 1); y = np.array([0, 0, 0, 1, 1, 1, 1, 1, 1, 1]). Design a Logistic Regression. What value of c gives you optimum result. Modify your model till you get 100% accuracy. (c=1,5,10) show the result.

In [7]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
x = np.arange(10).reshape(-1, 1)
y = np.array([0, 0, 0, 1, 1, 1, 1, 1, 1, 1])
model = LogisticRegression(solver='liblinear',random_state=0,C=1)
model.fit(x, y)
pred_m = model.predict(x)
print(pred_m)
print("Accuracy Score for c = 1: ", round(accuracy_score(pred_m, y) * 100, 2))
model = LogisticRegression(solver='liblinear',random_state=0,C=5)
model.fit(x, y)
pred_m = model.predict(x)
print(pred_m)
print("Accuracy Score for c = 5: ", round(accuracy_score(pred_m, y) * 100, 2))
model = LogisticRegression(solver='liblinear',random_state=0,C=10)
model.fit(x, y)
pred_m = model.predict(x)
print(pred_m)
print("Accuracy Score for c = 10: ", round(accuracy_score(pred_m, y) * 100, 2))
```

```
[0 0 1 1 1 1 1 1 1 1]
Accuracy Score for c = 1:  90.0
[0 0 0 1 1 1 1 1 1 1]
Accuracy Score for c = 5:  100.0
[0 0 0 1 1 1 1 1 1 1]
Accuracy Score for c = 10:  100.0
```