In [1]:

```
#IMPORTING THE LIBRARY
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Gradient Decent Algorithm Function

In [2]:

```
def train_model(x,y,alpha,max_epoch):

    #Creating the x0 = 1
    m = y.size
    weight = np.zeros((x.shape[1], 1))
    z_weight = np.zeros((x.shape[1], 1))

    #Creating a list to store our hist_lost
    cost_list = []

    #Loop for demanding iterration
    for i in range(max_epoch):

        #yprediciton/y hat = dot product of weight and x
        #calling the prediction function and passing weight and x for argument
        y_hat = prediction(weight,x)
        y_new = (y-y_hat)

        #updating the weight for every epoch
        j = 0

        while j < weight.shape[0]:
            k = 0
            sum_weight = 0
            while k < m:
                temp_weight = 0
                temp_weight = x[k][j] * y_new[k]
                sum_weight = sum_weight + temp_weight
                k = k + 1
            z_weight[j] = (-1/m)*sum_weight

            j = j + 1

        l = 0
        while l < weight.shape[0]:
            weight[l] = weight[l] - (alpha*z_weight[l][0])
            l = l+1

        #Calculating the cost/ hist_lost using function

        cost =loss_fn(y,y_hat)
        cost_list.append(cost)
        print("Cost is :", cost, "  iteration:   ", i+1)

    #retuning weight and cost_list

    return weight, cost_list
```

Prediction Function

In [3]:

```
#following the formula yprediction = w0x0 + w1x1+...+w5x5

def prediction(w,x):
    yprediction =np.dot(x,w)
    return yprediction
```

Loss Function

In [4]:

```
#Following the formula of 1/m (y-ypredic)^2

def loss_fn(y,yhat):
    cost=(1/(2*y.size))*np.sum(np.square(y-yhat))
    return cost
```

Training and Test Dataset

In [5]:

```

    #READING OUR DATA
    data = pd.read_csv("assignment1_dataset.csv",sep=',')
    data

    #SEPERATE THE TRAINING SET AND TEST SET
    #for training set
    y_train= data.iloc[0:800,5]
    x_train = data.iloc[0:800,0:5]

    #for test set
    y_test= data.iloc[800:1001,5]
    x_test= data.iloc[800:1001,0:5]

    #RESHAPE THE DATA TO AVOID SEQUENCE ERROR

    y_train=np.array(y_train,dtype=float).reshape(y_train.shape[0],1)
    y_test=np.array(y_test,dtype=float).reshape(y_test.shape[0],1)


    #Adding Weight = 0 at first column of x

    x_train = np.vstack((np.ones((x_train.shape[0], )), x_train.T)).T
    x_test= np.vstack((np.ones((x_test.shape[0], )), x_test.T)).T

    #default value for learing rate, cannot use 0.001 because otherwise
    #the iterration should be 10k otherwise we won't get atleast local minnima
    alpha=0.01

    #iterration value
    iterration= 1000
```

Weight and History Lost for Train Seting

In [6]:

```
#printing the the cost for every iterration adn store it
weight_train,hist_lost = train_model(x_train,y_train,alpha,iterration)
```

```
Cost is : 365.6631449762335   iteration:    42
Cost is : 358.0932845786134   iteration:    43
Cost is : 350.68639839196715   iteration:    44
Cost is : 343.43896425000713   iteration:    45
Cost is : 336.34753639795497   iteration:    46
Cost is : 329.40874382852616   iteration:    47
Cost is : 322.61928865428695   iteration:    48
Cost is : 315.97594451558837   iteration:    49
Cost is : 309.47555502329504   iteration:    50
Cost is : 303.11503223554655   iteration:    51
Cost is : 296.89135516780414   iteration:    52
Cost is : 290.8015683354533   iteration:    53
Cost is : 284.84278032824767   iteration:    54
Cost is : 279.01216241589674   iteration:    55
Cost is : 273.30694718411314   iteration:    56
Cost is : 267.7244272004535   iteration:    57
Cost is : 262.26195370929776   iteration:    58
Cost is : 256.9169353553298   iteration:    59
Cost is : 251.68683693489345   iteration:    60
Cost is : 246.56917817461283   iteration:    61
```

Weight for TrainingSet

In [7]:

```
print('The weight that we got after doin Gradiesnt Descent on Training Set')
weight_train
```

```
The weight that we got after doin Gradiesnt Descent on Training Set
```

Out[7]:

```
array([[ 9.80055588],
       [11.89119152],
       [ 0.03884797],
       [ 0.14245249],
       [36.85310611],
       [ 0.08566134]])
```

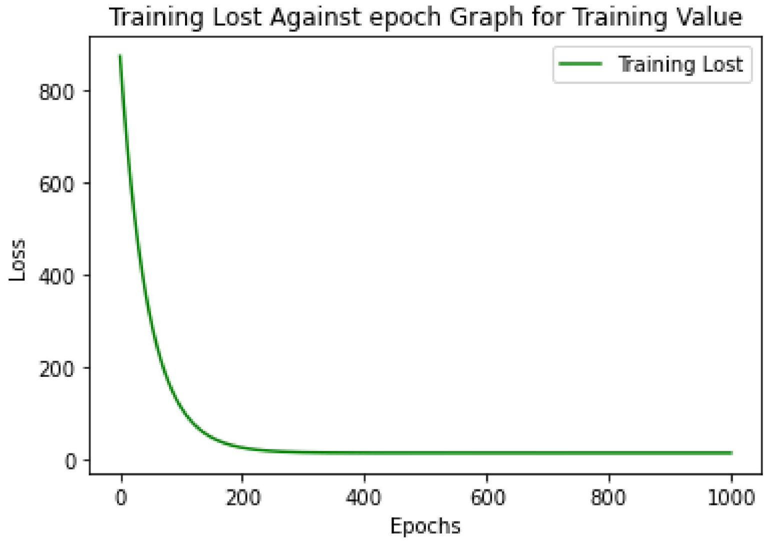
Graph of Training Lost Against Epoch for Training Set

In [8]:

```
traint =[]

for i in range(len(hist_lost)):
    traint.append(i)


plt.plot(traint,hist_lost,'g',label='Training Lost')
plt.title('Training Lost Against epoch Graph for Training Value')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Test Set

In [9]:

```
#calculating the y prediction for test set using training set weight.

y_test_predict= prediction(weight_train, x_test)

#calculating the mean squared error measure

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


MSE_test = mean_squared_error(y_test,y_test_predict)
r2=r2_score(y_test,y_test_predict)

#the lower the result of MSE, the better our model
#Higher R2 means the better our model

print('The Mean Squared Error For Test Set is',MSE_test)
print('The R2 Squared Error for Test Set is', r2)
```

```
The Mean Squared Error For Test Set is 23.59173197547919
The R2 Squared Error for Test Set is 0.9852484727536246
```

Plotting the result

In [10]:

```
import matplotlib.pyplot as plt

plt.figure()
plt.scatter(y_test,y_test_predict)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title( 'Actual vs predicted')
x = np.linspace(-100, 50, 100)
y = x
plt.plot(x, y, 'r')
```

Out[10]:

```
[<matplotlib.lines.Line2D at 0x1b460d99400>]
```