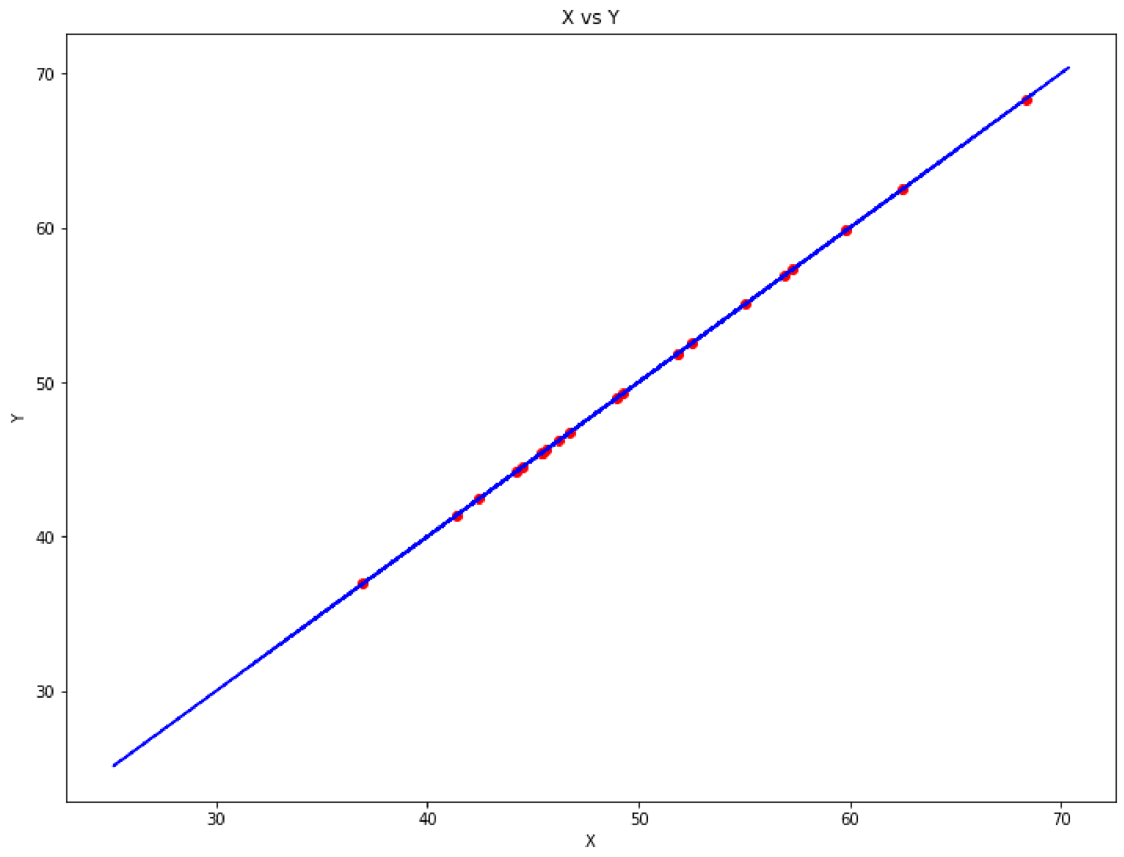
19MCEC08_Regression

1 Simple Linear Regression

[11]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12.0, 9.0)
data = pd.read_csv('data.csv')

X = data.iloc[:, :-1].values
Y = data.iloc[:, :1].values

#Splitting the dataset into the Training set and Test set

from sklearn.model_selection import train_test_split
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size =.20,␣
 ↪random_state = 0)

#Fitting Simple Linear Regression to the training set

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_Train, Y_Train)


#Predicting the Test set result

Y_Pred = regressor.predict(X_Test)


#Test data set output
plt.scatter(X_Test, Y_Test, color = 'red')
plt.plot(X_Train, regressor.predict(X_Train), color = 'blue')
plt.title('X vs Y')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

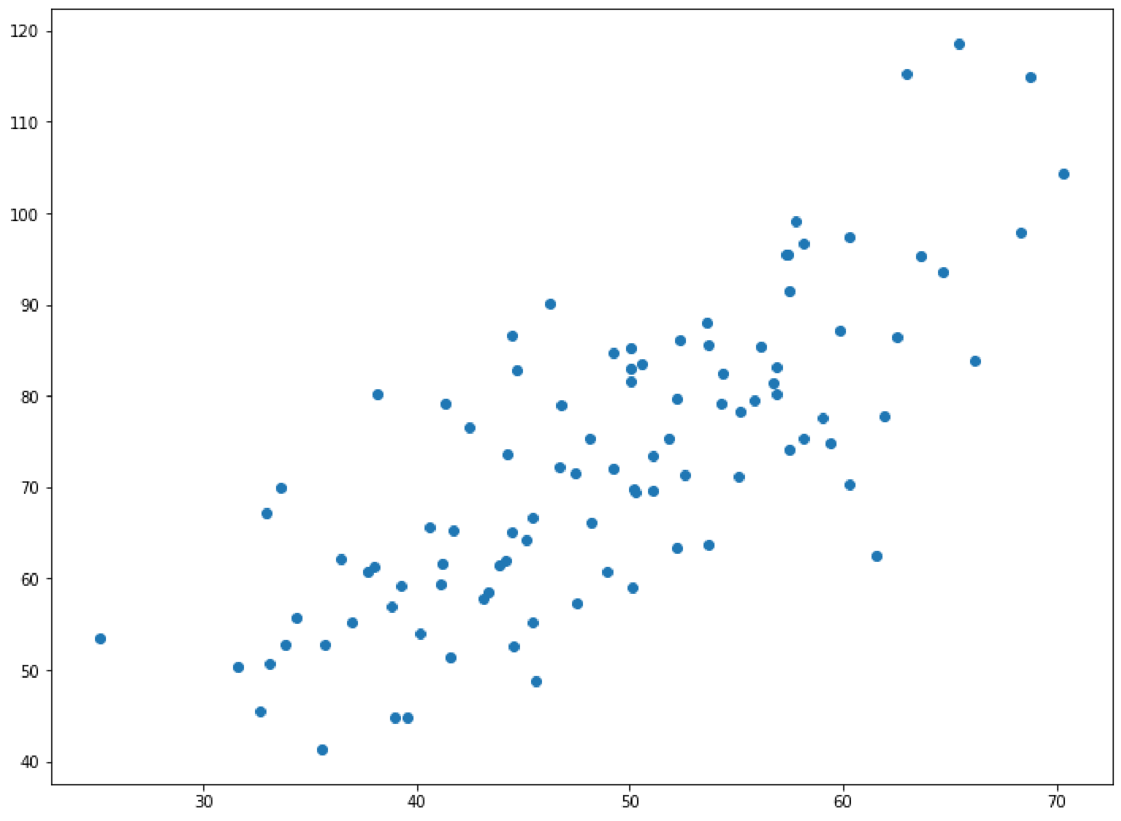


Analysis:- Simple Linear Regression with only one feature Used LinearRegression() class for training and predicting. After prediction we have displayed the calculated co-efficients and mean squared error of the data. It is found that if we have to predict continuous values to be predicted then Linear Regression can be used effectively.

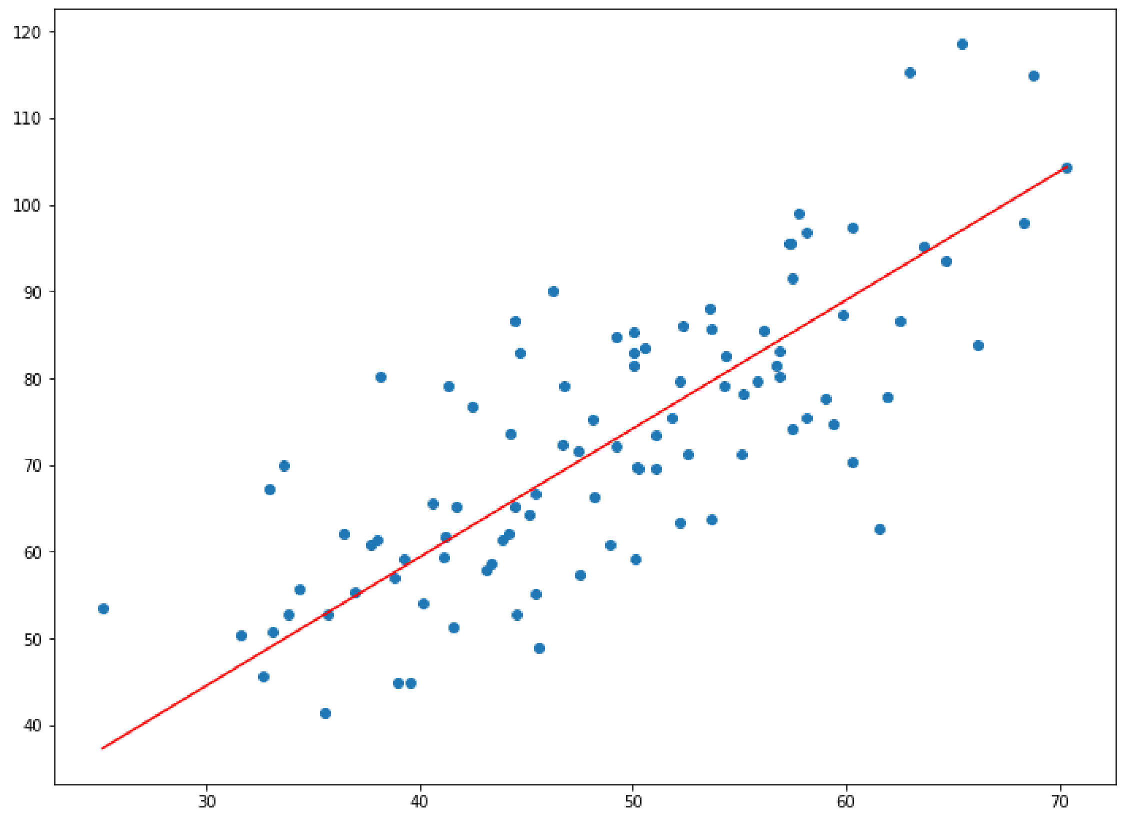
2 Linear Regression from Scratch

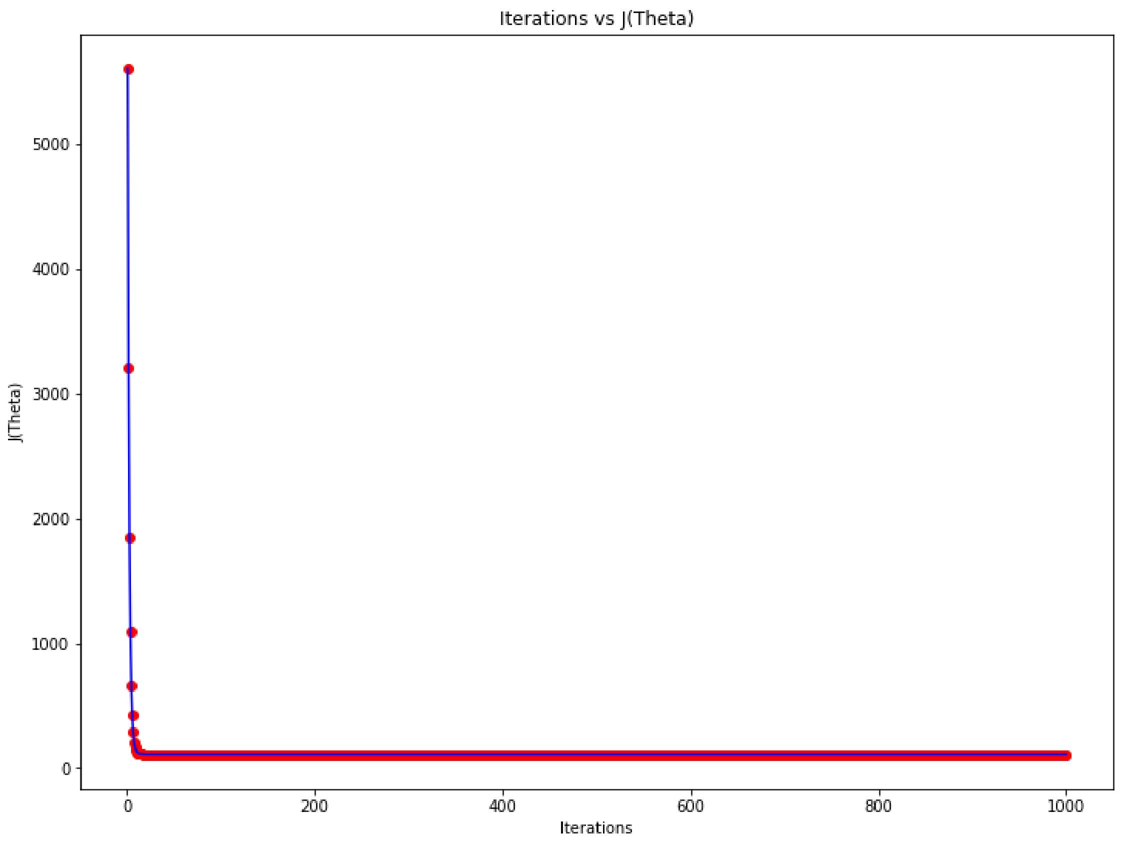
[10]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12.0, 9.0)
data = pd.read_csv('data.csv')
X = data.iloc[:, 0]
Y = data.iloc[:, 1]
plt.scatter(X, Y)
plt.show()
m = 0
c = 0
```

```
L = 0.0001
epochs = 1000

n = float(len(X))

j =np.zeros((1000,1),dtype=np.float64)
itr =np.zeros(shape=(1000,),dtype=np.float64)

v=0
for i in range(0,1000):
    itr[v] = i+1
    v+=1
# Gradient Descent
for i in range(epochs):
    Y_pred = m*X + c
    j[i]=(1/n)*sum((Y_pred-Y)**2)
    D_m = (-1/n) * sum(X * (Y - Y_pred))
    D_c = (-1/n) * sum(Y - Y_pred)
    m = m - L * D_m
    c = c - L * D_c

# Value of parameters Theta0 and Theta1(m and c)
print ("The value of m and c are ",m," and ",c," respectively.")

Y_pred = m*X + c


plt.scatter(X, Y)
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='red')
plt.show()

# Graph of number of iterations vs  Error
plt.scatter(itr, j, color = 'red')
plt.plot(itr, j, color = 'blue')
plt.title('Iterations vs J(Theta)')
plt.xlabel('Iterations')
plt.ylabel('J(Theta)')
plt.show()
```



```
The value of m and c are  1.4821363992234418  and  0.06459149352201031
respectively.
```


120
110
100
90
80
70
60
50
40
30
40
50
60
70



Analysis:- Using Gradient Descent Algorithm we find best co-efficients for the model.From the graph plotted above it represents that after as iterations of the algorithm increases the Error decreases.

3 Multiple Regression

[26]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_boston
from sklearn.metrics import *

bh_data = load_boston()



boston = pd.DataFrame(bh_data.data, columns=bh_data.feature_names)
```

```
#Creating a new column in the dataframe and loading in the target values from␣
↪the dataset.
boston['MEDV'] = bh_data.target

X = pd.DataFrame(np.c_[boston['LSTAT'], boston['RM']], columns=['LSTAT','RM'])
Y = boston['MEDV']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,␣
↪random_state=9)

lin_reg_mod = LinearRegression()

lin_reg_mod.fit(X_train, y_train)

pred = lin_reg_mod.predict(X_test)

test_set_rmse = (np.sqrt(mean_squared_error(y_test, pred)))

test_set_r2 = r2_score(y_test, pred)

print(test_set_rmse)
print(test_set_r2)
```

```
6.035041736063677
0.6400551238836979
```

Analysis:- Multivariate/multiple linear regression is just a linear regression carried out on more than one independent variable. Make use multiple feature attributes. Above we make use of 'RM' and 'LSTAT' as our variables for linear regression. Then we check the predictions against the actual values by using the RMSE and R-2 metrics, two metrics commonly used to evaluate regression tasks.

4 Normal equation method

[4]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from sklearn import datasets
dataset=pd.read_csv('patientData.csv')
X=dataset.iloc[:,:-1].values
Y=dataset.iloc[:, 3].values


from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
```

```
onehotencoder = OneHotEncoder(categorical_features = [0])
X = onehotencoder.fit_transform(X).toarray()
labelencoder_Y = LabelEncoder()
Y = labelencoder_Y.fit_transform(Y)


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)
from sklearn import linear_model as lm

from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
reg=lm.LinearRegression()
reg.fit(X_train,Y_train)
print('Coefficients : \n', reg.coef_)

from sklearn.metrics import mean_squared_error
Y_pred=reg.predict(X_test)

X_transpose=np.transpose(X)
X_transpose_dot_X=X_transpose.dot(X)
temp_1=np.linalg.inv(X_transpose_dot_X)
temp_2=X_transpose.dot(Y)
theta=temp_1.dot(temp_2)
print("THETA=",theta)
mse=mean_squared_error(Y_test,Y_pred)
print("Mean squared error=",mse)
```

```
Coefficients :
 [-0.01458114  0.01458114  0.20254456  0.05188802]
THETA= [-0.9639077  -0.90656901  0.02008834  0.10732979]
Mean squared error= 0.232054308129123
```

```
D:\Users\kshitij\Anaconda3\lib\site-
packages\sklearn\preprocessing\_encoders.py:415: FutureWarning: The handling of
integer data will change in version 0.22. Currently, the categories are
determined based on the range [0, max(values)], while in the future they will be
determined based on the unique values.
If you want the future behaviour and silence this warning, you can specify
"categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the
```

```
categories to integers, then you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
D:\Users\kshitij\Anaconda3\lib\site-
packages\sklearn\preprocessing\_encoders.py:451: DeprecationWarning: The
'categorical_features' keyword is deprecated in version 0.20 and will be removed
in 0.22. You can use the ColumnTransformer instead.
  "use the ColumnTransformer instead.", DeprecationWarning)
```

Analysis:- Normal Equation is an analytical approach to Linear Regression with a Least Square Cost Function. We can directly find out the value of without using Gradient Descent.We make use of an equation of form ′=(xT.x)^(-1).xT.y

5 Logistic regression

[5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

w = pd.read_csv('iris.csv')
w = w.iloc[:,1:]
X = w.iloc[:,:4]
y = w.iloc[:,4]
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X = LabelEncoder()
y = labelencoder_X.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.23,␣
 ↪random_state = 0)

from sklearn.linear_model import LogisticRegression
classifier =␣
 ↪LogisticRegression(multi_class='auto',solver='liblinear',max_iter=200)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import␣
 ↪classification_report,confusion_matrix,accuracy_score,recall_score,precision_score,f1_score

print("Confusion matrix : \n "+str(confusion_matrix(y_test,y_pred)))
print("\nClassification Report : " + str(classification_report(y_test,y_pred)))
print("\nAccuracy Score : {0}".format(accuracy_score(y_pred,y_test)))
print("Recall Score : {0}".format(recall_score(y_pred,y_test,average=None)))
print("Precision Score : {0}".
 ↪format(precision_score(y_pred,y_test,average=None)))
```

```
print("F1 Score : {0}".format(f1_score(y_pred,y_test,average=None)))
```

```
Confusion matrix :
 [[12  0  0]
 [ 0 12  2]
 [ 0  0  9]]

Classification Report :               precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       1.00      0.86      0.92        14
           2       0.82      1.00      0.90         9

    accuracy                           0.94        35
   macro avg       0.94      0.95      0.94        35
weighted avg       0.95      0.94      0.94        35


Accuracy Score : 0.9428571428571428
Recall Score : [1.         1.         0.81818182]
Precision Score : [1.         0.85714286 1.        ]
F1 Score : [1.         0.92307692 0.9       ]
```

[6]:
```
from sklearn.model_selection import GridSearchCV
grid_values = {'penalty': ['l1', 'l2'],'C':[0.001,.009,0.01,.09,1,5,10]}
g_c_a = GridSearchCV(classifier,param_grid = grid_values,cv = 2,scoring =␣
 ↪'accuracy')
g_c_a.fit(X_train, y_train)
```

```
D:\Users\kshitij\Anaconda3\lib\site-
packages\sklearn\model_selection\_search.py:813: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and will
be removed in 0.24. This will change numeric results when test-set sizes are
unequal.
  DeprecationWarning)
```

[6]:
```
GridSearchCV(cv=2, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=200, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='liblinear',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'C': [0.001, 0.009, 0.01, 0.09, 1, 5, 10],
                    'penalty': ['l1', 'l2']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

[7]:
```
g_c_a.best_score_
```

[7]:
```
0.9652173913043478
```

[8]:
```
g_c_a.best_params_
```

[8]:
```
{'C': 10, 'penalty': 'l1'}
```

[9]:
```
c_g =␣
 ↪LogisticRegression(C=10,penalty='l1',multi_class='auto',solver='liblinear',max_iter=200)
c_g.fit(X_train, y_train)

y_p_g = c_g.predict(X_test)

print("\nConfusion Matrix : \n " + str(confusion_matrix(y_test,y_p_g)))
print("\nClassification Report : "+ str(classification_report(y_test,y_p_g)))
print("\nAccuracy Score : {0}".format(accuracy_score(y_p_g,y_test)))
print("Recall Score : {0}".format(recall_score(y_p_g,y_test,average=None)))
print("Precision Score : {0}".
 ↪format(precision_score(y_p_g,y_test,average=None)))
print("F1 Score : {0}".format(f1_score(y_p_g,y_test,average=None)))
```

```
Confusion Matrix :
 [[12  0  0]
 [ 0 14  0]
 [ 0  0  9]]

Classification Report :               precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       1.00      1.00      1.00        14
           2       1.00      1.00      1.00         9

    accuracy                           1.00        35
   macro avg       1.00      1.00      1.00        35
weighted avg       1.00      1.00      1.00        35


Accuracy Score : 1.0
Recall Score : [1. 1. 1.]
Precision Score : [1. 1. 1.]
F1 Score : [1. 1. 1.]
```

[10]:
```python
from sklearn.linear_model import LogisticRegression

combinations=[]
accuracies=[]
penalty = ['l1', 'l2']
C = [0.001,.009,0.01,.09,1,5,10]
for i in penalty:
    for j in C:
        cls =␣
↪LogisticRegression(penalty=i,C=j,multi_class='auto',solver='liblinear',max_iter=200)
        cls.fit(X_train, y_train)
        y_pred_train = cls.predict(X_train)
    combinations.append([i,j,accuracy_score(y_train,y_pred_train)])
    accuracies.append(accuracy_score(y_train,y_pred_train))

print("Penalty \t \t C \t \t Accuracy \n")
for i in combinations:
    print(" {0} \t \t {1} \t \t {2} ".format(i[0],i[1],i[2]))

print("Lowest Accuracy is : {0}".format(min(accuracies)))

index=-1
for i in range(0,len(accuracies)):
    if accuracies[i]==min(accuracies):
        index=i
print("Worst Parameters : \nPenalty : {0} , C : {1} ".
↪format(combinations[index][0],combinations[index][1]))
```

```
Penalty                  C                Accuracy

 l1               10              0.9652173913043478
 l2               10              0.9739130434782609
Lowest Accuracy is : 0.9652173913043478
Worst Parameters :
Penalty : l1 , C : 10
```

Analysis:- Logistic regression is basically a supervised classification algorithm. In a classification problem, the target variable(or output), y, can take only discrete values for given set of features(or inputs), X. Just like Linear regression assumes that the data follows a linear function, Logistic regression models the data using the sigmoid function. Above we use grid search with estimator model as logistic regression.