K.N.Toosi University of Technology

Mini project of Machine Learning Course

Mohammadghasem Emamimoghaddam

Std num : 40202314

Dr. Mehdi Alliyari

Mini project Google Colab drive ,

APR-2024

Contents

Question 1 .... ٤
Question 2 .... ١٥
Question 3 .... ٢٥

Figures

Diagram of the learning and assessment 4
Model 4
making data set 5
Data set with data-sep 1 5
Data set with data-sep 0.1 6
Spliting data and compare the model 7
accuracy of LR 7
High-Parameters for LR 8
Predicted with LR 8
the accuracy of the SGD 9
high parameters of SGD 9
Decision boundary for LR 10
Decision boundry for SGD 10
Decision boundry 11
Dataset with Drawdata tool 11
shape of Data 12
2D plot of data set 12
spliting data to train and test 13
LR high parameters 13
SGD high parameters 13
classification with LR 14
Classification with SGD 14
main system 15
modeled system 16
Noraml data table 16
Fault data table 17
Different types of faults: a) outer race fault , b)ball fault , c) inner race fault , d) combination of parts 17
Deviding data to groups 18
shape of the new data set 18
Different methods of feature extraction 19
Shuffeling data 19
Spliting data to train and test after shuffeling 19
Minmax code 20
Coding og Gradient decent method 20

Implimenting the Gradient descent........21
Error diagram........21
Accuracy definition........22
Precision definition........22
accuracy of ther code........23
Loss function diagram........23
Loss function diagram of the train data........23
Loss function of the Test data........24
Data frame........25
Total information of Data........25
Null values in data........26
Removing null values........26
Histogram of the Temperature........27
Histogram of the Apparent Temperature........27
Histogram of the Humidity........28
Histogram of Visibility........28
Heat map Diagram........29
Least square method........30
Recursive least square code........30
MSE of the Temperature........31
prediction of Temperature with LS........31
Prediction of Apparent Temperature with LS........32
MSE for Temperature and Apparent Temperature........33
Prediction of temperature with RLS........33
Prediction of apparent temperature with RLS........34
WLS for Temperature Data........35
WLS for Apparent Temperature Data........35

Question 1

Machine learning totally include 5 steps. Initially the process starts with the data set which all the works and processing will be done on these data set. Then the data set will be analyzed by the different methods to find the pattern and feature relation between data set to focus on them. For instance the feature extraction , decreasing dimension of data and … all done in this section. Next, by using the relation, features and some unique features the machine try to find the pattern of the model with according to the accuracy and method. In the subsequent stage the model have the main pattern of the data set and is able to evaluate the loss function and error and also estimate the target values. Her we figure out the performance of the main system and try to change some parameters which have effects on model such as weights , methos, feature extraction method and so on. But basically, the Machin try to updates its weights after each iteration to reach the best performance.

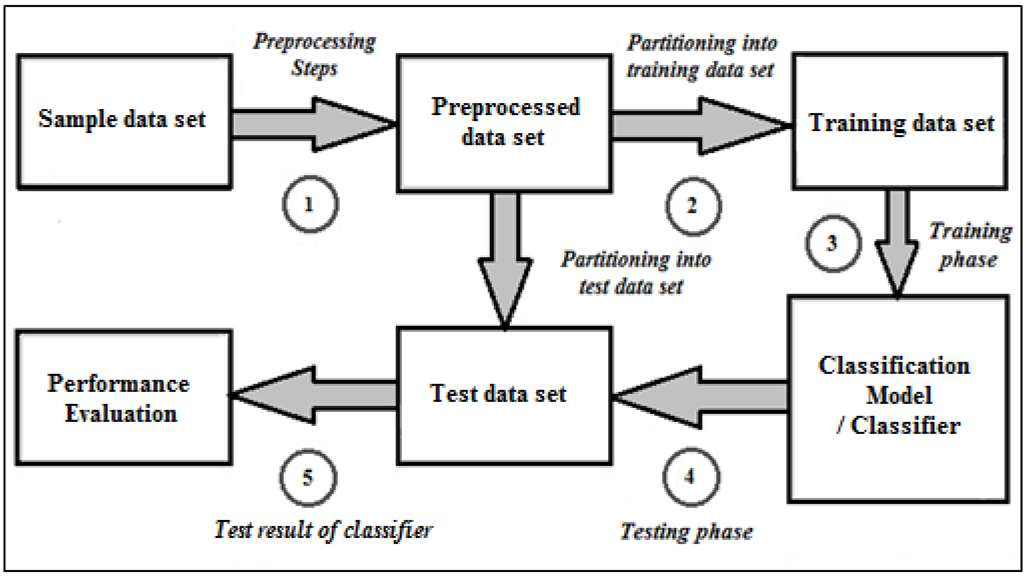


Figure 1 Diagram of the learning and assessment

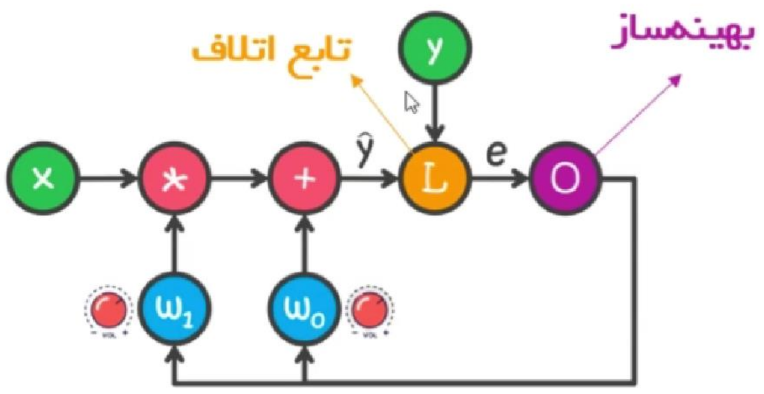
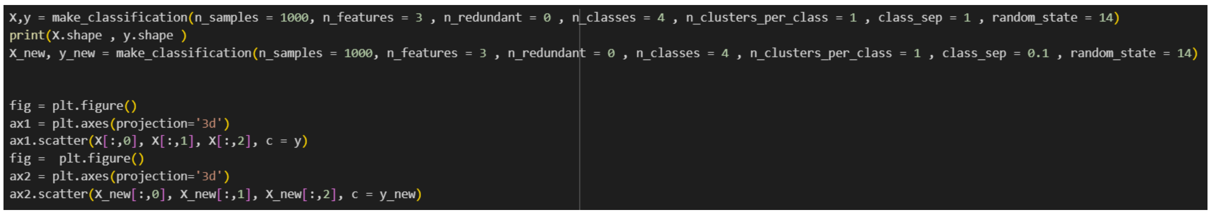


Figure 2 Model

The learning process is based on the dataset which called train data. At the beginning we have data set. We divide this data to two section train and test both of them are similar but the train data is given to the machine with the labeled target while we give the test data to predict the label of the test data. So at first we give the train data to the machine the system try to find best weight to classify as good as possible and decrease the error of the system. this process could be done with different methods such as SGD,GD and LS to calculate the error and feedback the weight to the system. then we test our model by giving the test data to evaluate the result and error then we feed back to the learning model to update the weights. We use different methods to feed back the result to system and we use optimizer here to update weights. So after these steps the optimal model will be get from the train data and now we give test data to the system and assessment the performance of the system with accuracy, possession or other methods for evaluating.

So if we change the classes of the data for example three classes instead of two classes the model will face the changes in the classification model to and label the prediction with 3 types of value and divided to three groups. More over if we have multi classes dataset we should use the loss function which are appropriate for these types of dataset such as; Categorial , Cross-Entropy and Multiclass Hinge Loss.

Making personal data set with Sikitlearn and the code and the result will be shown below:



```
X,y = make_classification(n_samples = 1000, n_features = 3 , n_redundant = 0 , n_classes = 4 , n_clusters_per_class = 1 , class_sep = 1 , random_state = 14)
print(X.shape , y.shape )
X_new, y_new = make_classification(n_samples = 1000, n_features = 3 , n_redundant = 0 , n_classes = 4 , n_clusters_per_class = 1 , class_sep = 0.1 , random_state = 14)


fig = plt.figure()
ax1 = plt.axes(projection='3d')
ax1.scatter(X[:,0], X[:,1], X[:,2], c = y)
fig =  plt.figure()
ax2 = plt.axes(projection='3d')
ax2.scatter(X_new[:,0], X_new[:,1], X_new[:,2], c = y_new)
```

Figure 3 making data set

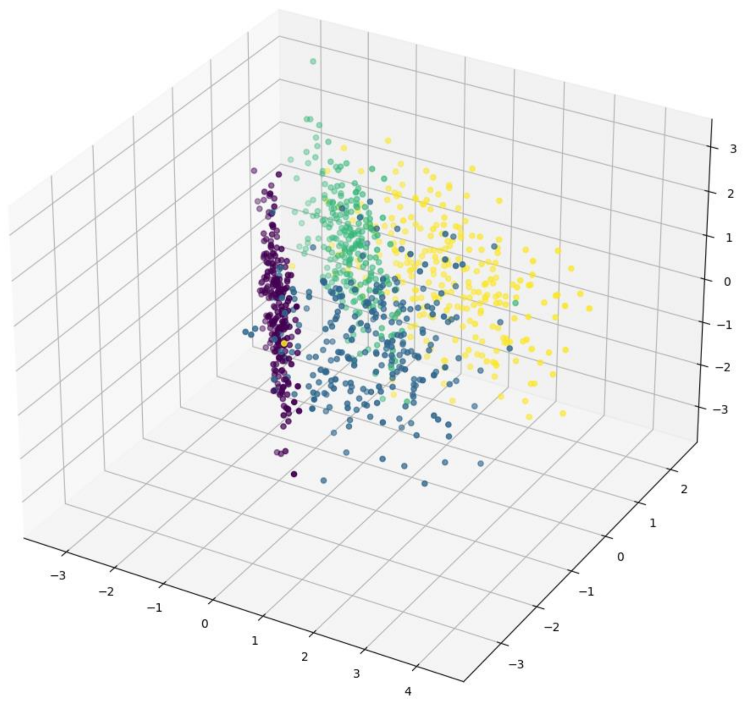


Figure 4 Data set with data-sep 1

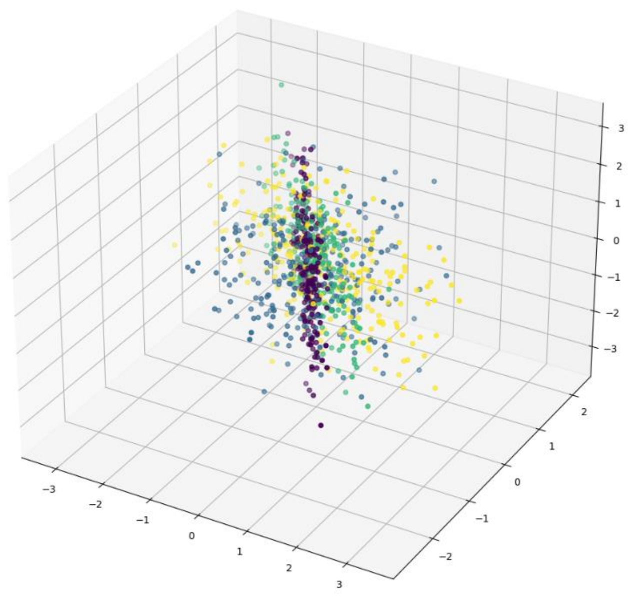
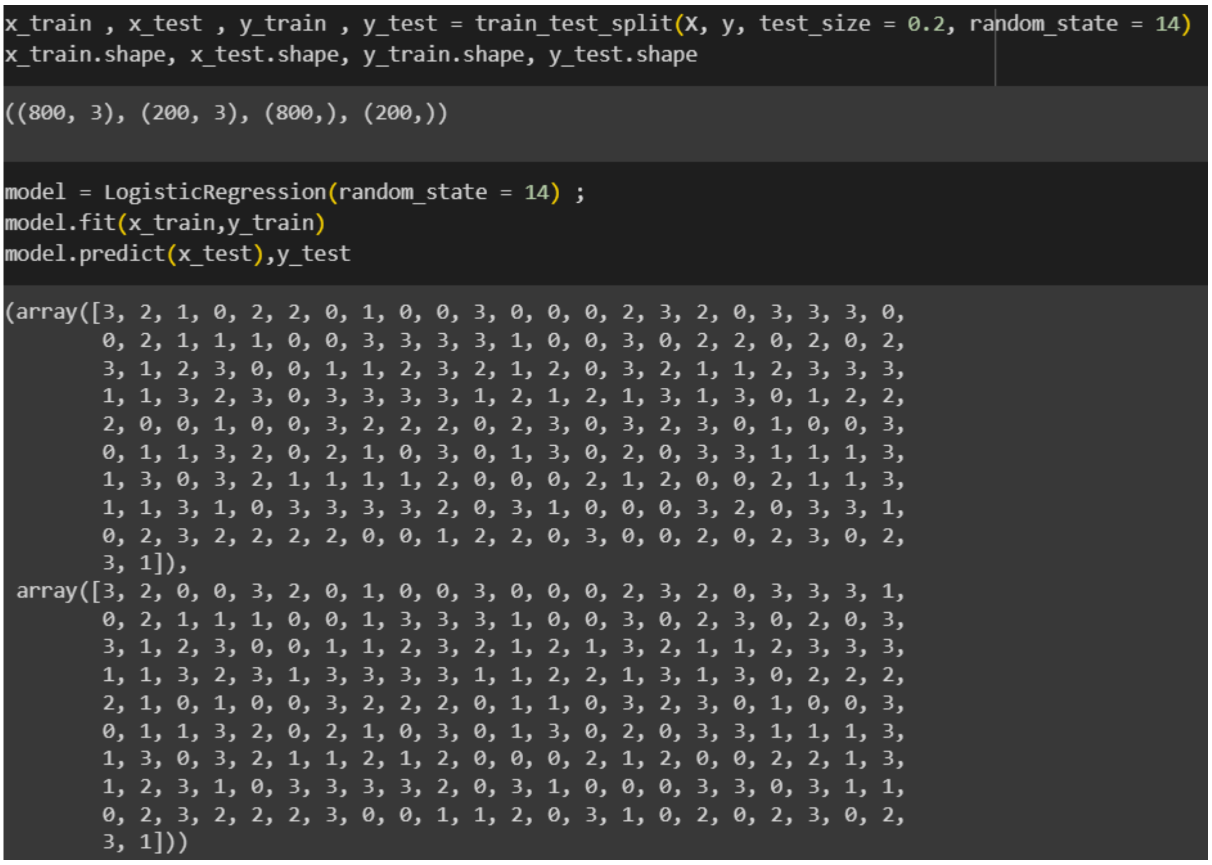


Figure 5 Data set with data-sep 0.1

It is obvious by changing the class_sep value the data set face huge differences. As the Class_sep be smaller it means the classes are mixed and they are separated from each other so it is difficult for the model to classify each data . if the Class_sep be big enough it means the data of different classes are separated from each other and it is clear and simple to classify classes.

As it can be see in above figures the first data set has 3 classes that each one is separated from others while the latter diagram shows the data which are mixed together and it is so hard to classify.

Now it is time to splitting data to two groups of train and test data . then we show the predicted of test data and target data to compare them with each other . her we use Logestic regression from the Sklearn and model the machine. And then we use SGD method.



```
x_train , x_test , y_train , y_test = train_test_split(X, y, test_size = 0.2, random_state = 14)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((800, 3), (200, 3), (800,), (200,))
```

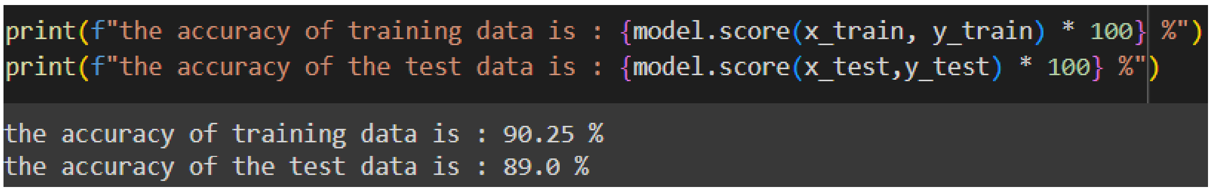
```
model = LogisticRegression(random_state = 14) ;
model.fit(x_train,y_train)
model.predict(x_test),y_test
```

```
(array([3, 2, 1, 0, 2, 2, 0, 1, 0, 0, 3, 0, 0, 0, 2, 3, 2, 0, 3, 3, 3, 0,
        0, 2, 1, 1, 1, 0, 0, 3, 3, 3, 3, 1, 0, 0, 3, 0, 2, 2, 0, 2, 0, 2,
        3, 1, 2, 3, 0, 0, 1, 1, 2, 3, 2, 1, 2, 0, 3, 2, 1, 1, 2, 3, 3, 3,
        1, 1, 3, 2, 3, 0, 3, 3, 3, 3, 1, 2, 1, 2, 1, 3, 1, 3, 0, 1, 2, 2,
        2, 0, 0, 1, 0, 0, 3, 2, 2, 2, 0, 2, 3, 0, 3, 2, 3, 0, 1, 0, 0, 3,
        0, 1, 1, 3, 2, 0, 2, 1, 0, 3, 0, 1, 3, 0, 2, 0, 3, 3, 1, 1, 1, 3,
        1, 3, 0, 3, 2, 1, 1, 1, 1, 2, 0, 0, 0, 2, 1, 2, 0, 0, 2, 1, 1, 3,
        1, 1, 3, 1, 0, 3, 3, 3, 3, 2, 0, 3, 1, 0, 0, 0, 3, 2, 0, 3, 3, 1,
        0, 2, 3, 2, 2, 2, 2, 0, 0, 1, 2, 2, 0, 3, 0, 0, 2, 0, 2, 3, 0, 2,
        3, 1]),
 array([3, 2, 0, 0, 3, 2, 0, 1, 0, 0, 3, 0, 0, 0, 2, 3, 2, 0, 3, 3, 3, 1,
        0, 2, 1, 1, 1, 0, 0, 1, 3, 3, 3, 1, 0, 0, 3, 0, 2, 3, 0, 2, 0, 3,
        3, 1, 2, 3, 0, 0, 1, 1, 2, 3, 2, 1, 2, 1, 3, 2, 1, 1, 2, 3, 3, 3,
        1, 1, 3, 2, 3, 1, 3, 3, 3, 3, 1, 1, 2, 2, 1, 3, 1, 3, 0, 2, 2, 2,
        2, 1, 0, 1, 0, 0, 3, 2, 2, 2, 0, 1, 1, 0, 3, 2, 3, 0, 1, 0, 0, 3,
        0, 1, 1, 3, 2, 0, 2, 1, 0, 3, 0, 1, 3, 0, 2, 0, 3, 3, 1, 1, 1, 3,
        1, 3, 0, 3, 2, 1, 1, 2, 1, 2, 0, 0, 0, 2, 1, 2, 0, 0, 2, 2, 1, 3,
        1, 2, 3, 1, 0, 3, 3, 3, 3, 2, 0, 3, 1, 0, 0, 0, 3, 3, 0, 3, 1, 1,
        0, 2, 3, 2, 2, 2, 3, 0, 0, 1, 1, 2, 0, 3, 1, 0, 2, 0, 2, 3, 0, 2,
        3, 1]))
```

Figure 6 Spliting data and compare the model

First array shows the predicted target and the second one shows the real target value and it can be seen the model is not correct completely and has some differences.

The accuracy of this model is shown as below:



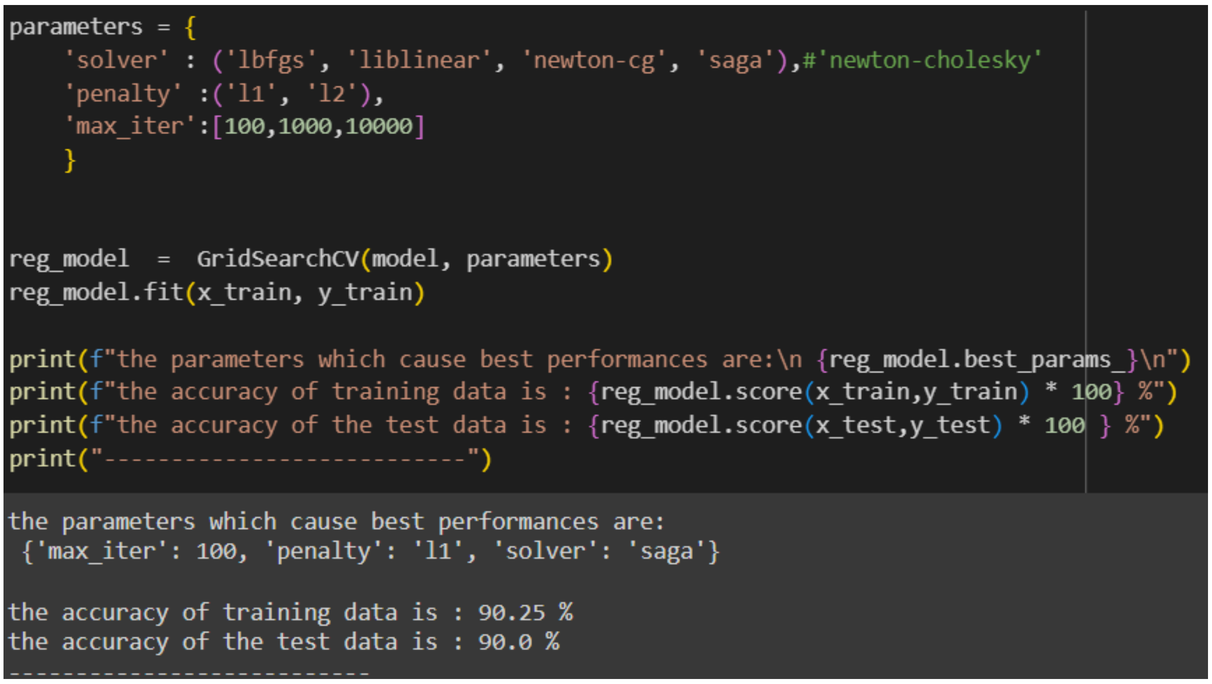
```
print(f"the accuracy of training data is : {model.score(x_train, y_train) * 100} %")
print(f"the accuracy of the test data is : {model.score(x_test,y_test) * 100} %")
```

```
the accuracy of training data is : 90.25 %
the accuracy of the test data is : 89.0 %
```

Figure 7 accuracy of LR

We want to find the best high paremeters which has the best performance for the system so we use Grid search method . in this method we insert different types of high parameters and find out the parameters which have the best accuray rather than others. So here the code check each pair of parameters and make the model and find the accuracoy to compare. Here the best high parameters are :

Max_iter = 100 , penalty = l1 , solver saga. And the accuracy is approximately 90% for test data.

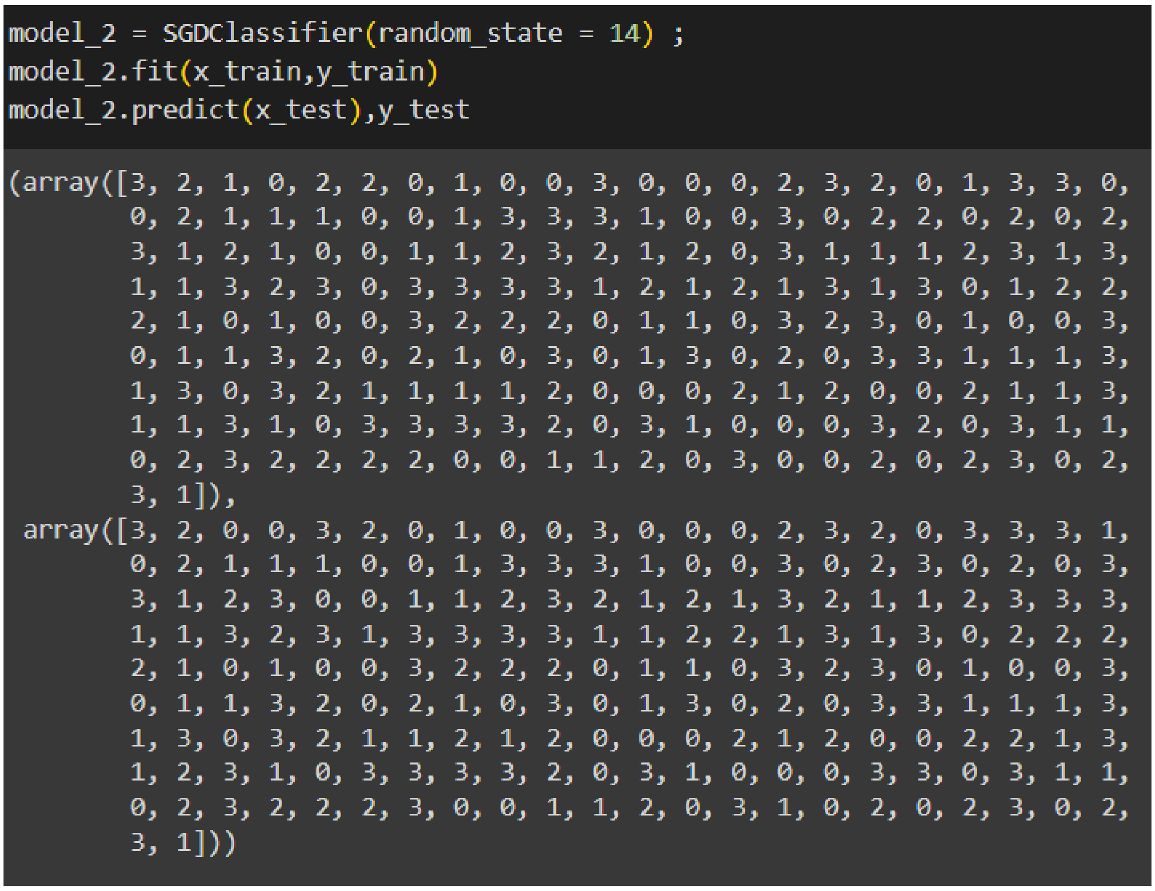


```
parameters = {
    'solver' : ('lbfgs', 'liblinear', 'newton-cg', 'saga'),#'newton-cholesky'
    'penalty' :('l1', 'l2'),
    'max_iter':[100,1000,10000]
    }


reg_model  =  GridSearchCV(model, parameters)
reg_model.fit(x_train, y_train)

print(f"the parameters which cause best performances are:\n {reg_model.best_params_}\n")
print(f"the accuracy of training data is : {reg_model.score(x_train,y_train) * 100} %")
print(f"the accuracy of the test data is : {reg_model.score(x_test,y_test) * 100 } %")
print("---------------------------")
```

```
the parameters which cause best performances are:
 {'max_iter': 100, 'penalty': 'l1', 'solver': 'saga'}

the accuracy of training data is : 90.25 %
the accuracy of the test data is : 90.0 %
---------------------------
```

Figure 8 High-Parameters for LR

Now its time to try our model with another method such as SGD. At first like previous parts we show the predicted value and target value to compare.

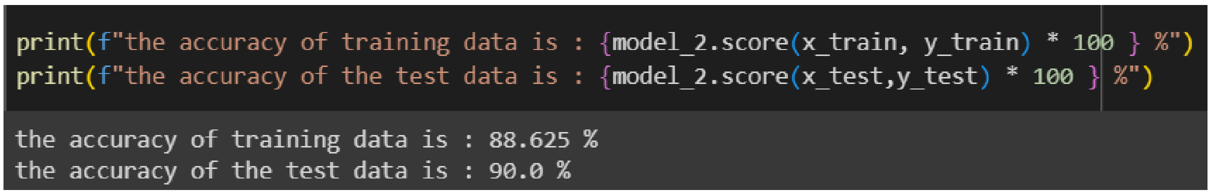


```
model_2 = SGDClassifier(random_state = 14) ;
model_2.fit(x_train,y_train)
model_2.predict(x_test),y_test
```

```
(array([3, 2, 1, 0, 2, 2, 0, 1, 0, 0, 3, 0, 0, 0, 2, 3, 2, 0, 1, 3, 3, 0,
        0, 2, 1, 1, 1, 0, 0, 1, 3, 3, 3, 1, 0, 0, 3, 0, 2, 2, 0, 2, 0, 2,
        3, 1, 2, 1, 0, 0, 1, 1, 2, 3, 2, 1, 2, 0, 3, 1, 1, 1, 2, 3, 1, 3,
        1, 1, 3, 2, 3, 0, 3, 3, 3, 3, 1, 2, 1, 2, 1, 3, 1, 3, 0, 1, 2, 2,
        2, 1, 0, 1, 0, 0, 3, 2, 2, 2, 0, 1, 1, 0, 3, 2, 3, 0, 1, 0, 0, 3,
        0, 1, 1, 3, 2, 0, 2, 1, 0, 3, 0, 1, 3, 0, 2, 0, 3, 3, 1, 1, 1, 3,
        1, 3, 0, 3, 2, 1, 1, 1, 1, 2, 0, 0, 0, 2, 1, 2, 0, 0, 2, 1, 1, 3,
        1, 1, 3, 1, 0, 3, 3, 3, 3, 2, 0, 3, 1, 0, 0, 0, 3, 2, 0, 3, 1, 1,
        0, 2, 3, 2, 2, 2, 2, 0, 0, 1, 1, 2, 0, 3, 0, 0, 2, 0, 2, 3, 0, 2,
        3, 1]),
 array([3, 2, 0, 0, 3, 2, 0, 1, 0, 0, 3, 0, 0, 0, 2, 3, 2, 0, 3, 3, 3, 1,
        0, 2, 1, 1, 1, 0, 0, 1, 3, 3, 3, 1, 0, 0, 3, 0, 2, 3, 0, 2, 0, 3,
        3, 1, 2, 3, 0, 0, 1, 1, 2, 3, 2, 1, 2, 1, 3, 2, 1, 1, 2, 3, 3, 3,
        1, 1, 3, 2, 3, 1, 3, 3, 3, 3, 1, 1, 2, 2, 1, 3, 1, 3, 0, 2, 2, 2,
        2, 1, 0, 1, 0, 0, 3, 2, 2, 2, 0, 1, 1, 0, 3, 2, 3, 0, 1, 0, 0, 3,
        0, 1, 1, 3, 2, 0, 2, 1, 0, 3, 0, 1, 3, 0, 2, 0, 3, 3, 1, 1, 1, 3,
        1, 3, 0, 3, 2, 1, 1, 2, 1, 2, 0, 0, 0, 2, 1, 2, 0, 0, 2, 2, 1, 3,
        1, 2, 3, 1, 0, 3, 3, 3, 3, 2, 0, 3, 1, 0, 0, 0, 3, 3, 0, 3, 1, 1,
        0, 2, 3, 2, 2, 2, 3, 0, 0, 1, 1, 2, 0, 3, 1, 0, 2, 0, 2, 3, 0, 2,
        3, 1]))
```

Figure 9 Predicted with LR

Here wo show the accuracy of the SGD model with basic parameters in the next section we try to change the high parameters to find the best accuracy among these parameters.

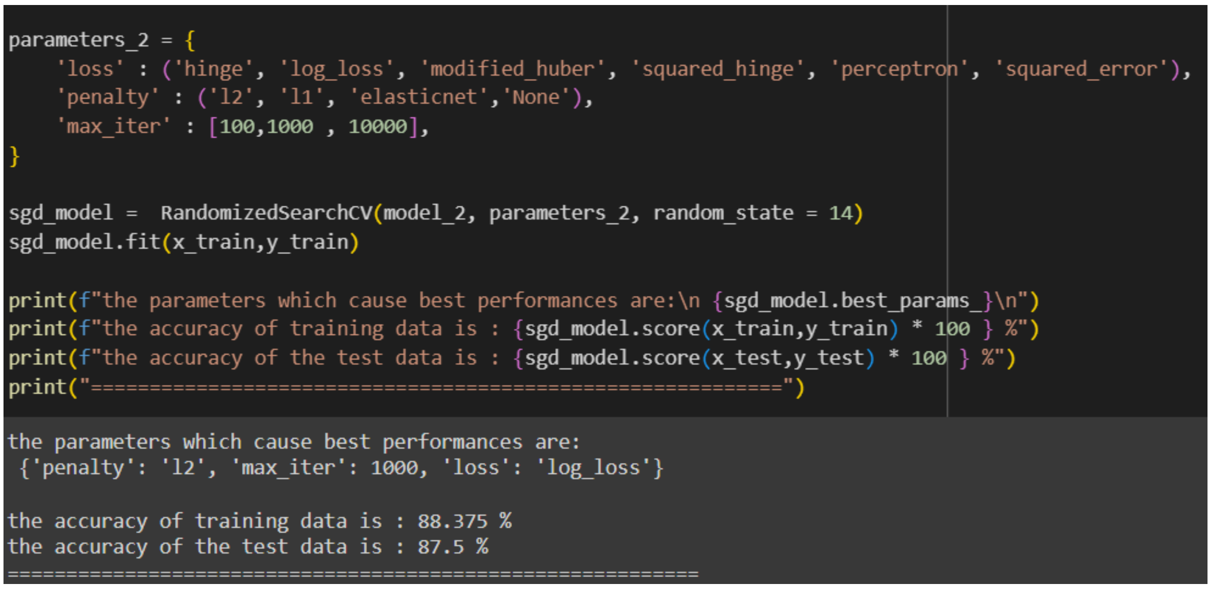


```
print(f"the accuracy of training data is : {model_2.score(x_train, y_train) * 100 } %")
print(f"the accuracy of the test data is : {model_2.score(x_test,y_test) * 100 } %")
```

```
the accuracy of training data is : 88.625 %
the accuracy of the test data is : 90.0 %
```

Figure 10 the accuracy of the SGD

Implementing the Random search to find the best high parameters for our desire. This section can be done with the Grid search method.



```
parameters_2 = {
    'loss' : ('hinge', 'log_loss', 'modified_huber', 'squared_hinge', 'perceptron', 'squared_error'),
    'penalty' : ('l2', 'l1', 'elasticnet','None'),
    'max_iter' : [100,1000 , 10000],
}

sgd_model =  RandomizedSearchCV(model_2, parameters_2, random_state = 14)
sgd_model.fit(x_train,y_train)

print(f"the parameters which cause best performances are:\n {sgd_model.best_params_}\n")
print(f"the accuracy of training data is : {sgd_model.score(x_train,y_train) * 100 } %")
print(f"the accuracy of the test data is : {sgd_model.score(x_test,y_test) * 100 } %")
print("==========================================================")
```

```
the parameters which cause best performances are:
 {'penalty': 'l2', 'max_iter': 1000, 'loss': 'log_loss'}

the accuracy of training data is : 88.375 %
the accuracy of the test data is : 87.5 %
==========================================================
```

Figure 11 high parameters of SGD

Therefore, the high parameters that make the accuracy 87.5 % is penalty: l2, max_iter:1000 , loss:log_loss.

We use two methods to find best high parameters for our desire. Grid search and Random search.

Here we try to find the decision boundary for LR and SGD method to classify data.

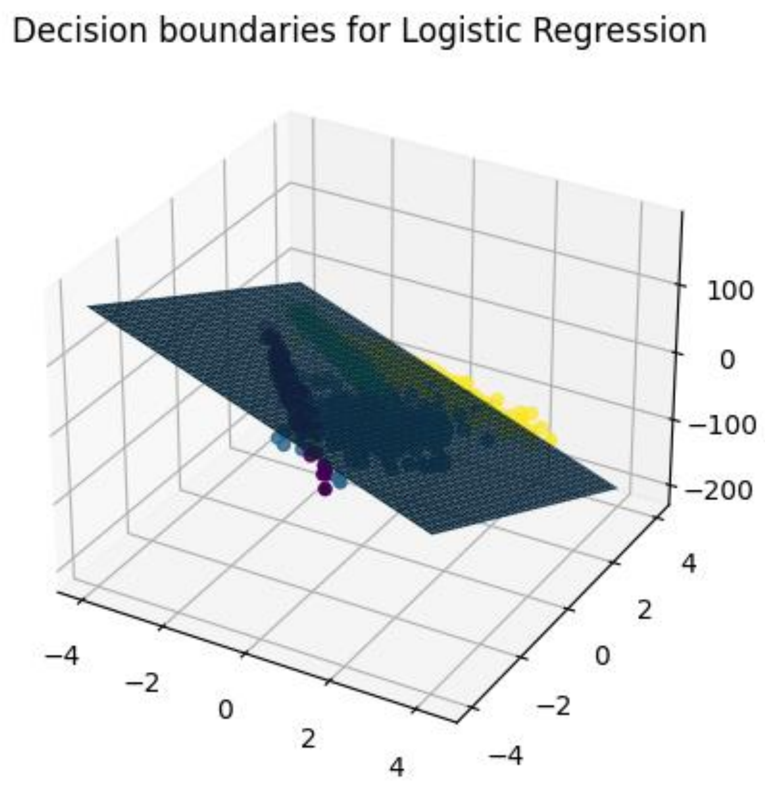


Figure 12 Decision boundary for LR

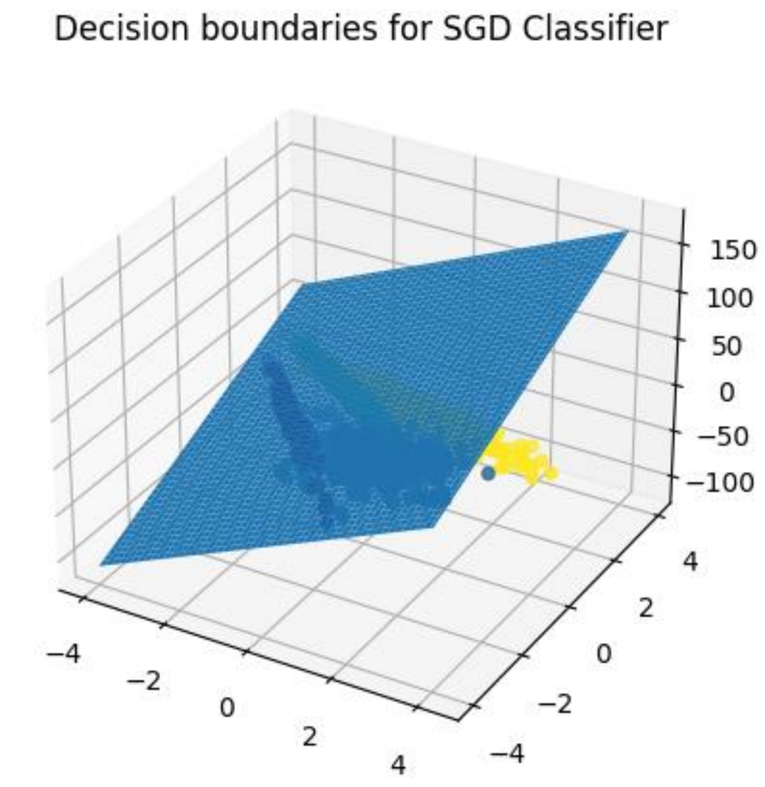


Figure 13Decision boundry for SGD

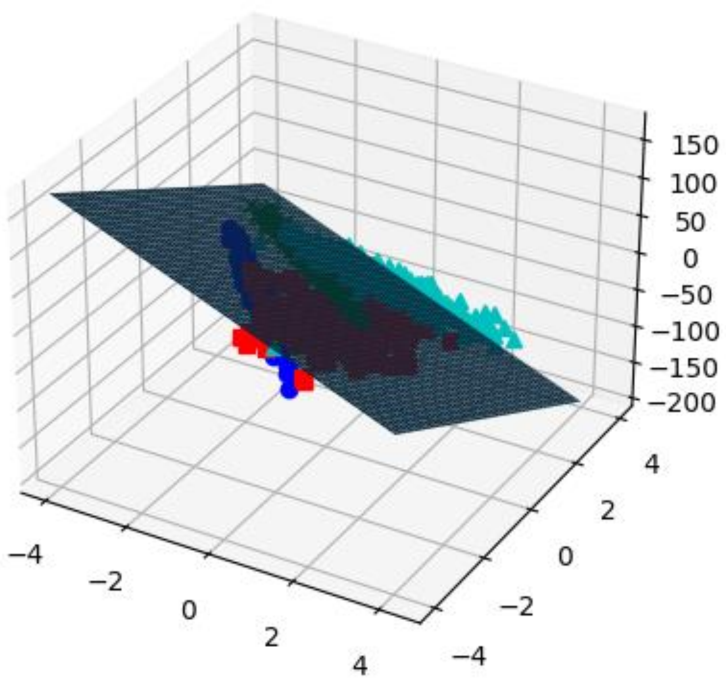


Figure 14 Decision boundry

Here we use Drawdata tool to make data set for our desire to make the data which is appropriate for our desire. We make the data set like this:

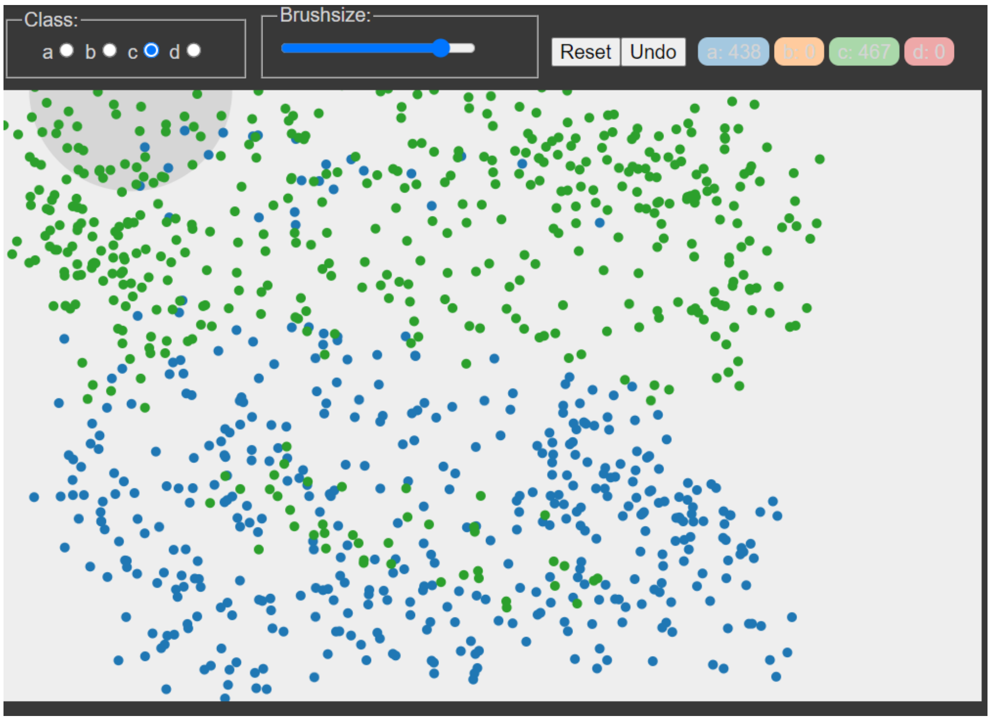
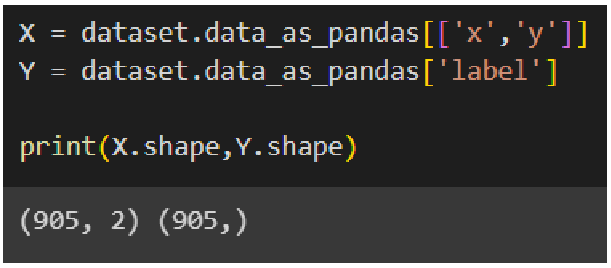


Figure 15 Dataset with Drawdata tool

Now we illustrate size of data that showed in the previous section.

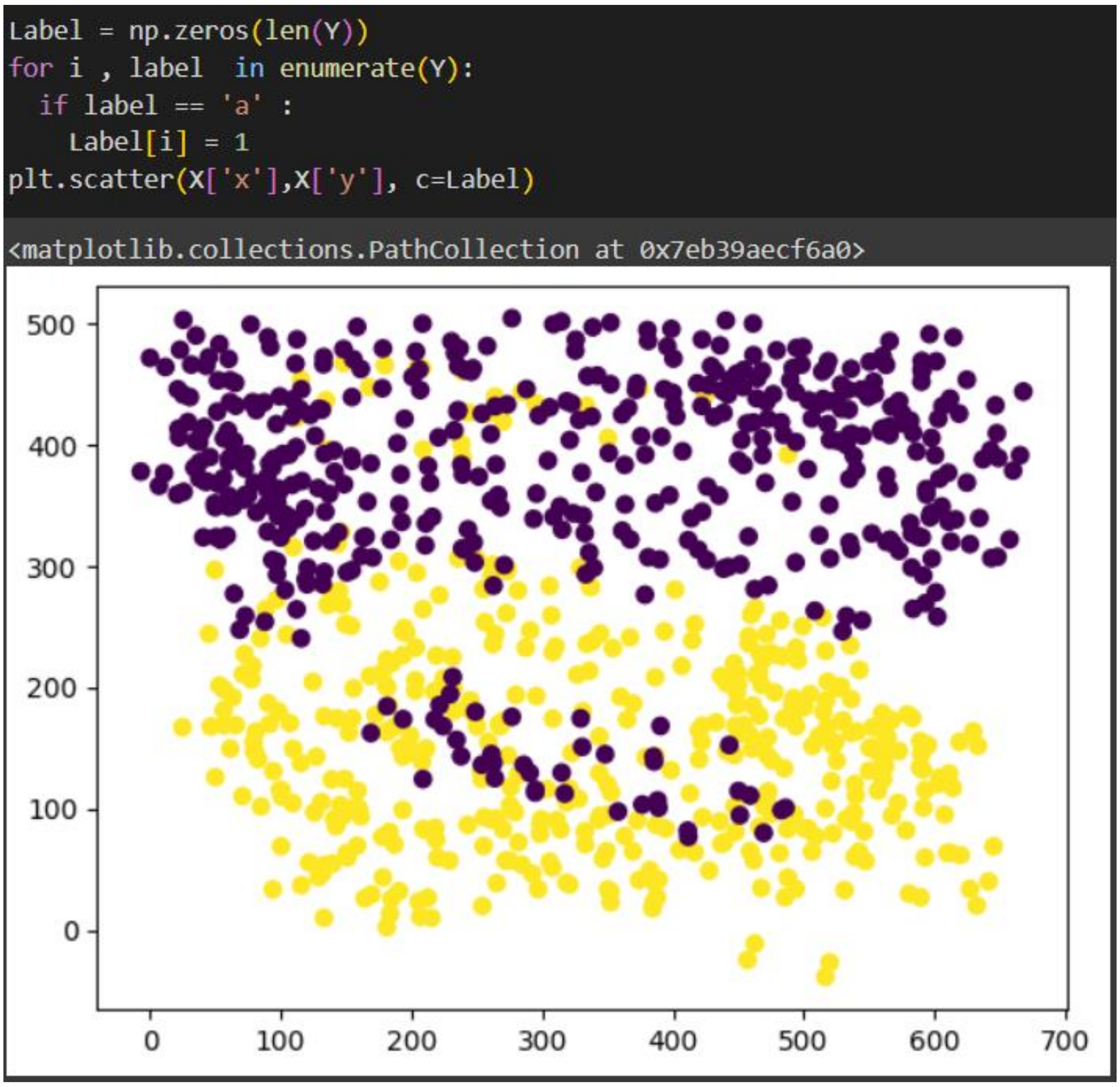


```
X = dataset.data_as_pandas[['x','y']]
Y = dataset.data_as_pandas['label']

print(X.shape,Y.shape)
```

```
(905, 2) (905,)
```

Figure 16 shape of Data

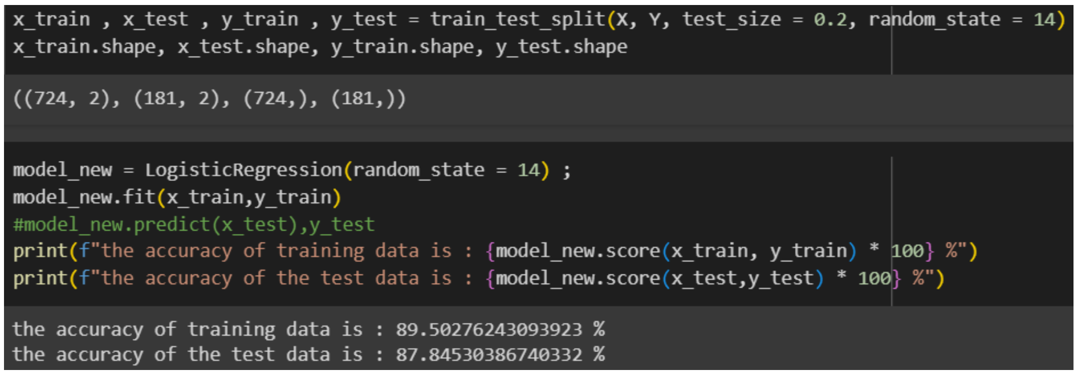
We allocate label to data set of the previous section. The dataset of the class A is 1 and class C is zero. Then we plot 2D of the data set.



```
Label = np.zeros(len(Y))
for i , label  in enumerate(Y):
  if label == 'a' :
    Label[i] = 1
plt.scatter(X['x'],X['y'], c=Label)
```

```
<matplotlib.collections.PathCollection at 0x7eb39aecf6a0>
```

Figure 17 2D plot of data set



```
x_train , x_test , y_train , y_test = train_test_split(X, Y, test_size = 0.2, random_state = 14)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

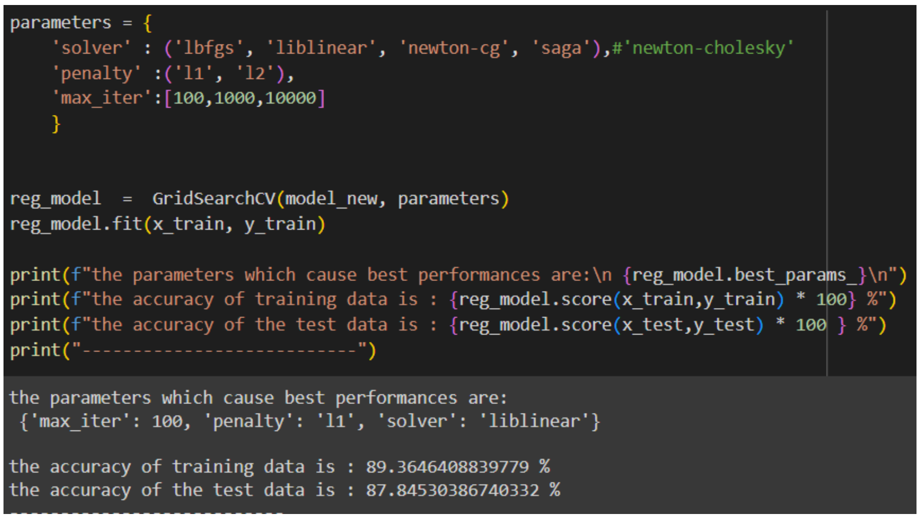
```
((724, 2), (181, 2), (724,), (181,))
```

```
model_new = LogisticRegression(random_state = 14) ;
model_new.fit(x_train,y_train)
#model_new.predict(x_test),y_test
print(f"the accuracy of training data is : {model_new.score(x_train, y_train) * 100} %")
print(f"the accuracy of the test data is : {model_new.score(x_test,y_test) * 100} %")
```

```
the accuracy of training data is : 89.50276243093923 %
the accuracy of the test data is : 87.84530386740332 %
```

Figure 18 spliting data to train and test

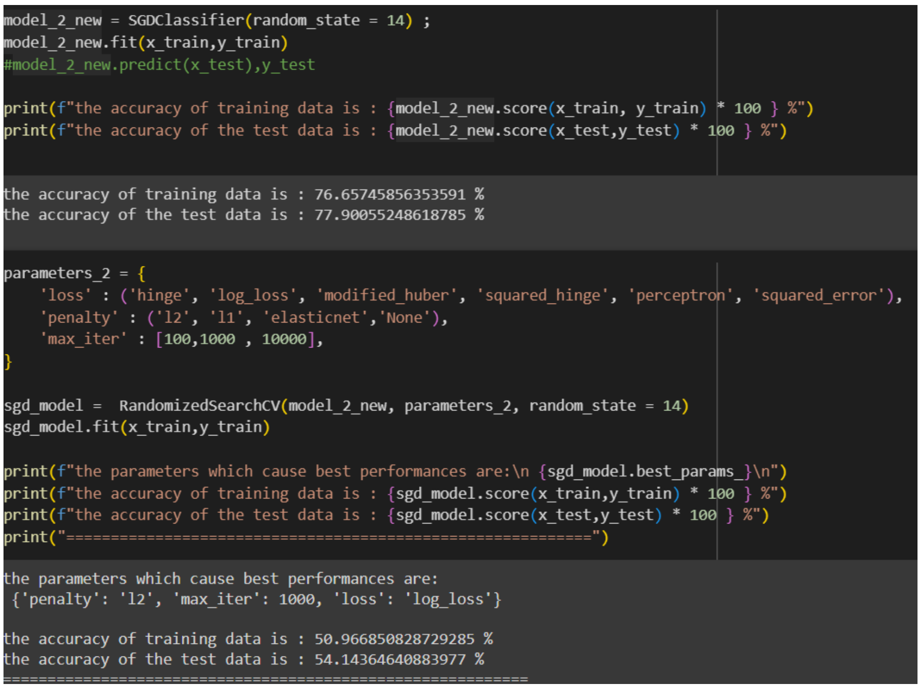
Give the parameters for finding the best parameters for classification.



```
parameters = {
    'solver' : ('lbfgs', 'liblinear', 'newton-cg', 'saga'),#'newton-cholesky'
    'penalty' :('l1', 'l2'),
    'max_iter':[100,1000,10000]
    }


reg_model  =  GridSearchCV(model_new, parameters)
reg_model.fit(x_train, y_train)

print(f"the parameters which cause best performances are:\n {reg_model.best_params_}\n")
print(f"the accuracy of training data is : {reg_model.score(x_train,y_train) * 100} %")
print(f"the accuracy of the test data is : {reg_model.score(x_test,y_test) * 100 } %")
print("--------------------------")
```

```
the parameters which cause best performances are:
 {'max_iter': 100, 'penalty': 'l1', 'solver': 'liblinear'}

the accuracy of training data is : 89.3646408839779 %
the accuracy of the test data is : 87.84530386740332 %
```

Figure 19 LR high parameters



```
model_2_new = SGDClassifier(random_state = 14) ;
model_2_new.fit(x_train,y_train)
#model_2_new.predict(x_test),y_test

print(f"the accuracy of training data is : {model_2_new.score(x_train, y_train) * 100 } %")
print(f"the accuracy of the test data is : {model_2_new.score(x_test,y_test) * 100 } %")
```

```
the accuracy of training data is : 76.65745856353591 %
the accuracy of the test data is : 77.90055248618785 %
```

```
parameters_2 = {
    'loss' : ('hinge', 'log_loss', 'modified_huber', 'squared_hinge', 'perceptron', 'squared_error'),
    'penalty' : ('l2', 'l1', 'elasticnet','None'),
    'max_iter' : [100,1000 , 10000],
}

sgd_model =  RandomizedSearchCV(model_2_new, parameters_2, random_state = 14)
sgd_model.fit(x_train,y_train)

print(f"the parameters which cause best performances are:\n {sgd_model.best_params_}\n")
print(f"the accuracy of training data is : {sgd_model.score(x_train,y_train) * 100 } %")
print(f"the accuracy of the test data is : {sgd_model.score(x_test,y_test) * 100 } %")
print("============================================================")
```

```
the parameters which cause best performances are:
 {'penalty': 'l2', 'max_iter': 1000, 'loss': 'log_loss'}

the accuracy of training data is : 50.966850828729285 %
the accuracy of the test data is : 54.14364640883977 %
============================================================
```

Figure 20 SGD high parameters

Here we implement two types of classification LR and SGD and plot the result . the result shows that the two classes are separated from each other correctly but the accuracy of the LR is higher than SGD and it can be seen in the two following pictures.

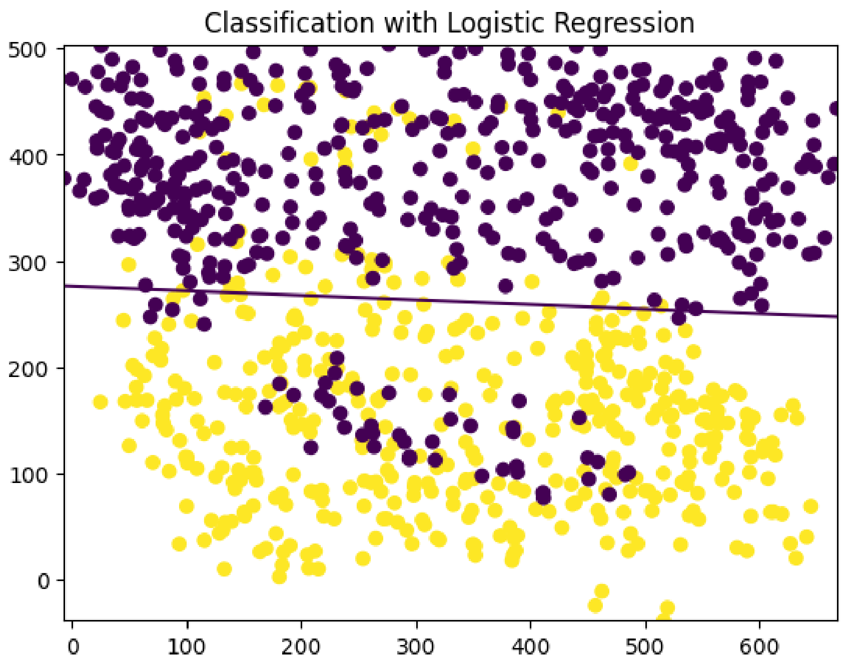


Figure 21 classification with LR

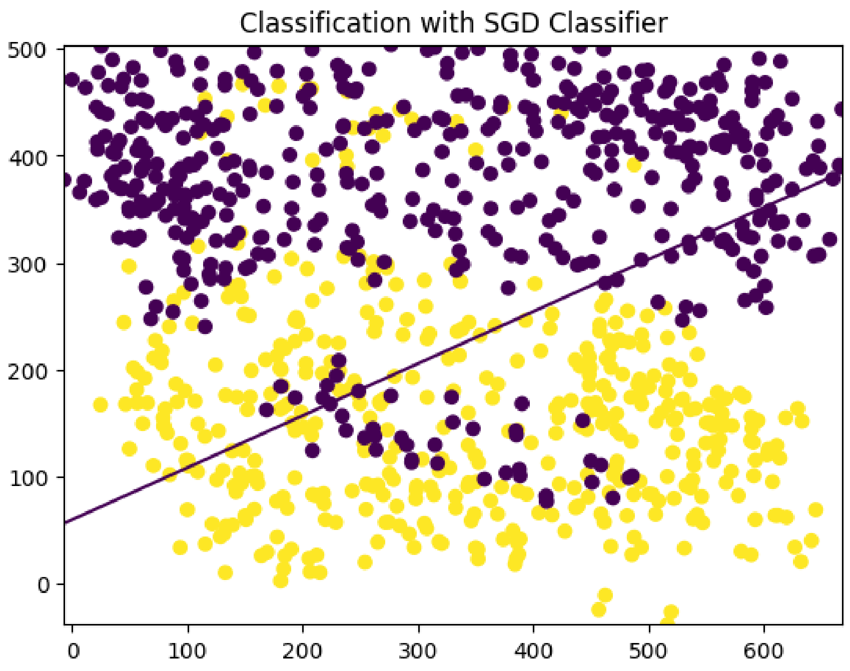


Figure 22 Classification with SGD

Question 2

In this question our desire is working on the dataset of the system which called CWRU Bearing. following information are getting from the site that the data set download from.

Data was collected for normal bearings, single-point drive end and fan end defects. Data was collected at 12,000 samples/second and at 48,000 samples/second for drive end bearing experiments. All fan end bearing data was collected at 12,000 samples/second.

Each file contains fan and drive end vibration data as well as motor rotational speed. For all files, the following item in the variable name indicates:

DE - drive end accelerometer data
FE - fan end accelerometer data
BA - base accelerometer data
time - time series data
RPM - rpm during testing

As mentioned the rate of the sampling is 12,000 and 48,000 so the data set is based on these values . the data is devided to four groups that the first group contain normal data whitout vibration the second and thirs groups of data are realated to the drive end bearing fault for 12k , 48k sample/second respectively . the last data is based on the fan_end bearing fault. Here we just work in the two groups of the data as it mentioned in the question Normal data and Drive end bearing fault data with rate of 12k. The following pictures shows the main system of the sample and different types of faults which could happen during the experiment.

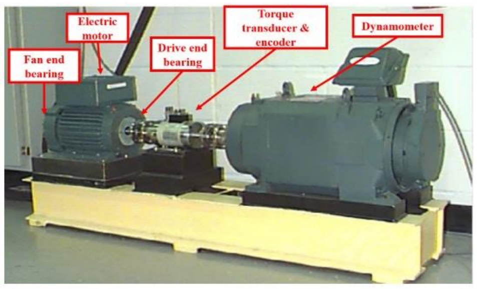


Figure 23 main system

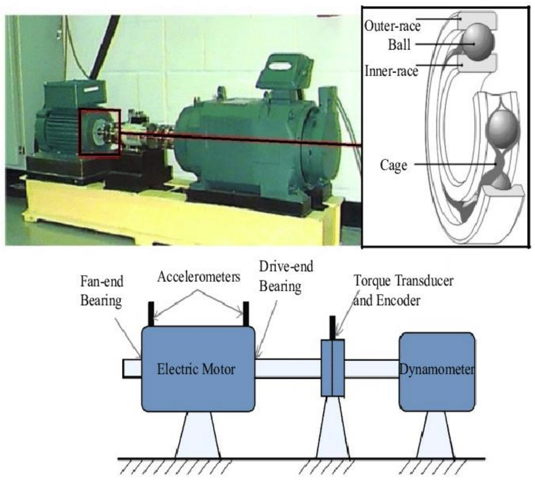
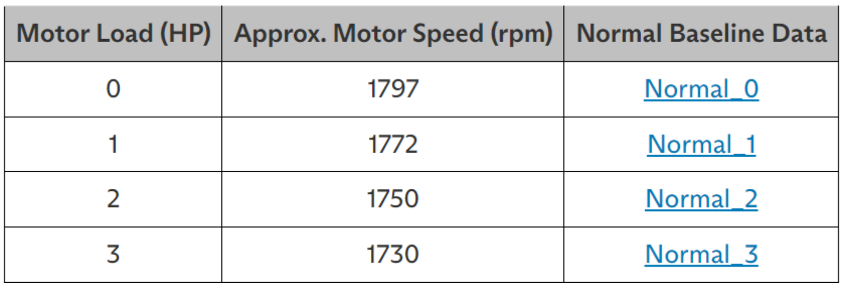


Figure 24modeled system

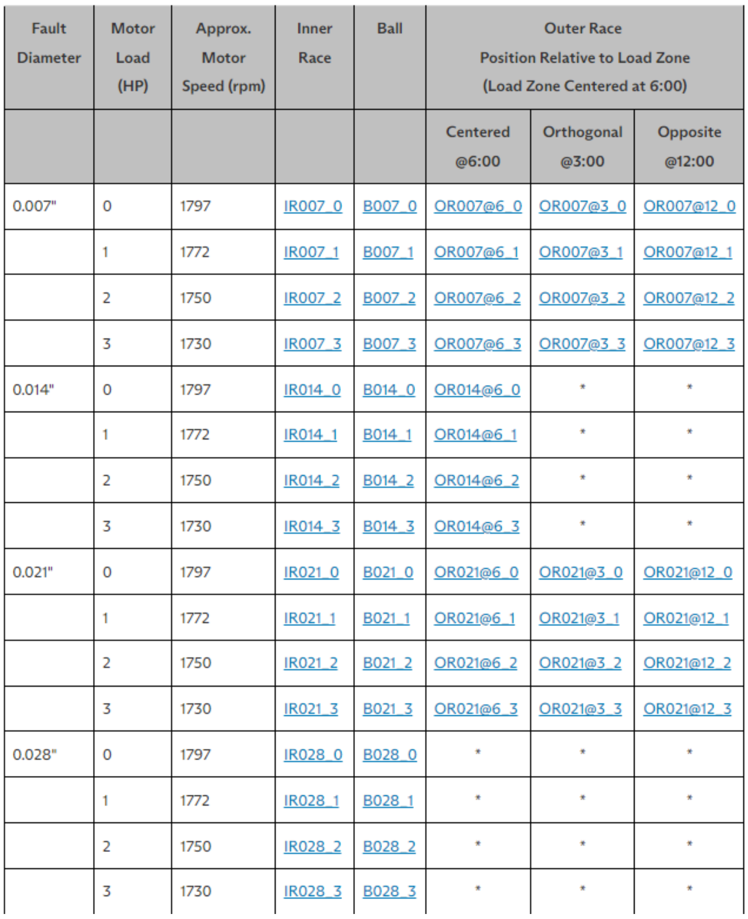
the page of the normal data shows the table as below that the groups are separated based on the Motor speed that can be 1797, 1772, 1750, 1730 rpm.



Motor Load (HP)	Approx. Motor Speed (rpm)	Normal Baseline Data
0	1797	Normal_0
1	1772	Normal_1
2	1750	Normal_2
3	1730	Normal_3

Figure 25 Noraml data table

The page of the fault data is :



Fault Diameter	Motor Load (HP)	Approx. Motor Speed (rpm)	Inner Race	Ball	Outer Race Position Relative to Load Zone (Load Zone Centered at 6:00)		
					Centered @6:00	Orthogonal @3:00	Opposite @12:00
0.007"	0	1797	IR007_0	B007_0	OR007@6_0	OR007@3_0	OR007@12_0
	1	1772	IR007_1	B007_1	OR007@6_1	OR007@3_1	OR007@12_1
	2	1750	IR007_2	B007_2	OR007@6_2	OR007@3_2	OR007@12_2
	3	1730	IR007_3	B007_3	OR007@6_3	OR007@3_3	OR007@12_3
0.014"	0	1797	IR014_0	B014_0	OR014@6_0	*	*
	1	1772	IR014_1	B014_1	OR014@6_1	*	*
	2	1750	IR014_2	B014_2	OR014@6_2	*	*
	3	1730	IR014_3	B014_3	OR014@6_3	*	*
0.021"	0	1797	IR021_0	B021_0	OR021@6_0	OR021@3_0	OR021@12_0
	1	1772	IR021_1	B021_1	OR021@6_1	OR021@3_1	OR021@12_1
	2	1750	IR021_2	B021_2	OR021@6_2	OR021@3_2	OR021@12_2
	3	1730	IR021_3	B021_3	OR021@6_3	OR021@3_3	OR021@12_3
0.028"	0	1797	IR028_0	B028_0	*	*	*
	1	1772	IR028_1	B028_1	*	*	*
	2	1750	IR028_2	B028_2	*	*	*
	3	1730	IR028_3	B028_3	*	*	*

Figure 26 Fault data table

The faults have many parameters: Fault diameter, Motor Load and Motor Speed. Each data are based on these parameters. More over the Inner race, Ball and Outer race section are different types of faults which will be shown below.

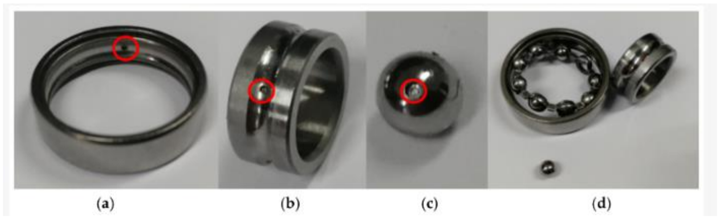
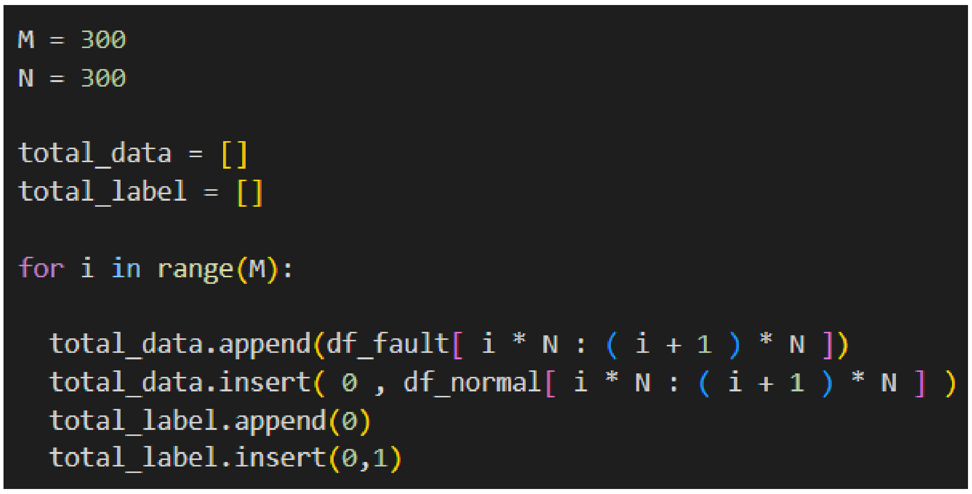


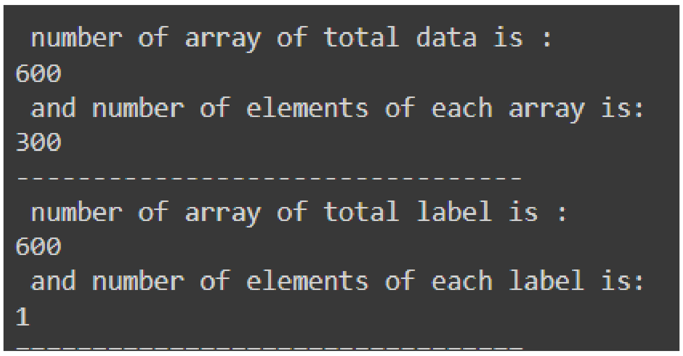
Figure 27 Different types of faults: a) outer race fault , b)ball fault , c) inner race fault , d) combination of parts

at first, we import data sets and shows the shape of each data. The normal and fault data are saved as df_normal and df_fault respectively. We want to makes M groups of data with length of N. our desire is to extract feature of these data and works with the features matrices. next, we add label to each class by assume the fault data as '0' class and Normal data as '1' class.



```
M = 300
N = 300

total_data = []
total_label = []

for i in range(M):

  total_data.append(df_fault[ i * N : ( i + 1 ) * N ])
  total_data.insert( 0 , df_normal[ i * N : ( i + 1 ) * N ] )
  total_label.append(0)
  total_label.insert(0,1)
```

Figure 28 Deviding data to groups



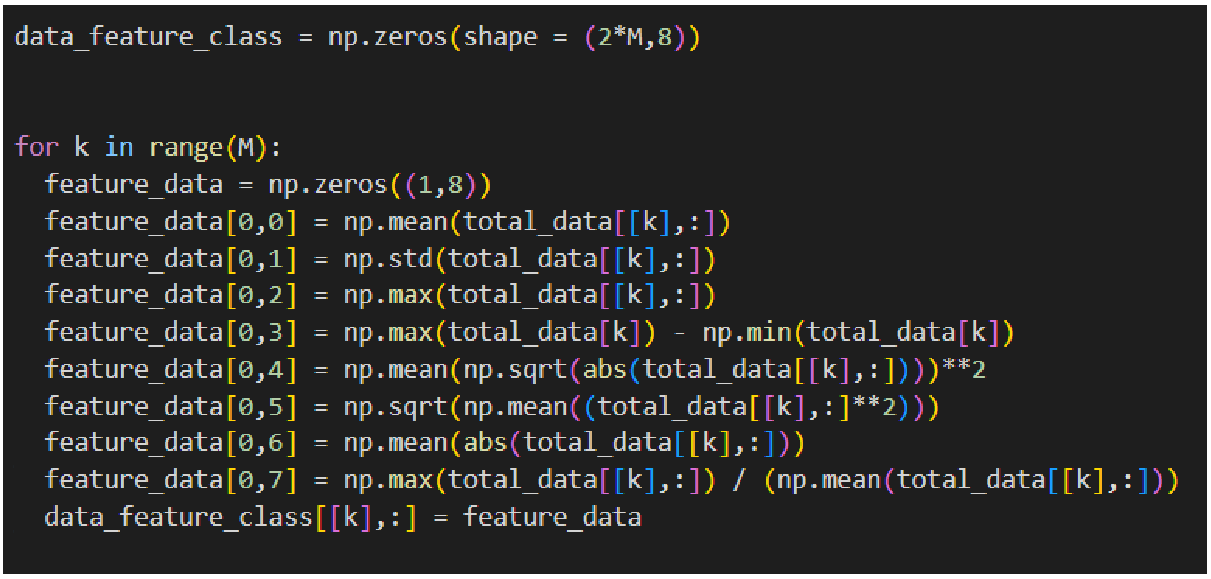
```
 number of array of total data is :
600
 and number of elements of each array is:
300
---------------------------------
 number of array of total label is :
600
 and number of elements of each label is:
1
---------------------------------
```

Figure 29 shape of the new data set

One of the main steps of the preprocessing is feature extraction. It means we focus on the main features of the data and working with them. more over the non-processed data is not appropriate for the Machin and it will not be accurate because it includes data which is so shuffled and without any specific pattern. So by using feature extraction we try to improve the performance od the machine and system.

By extracting main features, the dimension of the data will be decreased and it means the required space for data and calculating decline and the run time of the code increase. Other than that, working on the features which have the main effect on the classification or other methods, make the machine more accurate and increase the accuracy of the main system.

As we make a matrix of data which include m groups of data that each one length is N , here we want to work with the feature of each groups instead of working with non-processed data . so we collect eight method of statistic to extract feature of each group and the feature matrix will be 2M * 8 .

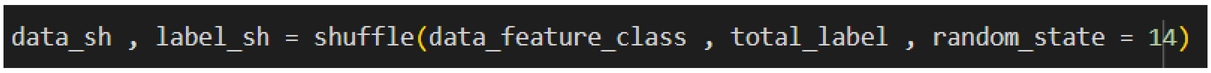


```
data_feature_class = np.zeros(shape = (2*M,8))


for k in range(M):
  feature_data = np.zeros((1,8))
  feature_data[0,0] = np.mean(total_data[[k],:])
  feature_data[0,1] = np.std(total_data[[k],:])
  feature_data[0,2] = np.max(total_data[[k],:])
  feature_data[0,3] = np.max(total_data[k]) - np.min(total_data[k])
  feature_data[0,4] = np.mean(np.sqrt(abs(total_data[[k],:])))**2
  feature_data[0,5] = np.sqrt(np.mean((total_data[[k],:]**2)))
  feature_data[0,6] = np.mean(abs(total_data[[k],:]))
  feature_data[0,7] = np.max(total_data[[k],:]) / (np.mean(total_data[[k],:]))
  data_feature_class[[k],:] = feature_data
```

Figure 30 Different methods of feature extraction

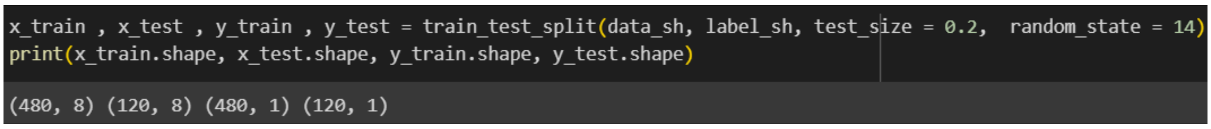
One of the most common methods to make the model more accurate is shuffling data set. for example, here we have an array of data that the normal data are at the first section of label array and the last section is allocated to fault data. So, at the first glance the best function to model this data is sigmoid. Whereas it is not globally correct and it is not practical so we try to prevent the system from these states by shuffling the data set. The main effect of the shuffling is to preventing the system from learning wrong pattern. The next one is helping the system to find the main pattern of the system and cancel the over-fitting states. Increasing the accuracy of the model by decrease the bias value of the modeling.

Before extracting data to two groups for train and test we shuffle the data set and then allocate 80 percent of the data to the train data and get remaining data as the test data to assess the model.



```
data_sh , label_sh = shuffle(data_feature_class , total_label , random_state = 14)
```

Figure 31 Shuffeling data



```
x_train , x_test , y_train , y_test = train_test_split(data_sh, label_sh, test_size = 0.2,  random_state = 14)
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)

(480, 8) (120, 8) (480, 1) (120, 1)
```

Figure 32 Spliting data to train and test after shuffeling

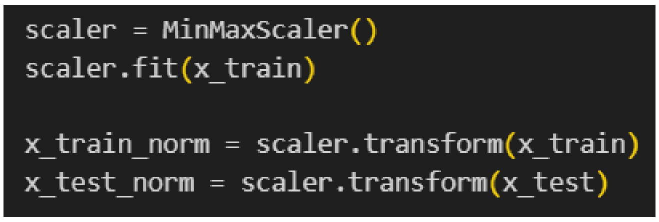
Another method for preprocessing data is normalizing. Normalizing is used to make the wight of each data equal as others for instance one of the features is varies from 1000 to 2000 and another features changes from 0.1 to 1. It is clear that the first feature is more effective than the later one so it makes the model impractical and useless. So we should normalized each data to make effectiveness of each feature as equal as others. Thus, as we mentioned normalizing prevent us from ignoring some features and increase the accuracy of the model. It is noticeable that we should normalize the train and test data. The main point which should considered is that the test data must be normalized as same as train data with same method and parameters otherwise the model is not enough accurate and the result is not guaranteed to be correct. Two main method of the normalizing is Min_Max and Standardization methods that are used in this section.

Min_Max method makes all variables from 0 to 1 by using the following equation:

$$X_{norm} = \frac{(x - x_{\min})}{x_{\max} - x_{\min}}$$

Standardization method makes the average of the data zero and the standard deviation of the data equal to 1. The equation using for this method is:

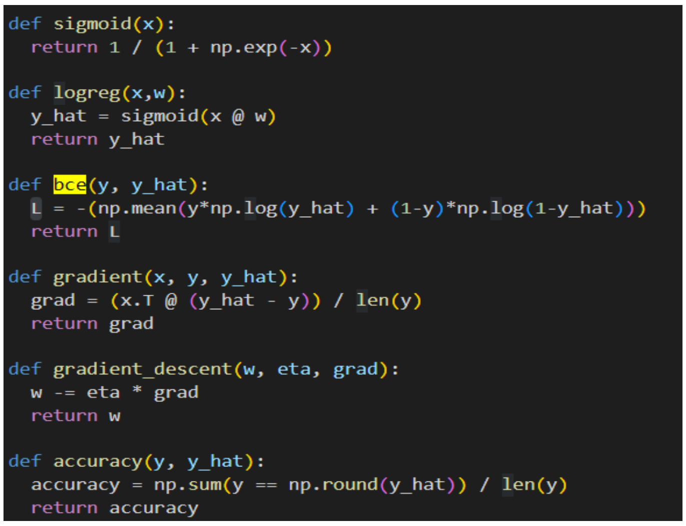
$$x_{norm} = \frac{x - \bar{x}}{\sigma}$$



```
scaler = MinMaxScaler()
scaler.fit(x_train)

x_train_norm = scaler.transform(x_train)
x_test_norm = scaler.transform(x_test)
```

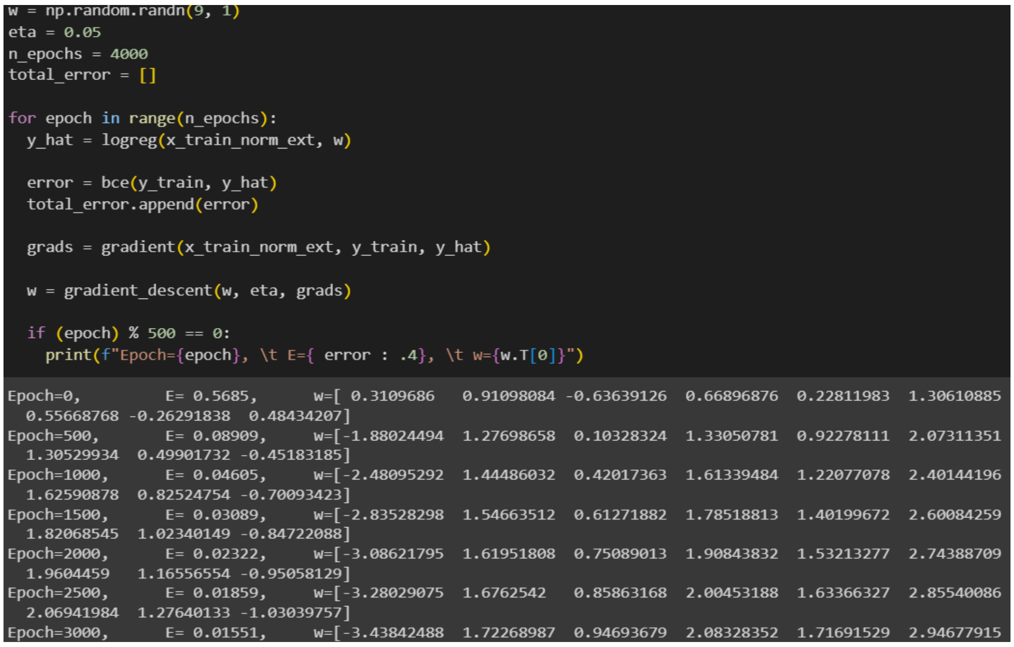
Figure 33Minmax code

In this section we should design Classification model, loss function and learning and evaluation algorithm without using ready to use function of python. We define gradient descent to update weight of the model after each iteration. Then we consider sigmoid function as the model of the function to predict the label of the variables. Using BCE (Binary Cross-Entropy) for the loss function.



```
def sigmoid(x):
  return 1 / (1 + np.exp(-x))

def logreg(x,w):
  y_hat = sigmoid(x @ w)
  return y_hat

def bce(y, y_hat):
  L = -(np.mean(y*np.log(y_hat) + (1-y)*np.log(1-y_hat)))
  return L

def gradient(x, y, y_hat):
  grad = (x.T @ (y_hat - y)) / len(y)
  return grad

def gradient_descent(w, eta, grad):
  w -= eta * grad
  return w

def accuracy(y, y_hat):
  accuracy = np.sum(y == np.round(y_hat)) / len(y)
  return accuracy
```

Figure 34 Coding og Gradient decent method



```
w = np.random.randn(9, 1)
eta = 0.05
n_epochs = 4000
total_error = []

for epoch in range(n_epochs):
  y_hat = logreg(x_train_norm_ext, w)

  error = bce(y_train, y_hat)
  total_error.append(error)

  grads = gradient(x_train_norm_ext, y_train, y_hat)

  w = gradient_descent(w, eta, grads)

  if (epoch) % 500 == 0:
    print(f"Epoch={epoch}, \t E={ error : .4}, \t w={w.T[0]}")
```

```
Epoch=0,         E= 0.5685,      w=[ 0.3109686   0.91098084 -0.63639126  0.66896876  0.22811983  1.30610885
  0.55668768 -0.26291838  0.48434207]
Epoch=500,       E= 0.08909,     w=[-1.88024494  1.27698658  0.10328324  1.33050781  0.92278111  2.07311351
  1.30529934  0.49901732 -0.45183185]
Epoch=1000,      E= 0.04605,     w=[-2.48095292  1.44486032  0.42017363  1.61339484  1.22077078  2.40144196
  1.62590878  0.82524754 -0.70093423]
Epoch=1500,      E= 0.03089,     w=[-2.83528298  1.54663512  0.61271882  1.78518813  1.40199672  2.60084259
  1.82068545  1.02340149 -0.84722088]
Epoch=2000,      E= 0.02322,     w=[-3.08621795  1.61951808  0.75089013  1.90843832  1.53213277  2.74388709
  1.9604459   1.16556554 -0.95058129]
Epoch=2500,      E= 0.01859,     w=[-3.28029075  1.6762542   0.85863168  2.00453188  1.63366327  2.85540086
  2.06941984  1.27640133 -1.03039757]
Epoch=3000,      E= 0.01551,     w=[-3.43842488  1.72268987  0.94693679  2.08328352  1.71691529  2.94677915
```

Figure 35 Implimenting the Gradient descent

The diagram of the error based on the epochs of the system.

```
final error is 0.011651527148260381
```

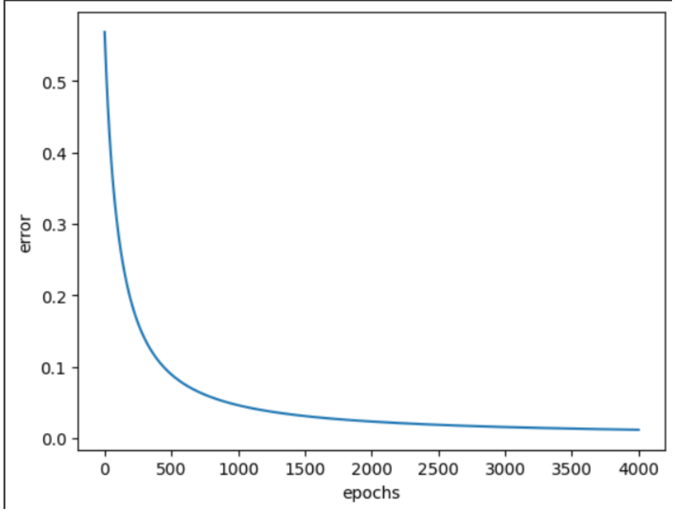
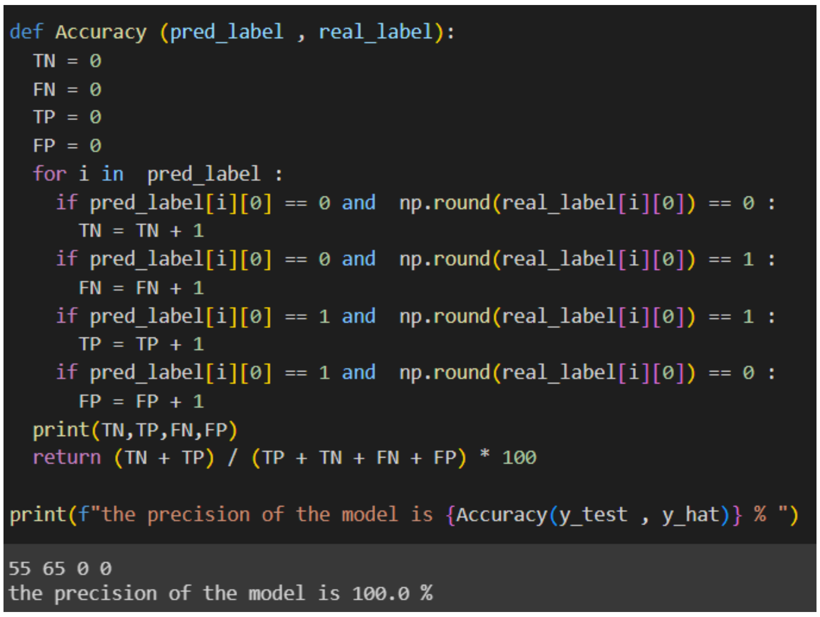


Figure 36 Error diagram

At the first glance it is clear that the pattern of the model perform well as the error decrease as the iteration gets bigger. So, if the number of the iteration be big enough the error will be close to zero. But this diagram is based on the train value and we are not sure about the performance of the model with test data. If our model is so accurate with train data it is probably appropriate for test data but is not globally true. Some times the model is over-fitted and is too close to the real value of the train data more over the it seems that the model learn noise and all the values and not the total pattern of the model to classify data so if one value will be added to the model, machine will not be able to classify that value and it cause main change in the model and cause huge error. So, we should test our model with the test data to make final decision and say the accuracy of the system. there are some methods to find total pattern of the data not completely specific pattern of the model. Using regularization method and using iteration which is not to big that cause overfitting and too short that cause under-fitting.

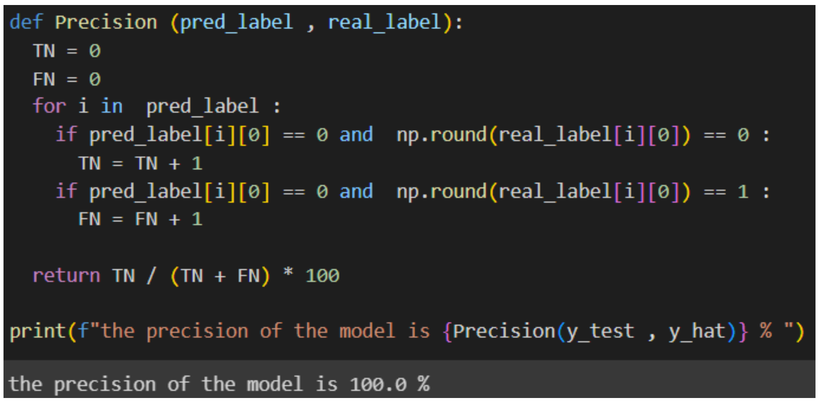
To assess the model we use the assessment method accuracy and precision. the accuracy tells the percentage of the values which are labeled correctly. And the precision is the percentage of the correct labeled values in one class. The code of these assessment methods is coming below:



```
def Accuracy (pred_label , real_label):
  TN = 0
  FN = 0
  TP = 0
  FP = 0
  for i in  pred_label :
    if pred_label[i][0] == 0 and  np.round(real_label[i][0]) == 0 :
      TN = TN + 1
    if pred_label[i][0] == 0 and  np.round(real_label[i][0]) == 1 :
      FN = FN + 1
    if pred_label[i][0] == 1 and  np.round(real_label[i][0]) == 1 :
      TP = TP + 1
    if pred_label[i][0] == 1 and  np.round(real_label[i][0]) == 0 :
      FP = FP + 1
  print(TN,TP,FN,FP)
  return (TN + TP) / (TP + TN + FN + FP) * 100

print(f"the precision of the model is {Accuracy(y_test , y_hat)} % ")
```

```
55 65 0 0
the precision of the model is 100.0 %
```

Figure 37 Accuracy definition

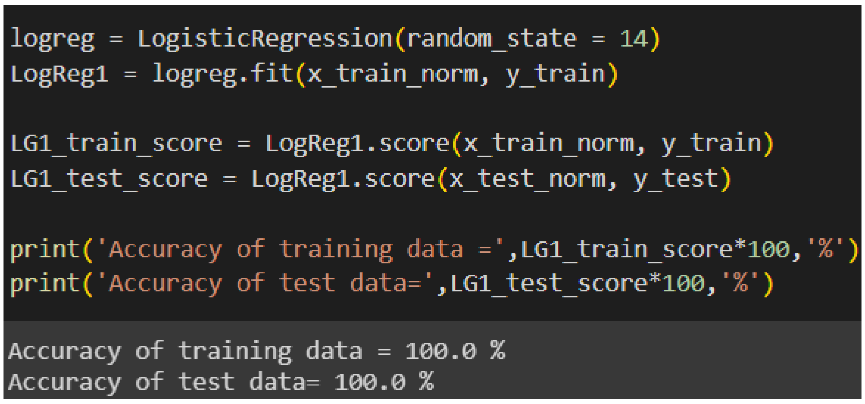


```
def Precision (pred_label , real_label):
  TN = 0
  FN = 0
  for i in  pred_label :
    if pred_label[i][0] == 0 and  np.round(real_label[i][0]) == 0 :
      TN = TN + 1
    if pred_label[i][0] == 0 and  np.round(real_label[i][0]) == 1 :
      FN = FN + 1

  return TN / (TN + FN) * 100

print(f"the precision of the model is {Precision(y_test , y_hat)} % ")
```

```
the precision of the model is 100.0 %
```

Figure 38Precision definition

In this section we used ready to used function of python. As below the code and the result of the code:

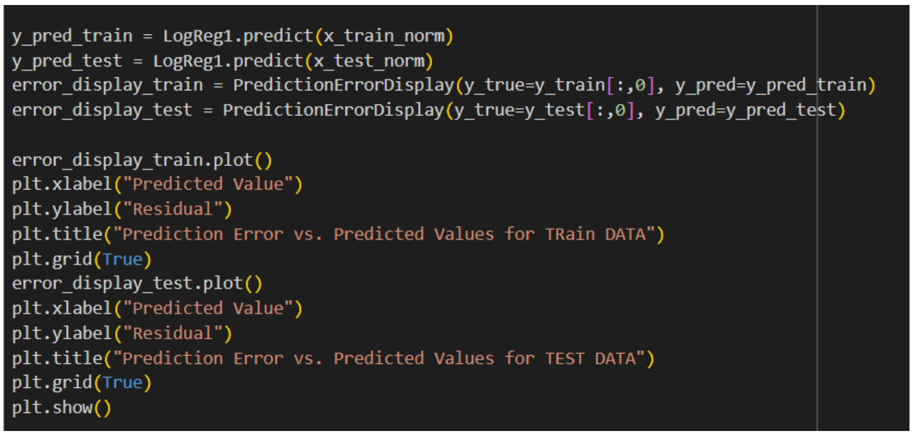


```
logreg = LogisticRegression(random_state = 14)
LogReg1 = logreg.fit(x_train_norm, y_train)

LG1_train_score = LogReg1.score(x_train_norm, y_train)
LG1_test_score = LogReg1.score(x_test_norm, y_test)

print('Accuracy of training data =',LG1_train_score*100,'%')
print('Accuracy of test data=',LG1_test_score*100,'%')
```

```
Accuracy of training data = 100.0 %
Accuracy of test data= 100.0 %
```

Figure 39 accuracy of ther code

To show the loss function we code as following and the result is:



```
y_pred_train = LogReg1.predict(x_train_norm)
y_pred_test = LogReg1.predict(x_test_norm)
error_display_train = PredictionErrorDisplay(y_true=y_train[:,0], y_pred=y_pred_train)
error_display_test = PredictionErrorDisplay(y_true=y_test[:,0], y_pred=y_pred_test)

error_display_train.plot()
plt.xlabel("Predicted Value")
plt.ylabel("Residual")
plt.title("Prediction Error vs. Predicted Values for TRain DATA")
plt.grid(True)
error_display_test.plot()
plt.xlabel("Predicted Value")
plt.ylabel("Residual")
plt.title("Prediction Error vs. Predicted Values for TEST DATA")
plt.grid(True)
plt.show()
```

Figure 40 Loss function diagram

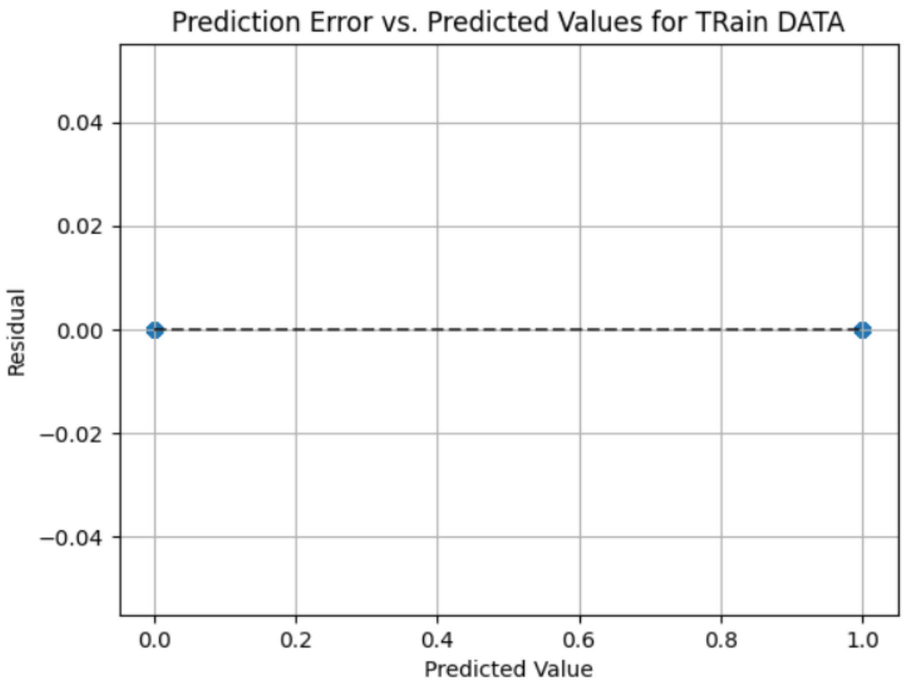


Figure 41 Loss function diagram of the train data

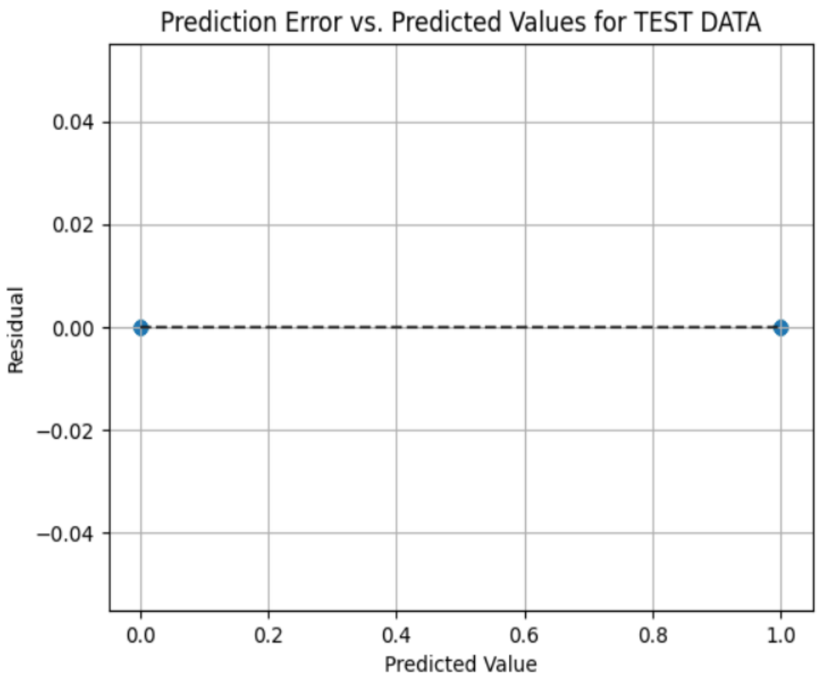
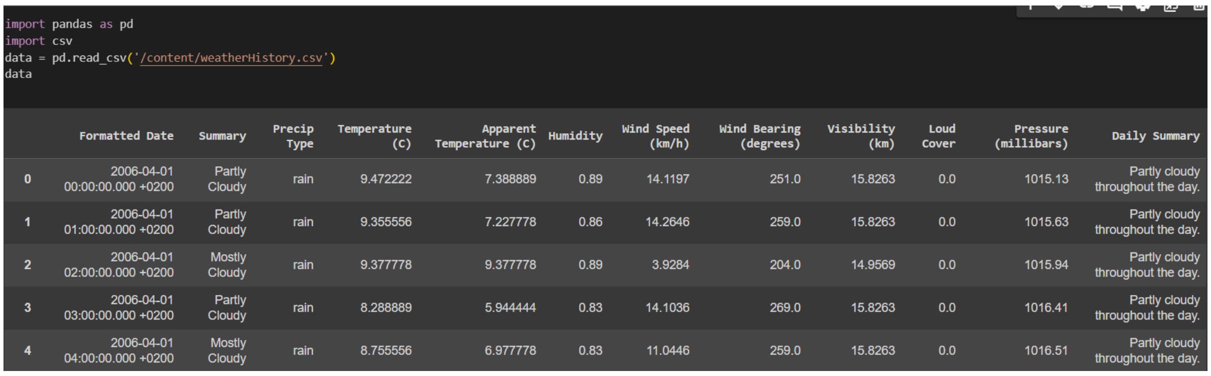


Figure 42 Loss function of the Test data

Question 3

The data set of this question is the weather condition of a country from 2006 to 2016. The data includes different classes of data. Humidity , wind bearing , loud cover , pressure and daily summary and , … . it can be understood that the class of the loud cover is zero for all data and we can delete this column from the data set to decrese the size of the data. More over we try to check each class and number of elements.

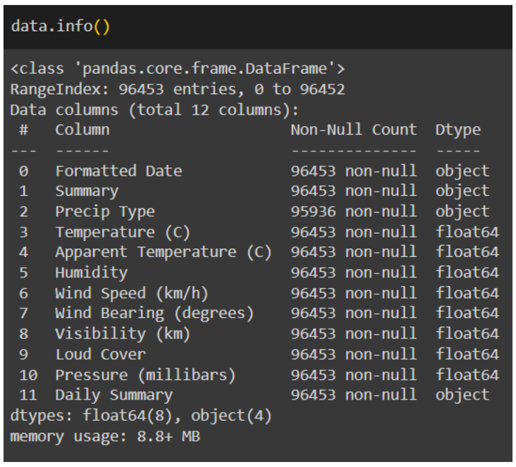


```
import pandas as pd
import csv
data = pd.read_csv('/content/weatherHistory.csv')
data
```

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Daily Summary
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	Partly cloudy throughout the day.
1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63	Partly cloudy throughout the day.
2	2006-04-01 02:00:00.000 +0200	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94	Partly cloudy throughout the day.
3	2006-04-01 03:00:00.000 +0200	Partly Cloudy	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41	Partly cloudy throughout the day.
4	2006-04-01 04:00:00.000 +0200	Mostly Cloudy	rain	8.755556	6.977778	0.83	11.0446	259.0	15.8263	0.0	1016.51	Partly cloudy throughout the day.

Figure 43 Data frame

By using the following code we can see all the classes and check the nimber of elements of each class. It can be seen that the number of the precip type is not equal to other clases so we need to check the data to figure out the reason.

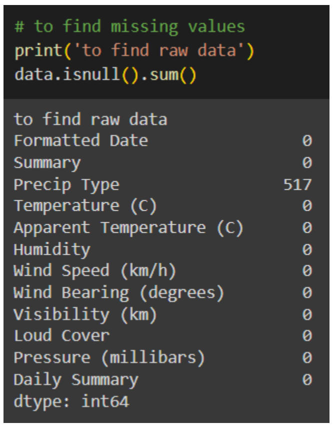


```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96453 entries, 0 to 96452
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Formatted Date            96453 non-null  object 
 1   Summary                   96453 non-null  object 
 2   Precip Type               95936 non-null  object 
 3   Temperature (C)           96453 non-null  float64
 4   Apparent Temperature (C)  96453 non-null  float64
 5   Humidity                  96453 non-null  float64
 6   Wind Speed (km/h)         96453 non-null  float64
 7   Wind Bearing (degrees)    96453 non-null  float64
 8   Visibility (km)           96453 non-null  float64
 9   Loud Cover                96453 non-null  float64
 10  Pressure (millibars)      96453 non-null  float64
 11  Daily Summary             96453 non-null  object 
dtypes: float64(8), object(4)
memory usage: 8.8+ MB
```

Figure 44 Total information of Data

Folloing code shows the reason for the difference of the number of elements and also it can be understood from previous part as it writed non_null values so the differences are null values.

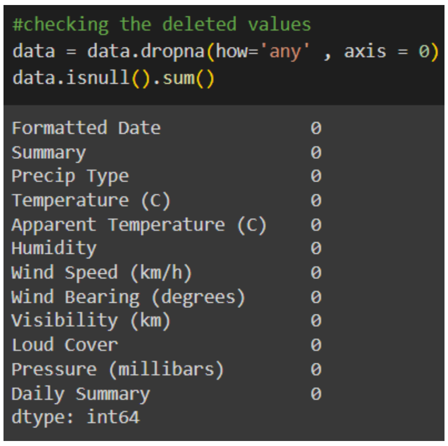


```
# to find missing values
print('to find raw data')
data.isnull().sum()
```

```
to find raw data
Formatted Date                0
Summary                       0
Precip Type                 517
Temperature (C)               0
Apparent Temperature (C)      0
Humidity                      0
Wind Speed (km/h)             0
Wind Bearing (degrees)        0
Visibility (km)               0
Loud Cover                    0
Pressure (millibars)          0
Daily Summary                 0
dtype: int64
```

Figure 45 Null values in data

So we delete the rows which include null data in them . it is noticeable that eliminated this data is correct as the number of the null elements is short enough to be ignored rather than other data.



```
#checking the deleted values
data = data.dropna(how='any' , axis = 0)
data.isnull().sum()
```

```
Formatted Date              0
Summary                     0
Precip Type                 0
Temperature (C)             0
Apparent Temperature (C)    0
Humidity                    0
Wind Speed (km/h)           0
Wind Bearing (degrees)      0
Visibility (km)             0
Loud Cover                  0
Pressure (millibars)        0
Daily Summary               0
dtype: int64
```

Figure 46 Removing null values

The desire of this section is to Drawing histogram and heat map of correlation matrix.

The histogram shows each class and its distribution such and basic information such as density, average , variance and so on. One the disadvantages of the histogram is it doesn’t show the correlation between two variable and check each parameter and value lonely.

Histogram of tempreture:

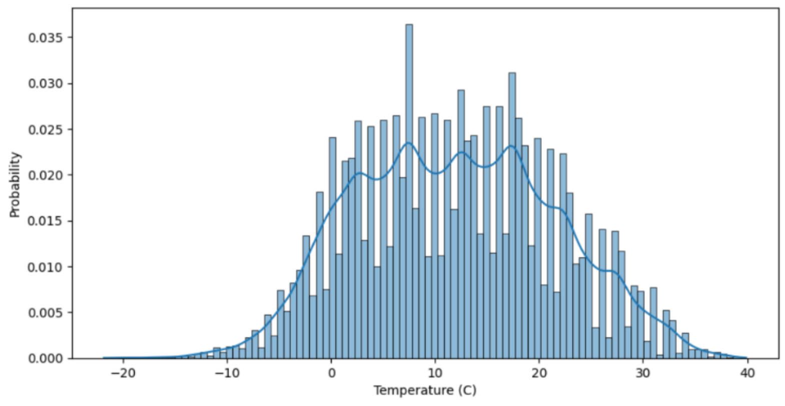


Figure 47 Histogram of the Temperature

In general, the data related to temperature measurement have a normal distribution with a mean of 10. Also, the variance of the data shows that in most days, the air temperature It has been between 0 and 20 degrees and temperatures outside this range have been experienced less.

Histogram of apparent tempreture:

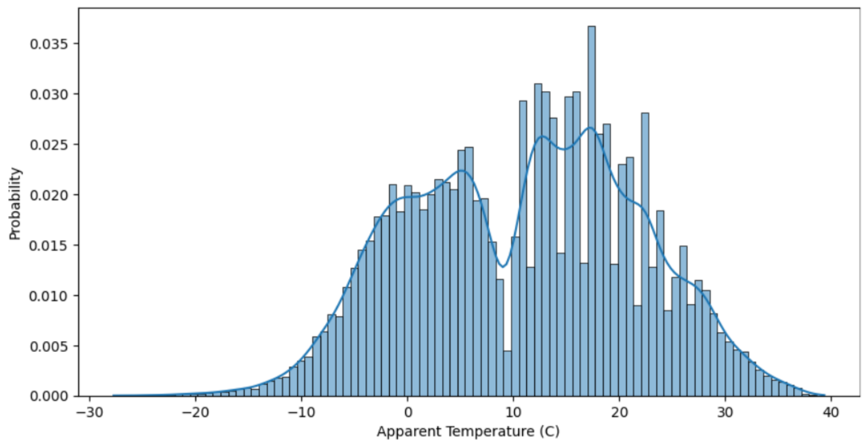


Figure 48 Histogram of the Apparent Temperature

the data related to temperature measurement have a normal distribution and changes from -30 to 40.

Histogram of Humidity:

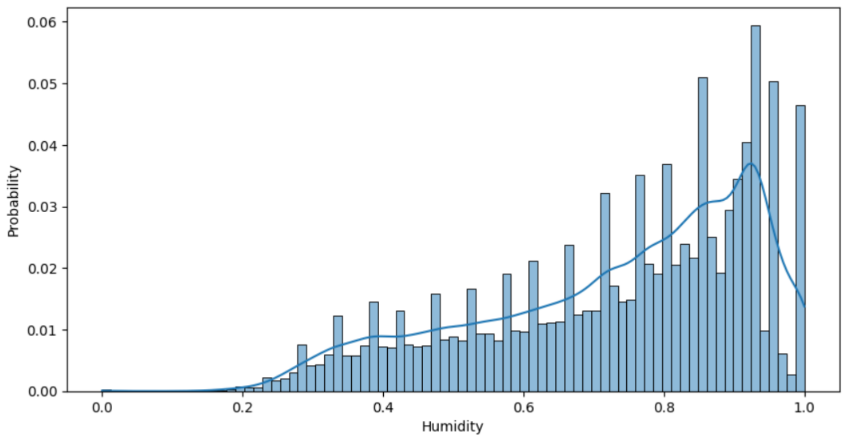


Figure 49 Histogram of the Humidity

the data related to temperature measurement do not have a normal distribution and the probability continuously increase as the humidity goes up.

Histogram of visibility:

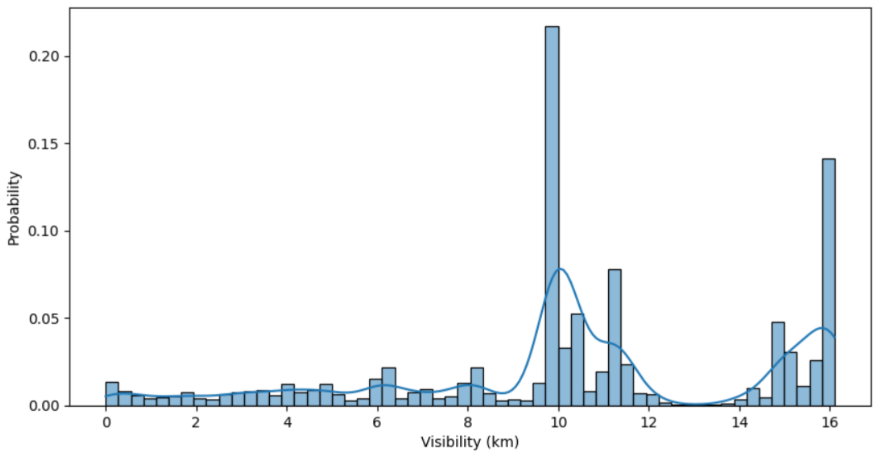


Figure 50 Histogram of Visibility

the data related to temperature measurement have any specific distribution. It change from 0 to 16 .

now we plot the histogram :

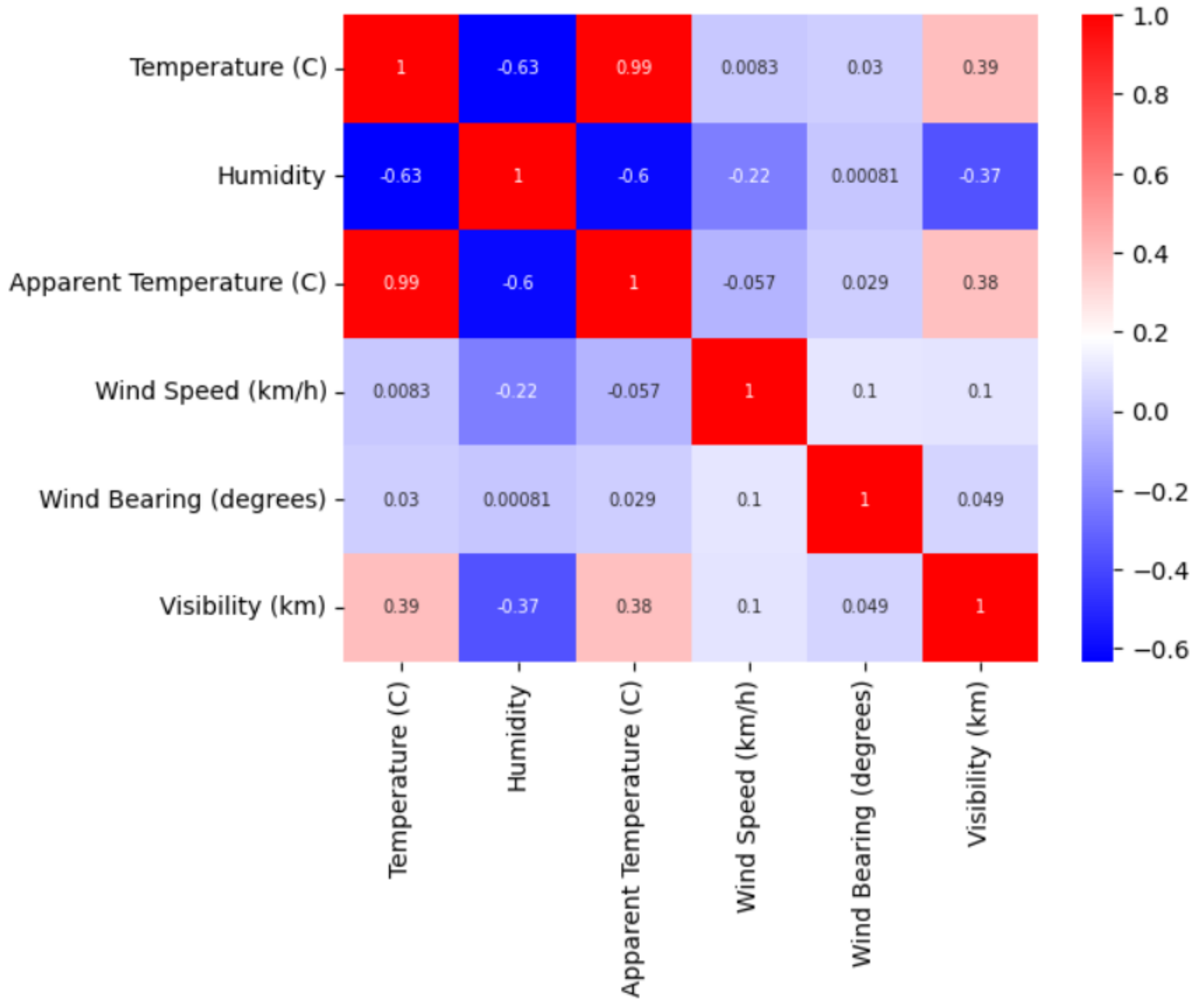
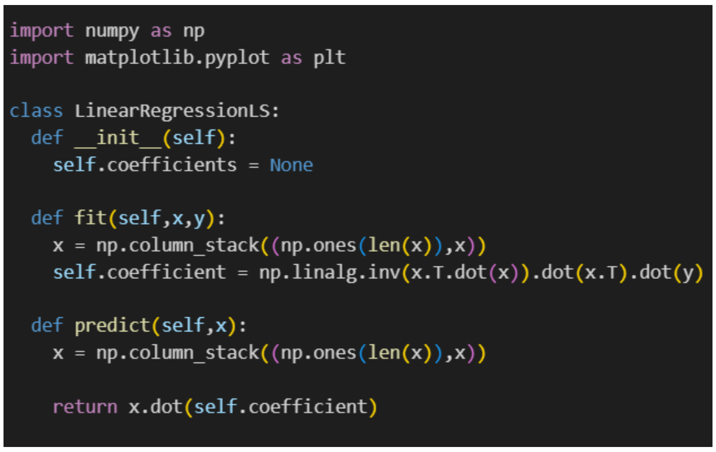


Figure 51 Heat map Diagram

The heatmap shows correlation between each two classes and let us to choose classes which have more effects o each other. In each heatmap diagram there are some basic points. In each cell, the degree of correlation between two variables is shown. The closer this number is to 1, it means that their correlation is positive and changes in a variable leads to changes in another variable in the same direction. The closer this number is to -1, it means that these two variables are opposite and so are their changes It is opposite to each other. Also, the number 1 is observed on the main diameter of this matrix, which is normal. Because each variable is most similar to itself. Thus, for this case, the Humidity, Temperature and apparent temperature are the variables which have more effect on each other so we use these classes. Moreover, apparent temperature and temperature have same direction so we prefer to predict temperature and apparent temperature by Humidity.

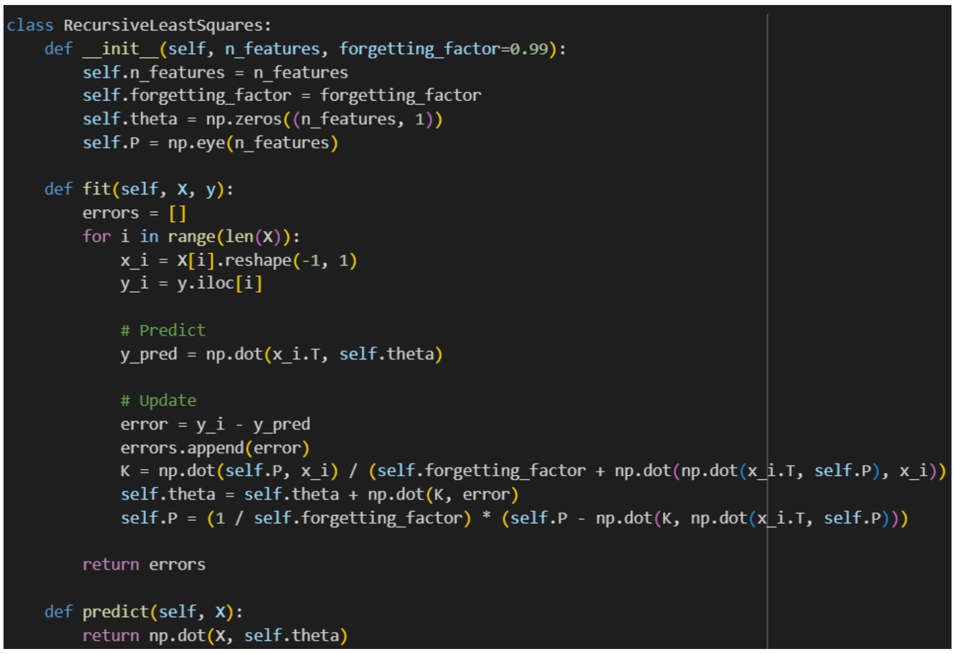
Now we want to implement LS and RLS estimation for this dataset.

The function to implement LS and RLS are shown in folloeing pictures:



```python
import numpy as np
import matplotlib.pyplot as plt

class LinearRegressionLS:
  def __init__(self):
    self.coefficients = None

  def fit(self,x,y):
    x = np.column_stack((np.ones(len(x)),x))
    self.coefficient = np.linalg.inv(x.T.dot(x)).dot(x.T).dot(y)

  def predict(self,x):
    x = np.column_stack((np.ones(len(x)),x))

    return x.dot(self.coefficient)
```

Figure 52 Least square method

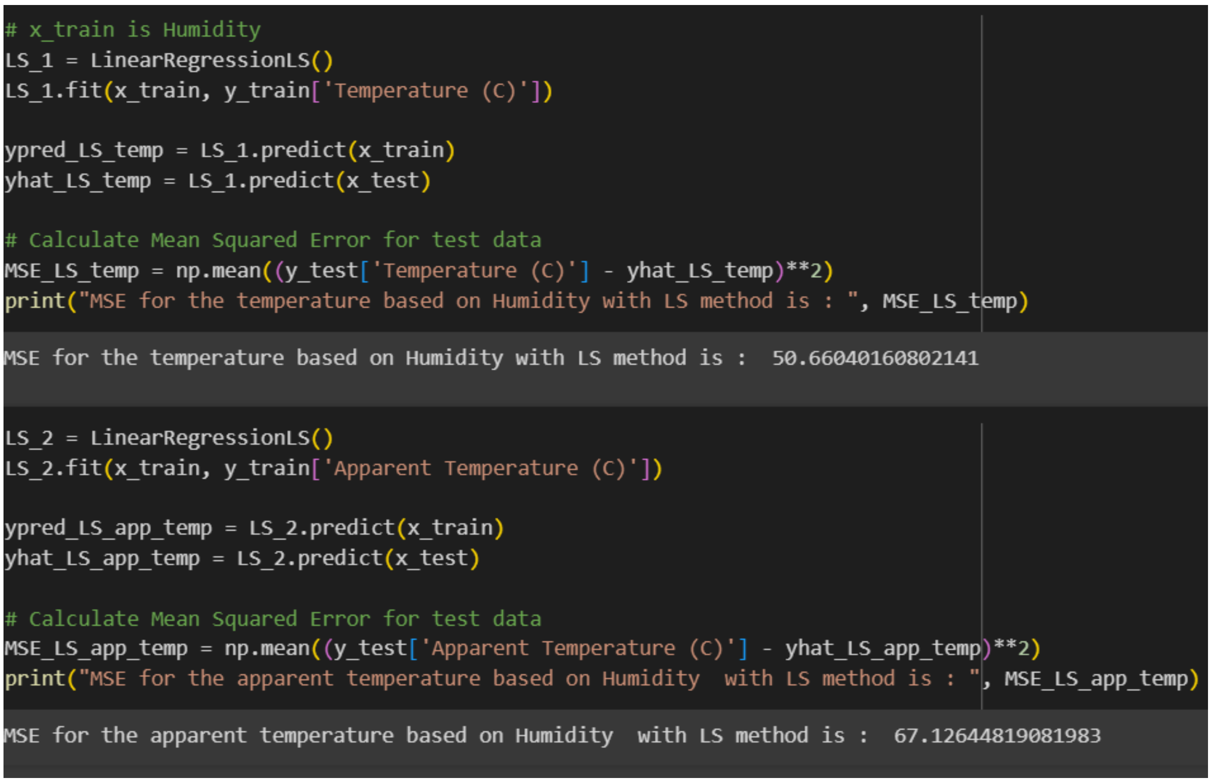


```python
class RecursiveLeastSquares:
    def __init__(self, n_features, forgetting_factor=0.99):
        self.n_features = n_features
        self.forgetting_factor = forgetting_factor
        self.theta = np.zeros((n_features, 1))
        self.P = np.eye(n_features)

    def fit(self, X, y):
        errors = []
        for i in range(len(X)):
            x_i = X[i].reshape(-1, 1)
            y_i = y.iloc[i]

            # Predict
            y_pred = np.dot(x_i.T, self.theta)

            # Update
            error = y_i - y_pred
            errors.append(error)
            K = np.dot(self.P, x_i) / (self.forgetting_factor + np.dot(np.dot(x_i.T, self.P), x_i))
            self.theta = self.theta + np.dot(K, error)
            self.P = (1 / self.forgetting_factor) * (self.P - np.dot(K, np.dot(x_i.T, self.P)))

        return errors

    def predict(self, X):
        return np.dot(X, self.theta)
```

Figure 53 Recursive least square code

Now we calculate the mean square error for the temperature and apparent temperature based on the humidity and shows the result. The former MSE is for predicting temperature based on humidity is 50.660 and the latter one is for apparent temperature is about 67.12.

We don't check the apparent temperature and temperature based on each other as they have same pattern and it is not practical to find the model.



```
# x_train is Humidity
LS_1 = LinearRegressionLS()
LS_1.fit(x_train, y_train['Temperature (C)'])

ypred_LS_temp = LS_1.predict(x_train)
yhat_LS_temp = LS_1.predict(x_test)

# Calculate Mean Squared Error for test data
MSE_LS_temp = np.mean((y_test['Temperature (C)'] - yhat_LS_temp)**2)
print("MSE for the temperature based on Humidity with LS method is : ", MSE_LS_temp)
```

```
MSE for the temperature based on Humidity with LS method is :  50.66040160802141
```

```
LS_2 = LinearRegressionLS()
LS_2.fit(x_train, y_train['Apparent Temperature (C)'])

ypred_LS_app_temp = LS_2.predict(x_train)
yhat_LS_app_temp = LS_2.predict(x_test)

# Calculate Mean Squared Error for test data
MSE_LS_app_temp = np.mean((y_test['Apparent Temperature (C)'] - yhat_LS_app_temp)**2)
print("MSE for the apparent temperature based on Humidity  with LS method is : ", MSE_LS_app_temp)
```

```
MSE for the apparent temperature based on Humidity  with LS method is :  67.12644819081983
```

Figure 54 MSE of the Temperature

Prediction of temperature with LS method and humidity data

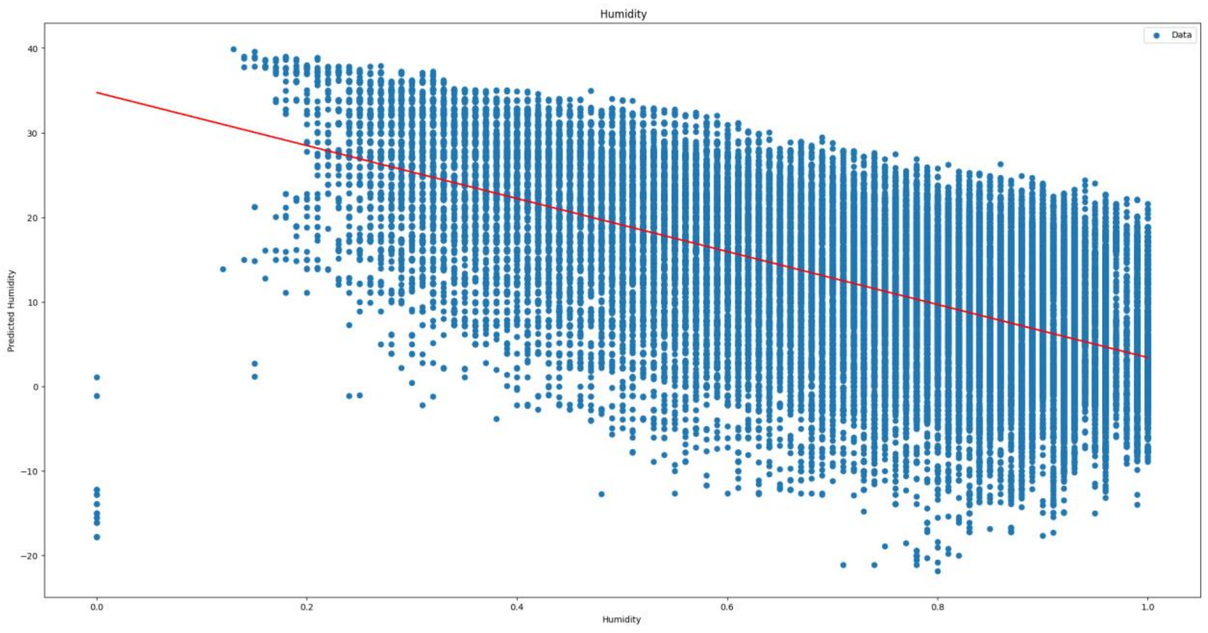


Figure 55 prediction of Temperature with LS

Prediction of apparent temperature with LS method and humidity data

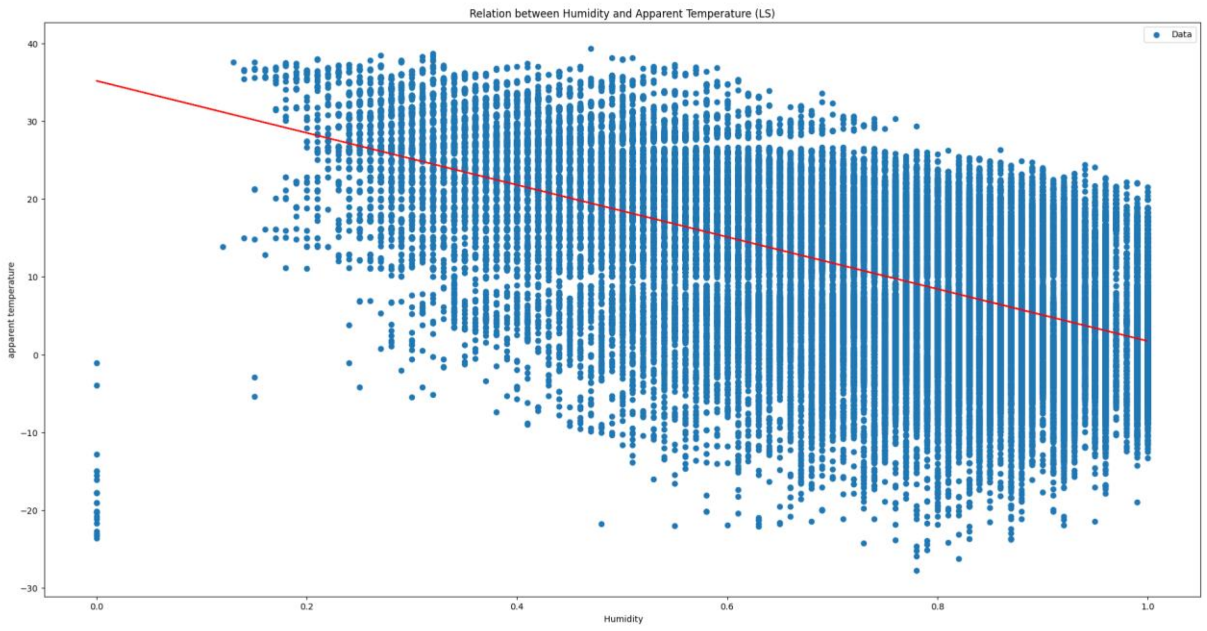
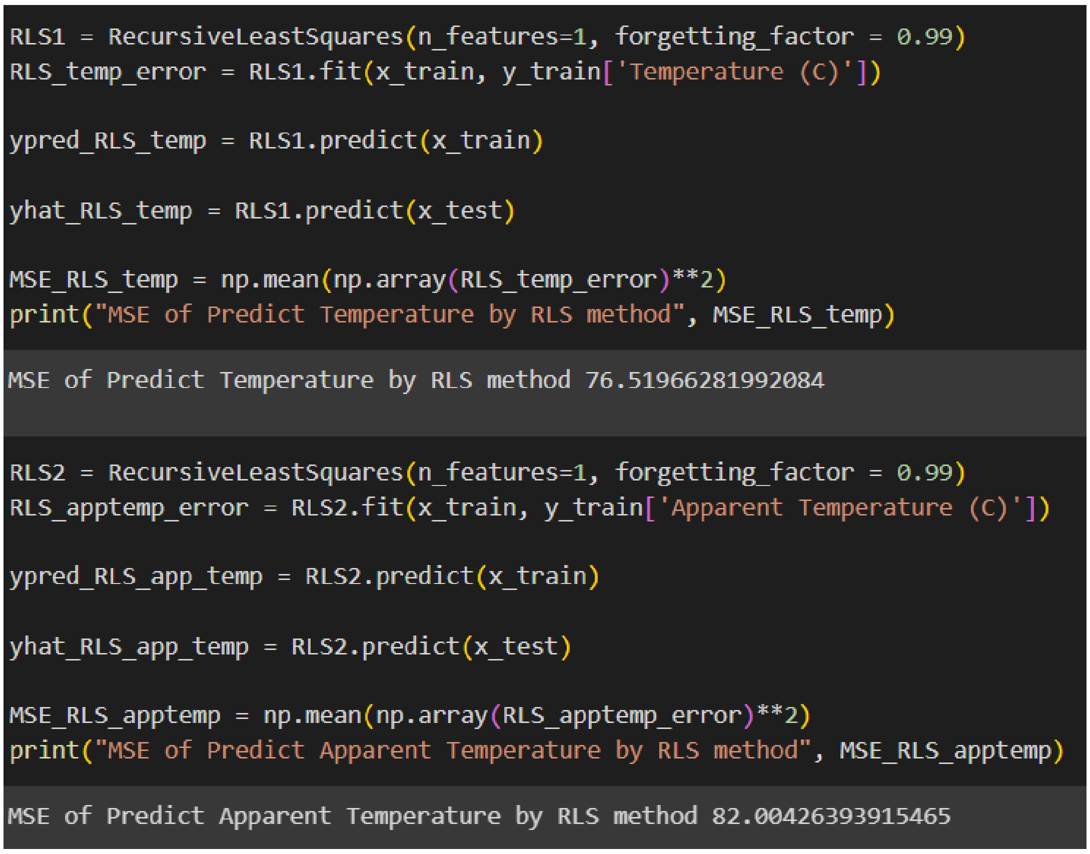


Figure 56 Prediction of Apparent Temperature with LS

	Train_MSE	Test_MSE
Temperature	56.15	50.66
Apparent temperature	74.79	67.12

RLS:

The RLS method is an online method and used for dataset which updated continuously and it using an adaptive algorithm to repeat the process on the dataset.



```
RLS1 = RecursiveLeastSquares(n_features=1, forgetting_factor = 0.99)
RLS_temp_error = RLS1.fit(x_train, y_train['Temperature (C)'])

ypred_RLS_temp = RLS1.predict(x_train)

yhat_RLS_temp = RLS1.predict(x_test)

MSE_RLS_temp = np.mean(np.array(RLS_temp_error)**2)
print("MSE of Predict Temperature by RLS method", MSE_RLS_temp)
```

```
MSE of Predict Temperature by RLS method 76.51966281992084
```

```
RLS2 = RecursiveLeastSquares(n_features=1, forgetting_factor = 0.99)
RLS_apptemp_error = RLS2.fit(x_train, y_train['Apparent Temperature (C)'])

ypred_RLS_app_temp = RLS2.predict(x_train)

yhat_RLS_app_temp = RLS2.predict(x_test)

MSE_RLS_apptemp = np.mean(np.array(RLS_apptemp_error)**2)
print("MSE of Predict Apparent Temperature by RLS method", MSE_RLS_apptemp)
```

```
MSE of Predict Apparent Temperature by RLS method 82.00426393915465
```

Figure 57 MSE for Temperature and Apparent Temperature

Prediction of temperature with RLS method and humidity data

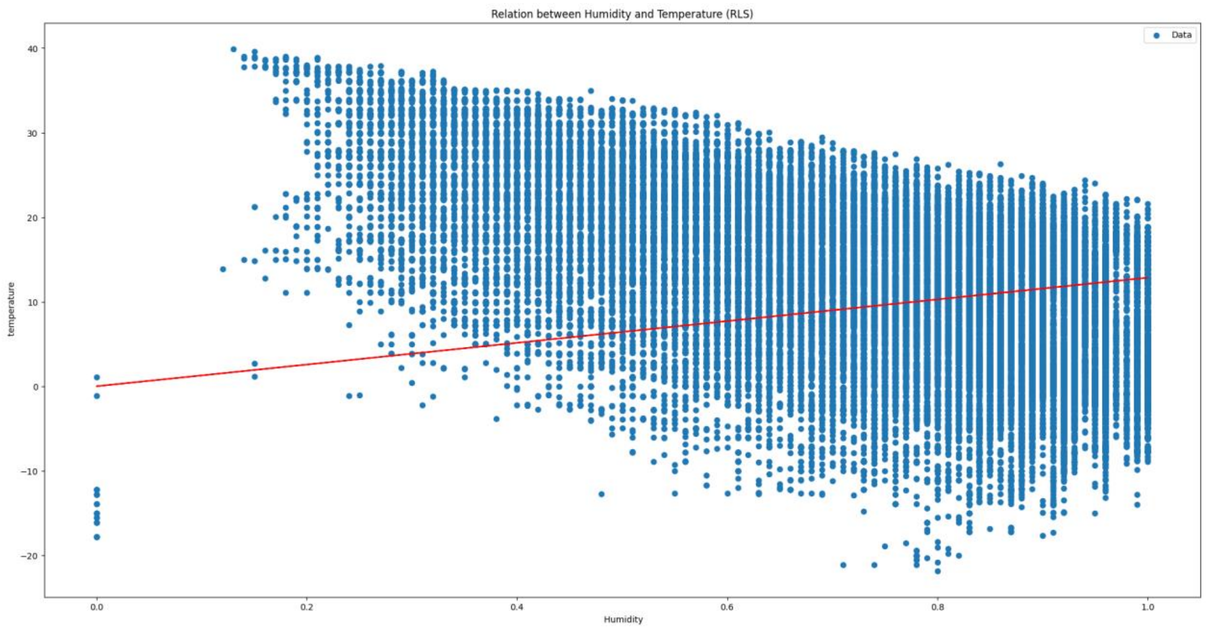


Figure 58Prediction of temperature with RLS

Prediction of apparent temperature with RLS method and humidity data

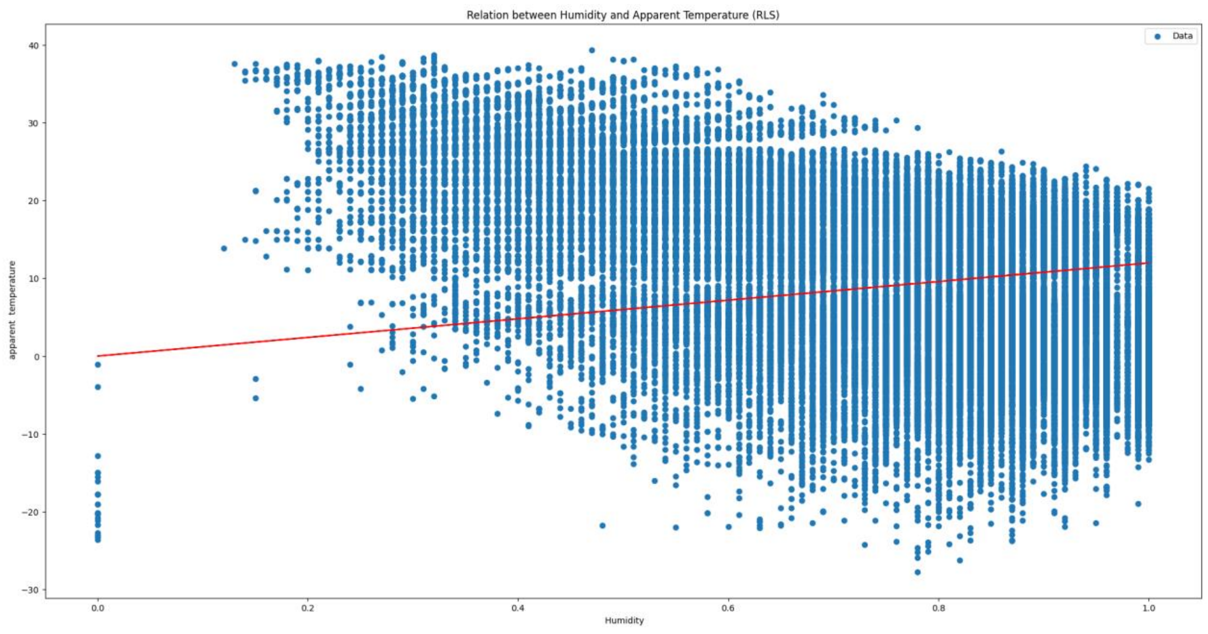


Figure 59 Prediction of apparent temperature with RLS

WLS)

Weighted least square is progressive LS and using wight for the data. Using weight for the data help the code to extract data which are more important than other and are more effective whereas it can ignore the different data or has less effect on the model. Thus, adding the weight to the data helps the model to use appropriate values and fit better model.

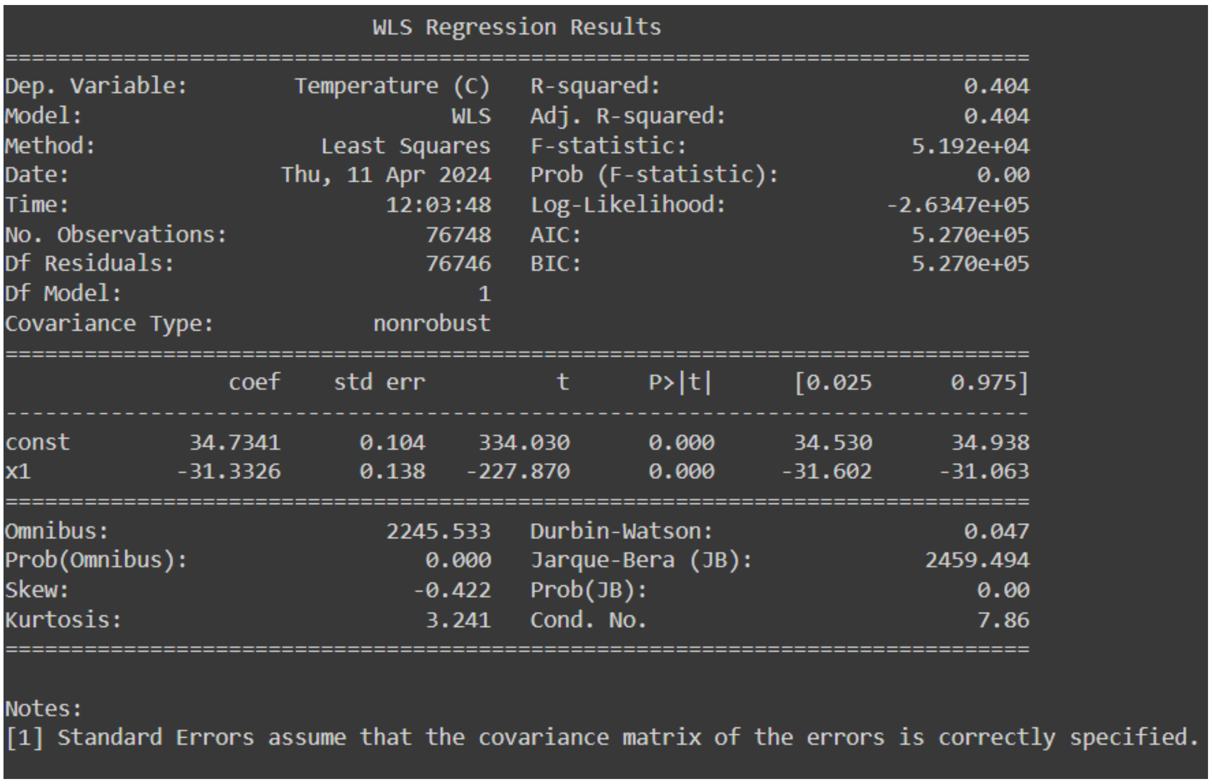
In this methos we use the variance and wight of each point. it means if a data has high varience it means it is similar to noise data so we decrease its wight so the weight of the data and its variance has opposite relation .

$$\sum w_i(y_t - y_i)^2 \rightarrow minimize\ this\ error\ by\ change\ the\ w_i for\ each\ point$$

$$w_i = \frac{1}{\sigma_i^2} \rightarrow w_i\ \sigma_i^2 = 1 \rightarrow opposite\ relation$$

$$X^T W X\hat{\beta} = X^T W y \rightarrow W = \begin{bmatrix} \frac{1}{\sigma_1^2} & 0 & 0 \\ 0 & \frac{1}{\sigma_2^2} & 0 \\ 0 & 0 & \ddots \end{bmatrix}, \hat{\beta} = \text{best linear unbiased estimator}$$

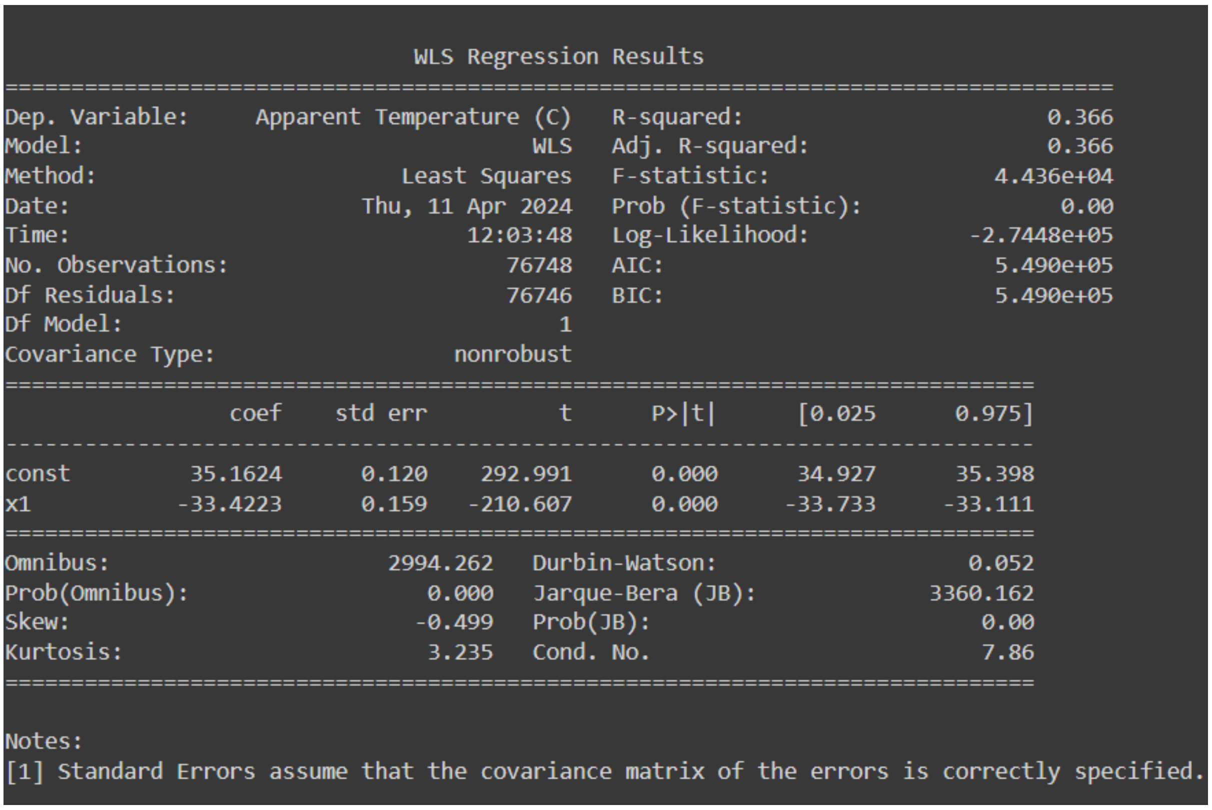
Humidity and Temperature



```
                            WLS Regression Results
==============================================================================
Dep. Variable:        Temperature (C)   R-squared:                       0.404
Model:                            WLS   Adj. R-squared:                  0.404
Method:                 Least Squares   F-statistic:                 5.192e+04
Date:                Thu, 11 Apr 2024   Prob (F-statistic):               0.00
Time:                        12:03:48   Log-Likelihood:            -2.6347e+05
No. Observations:               76748   AIC:                         5.270e+05
Df Residuals:                   76746   BIC:                         5.270e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         34.7341      0.104    334.030      0.000      34.530      34.938
x1           -31.3326      0.138   -227.870      0.000     -31.602     -31.063
==============================================================================
Omnibus:                     2245.533   Durbin-Watson:                   0.047
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2459.494
Skew:                          -0.422   Prob(JB):                         0.00
Kurtosis:                       3.241   Cond. No.                         7.86
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Figure 60 WLS for Temperature Data

Humidity and Apparent temperature



```
                               WLS Regression Results
====================================================================================
Dep. Variable:     Apparent Temperature (C)   R-squared:                       0.366
Model:                                  WLS   Adj. R-squared:                  0.366
Method:                       Least Squares   F-statistic:                 4.436e+04
Date:                      Thu, 11 Apr 2024   Prob (F-statistic):               0.00
Time:                              12:03:48   Log-Likelihood:            -2.7448e+05
No. Observations:                     76748   AIC:                         5.490e+05
Df Residuals:                         76746   BIC:                         5.490e+05
Df Model:                                 1
Covariance Type:                  nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         35.1624      0.120    292.991      0.000      34.927      35.398
x1           -33.4223      0.159   -210.607      0.000     -33.733     -33.111
==============================================================================
Omnibus:                     2994.262   Durbin-Watson:                   0.052
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3360.162
Skew:                          -0.499   Prob(JB):                         0.00
Kurtosis:                       3.235   Cond. No.                         7.86
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Figure 61 WLS for Apparent Temperature Data

The R_squared shows the performance of the fitting and how much it is higher the model is better.