MACHINE LEARNING LAB-5:LOGISTIC REGRESSION:

Submitted by:

Name: **Stebin George**

Register Number: **21122061**

Class: **2MSCDS**

Lab Overview

Objectives

TO get to know more anbout the logistic regression using the breast cancer dataset

LIBRARIES:

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

Questions:

Apply Logistic Regression for Breast Cancer Dataset. Use 60:40 train-test ratio for splitting the dataset.

1. Demonstrate the Logistic Regression for different penalties/regularisation methods - none, l1, l2 (you may use 'saga' solver as the parameter)
2. What happens when the Maximum Iterations are kept as 1, 2, 5, 10, 20, 50, 100, 500 and 1000? Is there any change in the accuracy.
3. Get the attributes: classes, *coef* and intercept_ and print the same in the above case.

Problem Definition:

This problem tries to predict the cancer predictions amoung the cancer dataset.The dataset used is cancer dataset in kaggle.

Approach

- Importing the necessary libraries
- analysing the data and doing the basic operations
- doing the EDA and pre-processing steps
- Loading the logistic regression function

- checking for the accuracy
- usage of penalties and maximum iter

Sections

1. Libraries
2. introduction
3. EDA
4. MODEL BUILDING
5. Q&A
6. Conclusion

References

1. https://www.quora.com/What-is-regularization-in-machine-learning
2. https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a
3. https://towardsdatascience.com/regularization-an-important-concept-in-machine-learning-5891628907ea

Loading Dataset:

In [2]:
```
df=pd.read_csv(r"C:\Users\stebi\OneDrive\Desktop\data.csv")
```

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

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840
1	842517	M	20.57	17.77	132.90	1326.0	0.08474
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960
3	84348301	M	11.42	20.38	77.58	386.1	0.14250
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030

5 rows × 33 columns

In [4]:
```
#checking the shape
df.shape
```

Out[4]: (569, 33)

Datset has 569 rows and 33 columns

In [5]:
```
df.describe()
```

Out[5]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400

8 rows × 32 columns

GIVES THE SUMMARY OF THE DATASET

CHECKING NULL VALUES:

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
```

```
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [7]:
```
df.isna().sum()
```

Out[7]:
```
id                           0
diagnosis                    0
radius_mean                  0
texture_mean                 0
perimeter_mean               0
area_mean                    0
smoothness_mean              0
compactness_mean             0
concavity_mean               0
concave points_mean          0
symmetry_mean                0
fractal_dimension_mean       0
radius_se                    0
texture_se                   0
perimeter_se                 0
area_se                      0
smoothness_se                0
compactness_se               0
concavity_se                 0
concave points_se            0
symmetry_se                  0
fractal_dimension_se         0
radius_worst                 0
texture_worst                0
perimeter_worst              0
area_worst                   0
smoothness_worst             0
compactness_worst            0
concavity_worst              0
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
```

In [8]:
```
df.dropna(axis=1,inplace=True)
```

We have dropped the entire column with null values.

LabelEnocding

In [9]:
```
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
df.iloc[:,1] = labelencoder.fit_transform(df.iloc[:,1].values)
```

Here the target variables are being encoded into numerical values as the exisiting dataset contains out put as 'M','B'.

In [10]:
```
df.head()
```

Out[10]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	842302	1	17.99	10.38	122.80	1001.0	0.11840
1	842517	1	20.57	17.77	132.90	1326.0	0.08474

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960
3	84348301	1	11.42	20.38	77.58	386.1	0.14250
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030

5 rows × 32 columns

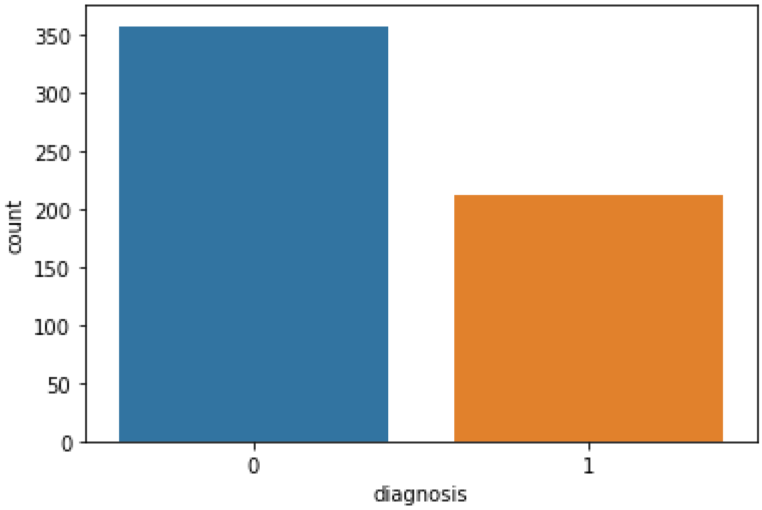
Here 'M' is 1 and 'B' is 0.

EDA:

In [11]:

```
ax = sns.countplot(df.diagnosis,label="Count")
B, M =df.diagnosis  .value_counts()
print('Number of Benign: ',B)
print('Number of Malignant : ',M)
```

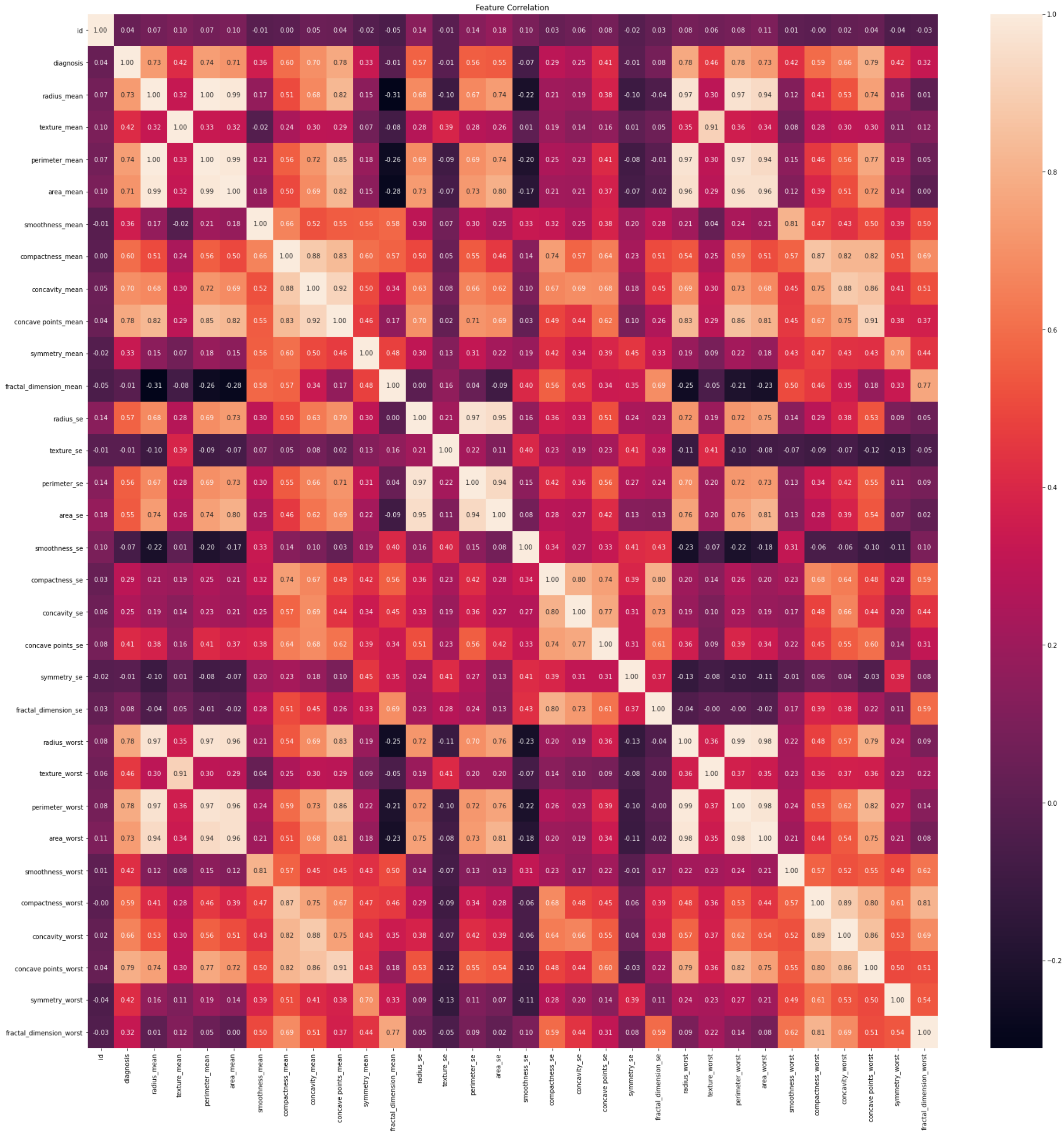
```
Number of Benign:  357
Number of Malignant :  212
C:\Users\stebi\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only valid p
ositional argument will be `data`, and passing other arguments without an explicit k
eyword will result in an error or misinterpretation.
  warnings.warn(
```



THE benign tumour has a number of 357 whereas the malignant tumour has the size of about 212.

In [12]:

```
# Correlation Matrix
plt.figure(figsize=(30,30))
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot = True, fmt = '.2f',)
plt.title("Feature Correlation")
plt.show()
```



The above graph shows the correlation plot of the different variables in the dataset.

In [13]:

```
df.corr()
```

Out[13]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_me
id	1.000000	0.039769	0.074626	0.099770	0.073159	0.0968
diagnosis	0.039769	1.000000	0.730029	0.415185	0.742636	0.7089
radius_mean	0.074626	0.730029	1.000000	0.323782	0.997855	0.9873
texture_mean	0.099770	0.415185	0.323782	1.000000	0.329533	0.3210
perimeter_mean	0.073159	0.742636	0.997855	0.329533	1.000000	0.9865
area_mean	0.096893	0.708984	0.987357	0.321086	0.986507	1.0000
smoothness_mean	-0.012968	0.358560	0.170581	-0.023389	0.207278	0.1770
compactness_mean	0.000096	0.596534	0.506124	0.236702	0.556936	0.4985
concavity_mean	0.050080	0.696360	0.676764	0.302418	0.716136	0.6859

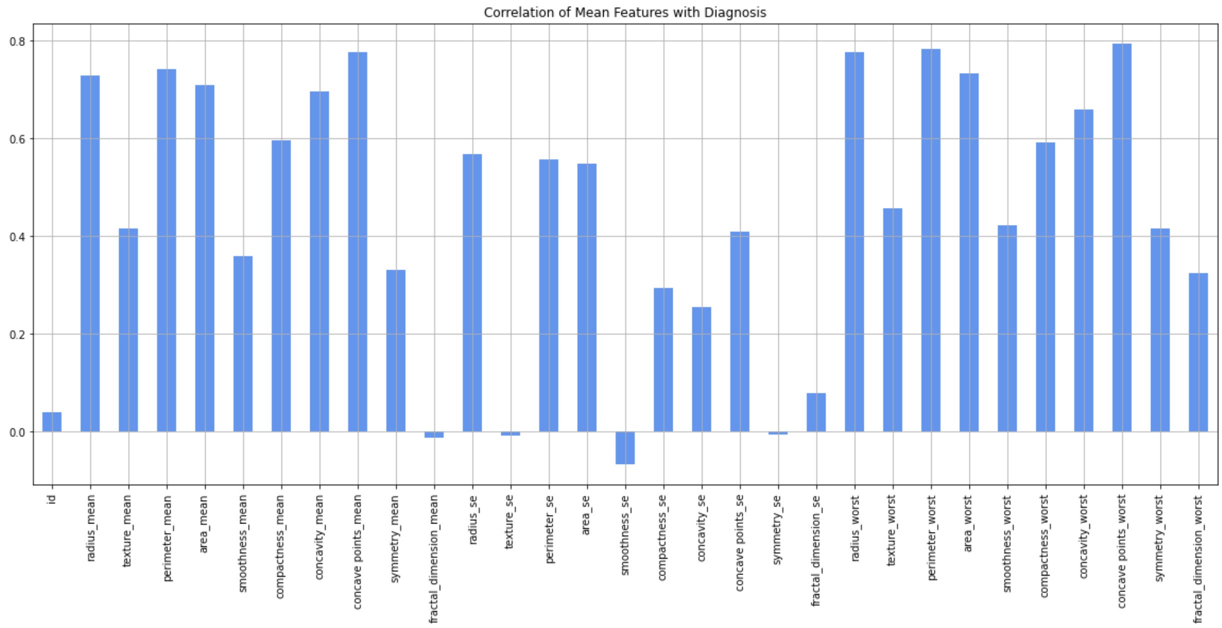
	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_me
concave points_mean	0.044158	0.776614	0.822529	0.293464	0.850977	0.8232
symmetry_mean	-0.022114	0.330499	0.147741	0.071401	0.183027	0.1512
fractal_dimension_mean	-0.052511	-0.012838	-0.311631	-0.076437	-0.261477	-0.2831
radius_se	0.143048	0.567134	0.679090	0.275869	0.691765	0.7325
texture_se	-0.007526	-0.008303	-0.097317	0.386358	-0.086761	-0.0662
perimeter_se	0.137331	0.556141	0.674172	0.281673	0.693135	0.7266
area_se	0.177742	0.548236	0.735864	0.259845	0.744983	0.8000
smoothness_se	0.096781	-0.067016	-0.222600	0.006614	-0.202694	-0.1667
compactness_se	0.033961	0.292999	0.206000	0.191975	0.250744	0.2125
concavity_se	0.055239	0.253730	0.194204	0.143293	0.228082	0.2076
concave points_se	0.078768	0.408042	0.376169	0.163851	0.407217	0.3723
symmetry_se	-0.017306	-0.006522	-0.104321	0.009127	-0.081629	-0.0724
fractal_dimension_se	0.025725	0.077972	-0.042641	0.054458	-0.005523	-0.0198
radius_worst	0.082405	0.776454	0.969539	0.352573	0.969476	0.9627
texture_worst	0.064720	0.456903	0.297008	0.912045	0.303038	0.2874
perimeter_worst	0.079986	0.782914	0.965137	0.358040	0.970387	0.9591
area_worst	0.107187	0.733825	0.941082	0.343546	0.941550	0.9592
smoothness_worst	0.010338	0.421465	0.119616	0.077503	0.150549	0.1235
compactness_worst	-0.002968	0.590998	0.413463	0.277830	0.455774	0.3904
concavity_worst	0.023203	0.659610	0.526911	0.301025	0.563879	0.5126
concave points_worst	0.035174	0.793566	0.744214	0.295316	0.771241	0.7220
symmetry_worst	-0.044224	0.416294	0.163953	0.105008	0.189115	0.1435
fractal_dimension_worst	-0.029866	0.323872	0.007066	0.119205	0.051019	0.0037

32 rows × 32 columns

Feature contributions to the target variable:

In [14]:

```
df_mean = df[df.columns[:]]
plt.figure(figsize=(20, 8))
df_mean.drop('diagnosis', axis=1).corrwith(df_mean.diagnosis).plot(kind='bar', grid=
```



MODEL BUILDING:

LOGISTIC REGRESSION:

In [15]:
```
import warnings
warnings.filterwarnings('ignore')
```

Splitting the data

In [16]:
```
X = df.iloc[:,2:]
y = df.iloc[:,1]
```

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,random_state
```

In [18]:
```
X_train.head()
```

Out[18]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mear
296	10.91	12.35	69.14	363.7	0.08518	0.04721
490	12.25	22.44	78.18	466.5	0.08192	0.0520(
519	12.75	16.70	82.51	493.8	0.11250	0.1117(
513	14.58	13.66	94.29	658.8	0.09832	0.0891ᵋ
473	12.27	29.97	77.42	465.4	0.07699	0.0339ᵋ

5 rows × 30 columns

Standardizing the dataset:

In [19]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

LOGISTIC REGRESSION:

In [20]:
```
classifer = LogisticRegression()
classifer.fit(X_train, y_train)
```

Out[20]: LogisticRegression()

Prediction on Unseen data:

In [21]:
```
predictions  = classifer.predict(X_test)
predictions
```

Out[21]:
```
array([0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 1, 0])
```

Q2:

What happens when the Maximum Iterations are kept as 1, 2, 5, 10, 20, 50, 100, 500 and 1000? Is there any change in the accuracy.

In [22]:
```
l1=[1, 2, 5, 10, 20, 50, 100, 500,1000]
iteration1=[]
accuracy=[]
for i in l1:
    classifer = LogisticRegression(max_iter=i)
    classifer.fit(X_train, y_train)
    predictions  = classifer.predict(X_test)
    accuracy.append(accuracy_score(y_test,predictions))
    iteration1.append(i)
```

In [23]:
```
score1=pd.DataFrame()
score1['iteration']=iteration1
score1['accuracy']=accuracy
score1
```

Out[23]:

	iteration	accuracy
0	1	0.934211
1	2	0.969298
2	5	0.991228
3	10	0.982456
4	20	0.982456
5	50	0.982456
6	100	0.982456

	iteration	accuracy
7	500	0.982456
8	1000	0.982456

The accuarancy is maximum at 5 iteration state and the accuracy changes for each iterations as well

Q1:

Demonstrate the Logistic Regression for different penalties/regularisation methods - none, l1, l2 (you may use 'saga' solver as the parameter.

Regularization in Machine Learning As mentioned above, regularization is used to avoid overfitting due to complex models. When a regularization model is used, the learning model takes only a limited set of parameters. Instead of choosing parameters from a discrete grid, this process chooses values from a continuum that produces a smoothing effect (thereby reducing the noise terms). In ML models, individual significance of variables and interaction effects is not observed stage wise. In such a case, regularization also helps in the selection process of features contribute to the model. The regularization methods in ML generally adds some kind of penalty to the cost function which is further used in the adjustment process.

L1 and L2 are two types of regularization techniques. Both of these models introduces an additional term of "penalty" on the model based on the error function. The process of weight adjustment hence will also consider the penalty that is applied by these penalties. The key difference between these two is the penalty term.

L1 Regularization: L1 (Lazzo) regularization uses the sum of the absolute values of the weights is considered as a penalty. This type is preferred when the model is generally linear in nature with lesser number of coefficients, since it encourages the convergence towards 0. It is also useful for the case of considering a categorical variable with many levels (as mention above, helps in feature selection).

L2 Regularization: L2 (Ridge) regularization uses the sum of squared values of weights as the penalty. It tries to make the convergence closer to 0 and prevents overfitting; which becomes very useful when the number of variables are very large and smaller data samples. Genomic data is a very good example to apply L2 regularization.

The regularization parameter penalizes all parameters except intercept; and as the complexity of the model is increased, it also adds penalty for the higher terms. Apart from L1 and L2 regularization techniques, there is something also known as "Elastic Net" which is a hybrid type of both of these techniques.

In [24]:

```
l2=['l1','l2','none']
pen=[]
accuracy=[]
iteration=[]
l1=[1, 2, 5, 10, 20, 50, 100, 500,1000]
for i in l1:
    for j in l2:
        classifer = LogisticRegression(penalty=j,solver='saga',max_iter=i)
```

```
        classifer.fit(X_train, y_train)
        predictions = classifer.predict(X_test)
        pen.append(j)
        accuracy.append(accuracy_score(y_test,predictions))
        iteration.append(i)
```

In [25]:

```
score=pd.DataFrame()
score['penalty']=pen
score['iteration']=iteration
score['accuracy']=accuracy
score
```

Out[25]:

	penalty	iteration	accuracy
0	l1	1	0.978070
1	l2	1	0.978070
2	none	1	0.982456
3	l1	2	0.982456
4	l2	2	0.978070
5	none	2	0.978070
6	l1	5	0.986842
7	l2	5	0.991228
8	none	5	0.986842
9	l1	10	0.986842
10	l2	10	0.986842
11	none	10	0.986842
12	l1	20	0.986842
13	l2	20	0.986842
14	none	20	0.986842
15	l1	50	0.986842
16	l2	50	0.986842
17	none	50	0.986842
18	l1	100	0.986842
19	l2	100	0.986842
20	none	100	0.986842
21	l1	500	0.982456
22	l2	500	0.982456
23	none	500	0.986842
24	l1	1000	0.986842
25	l2	1000	0.982456
26	none	1000	0.978070

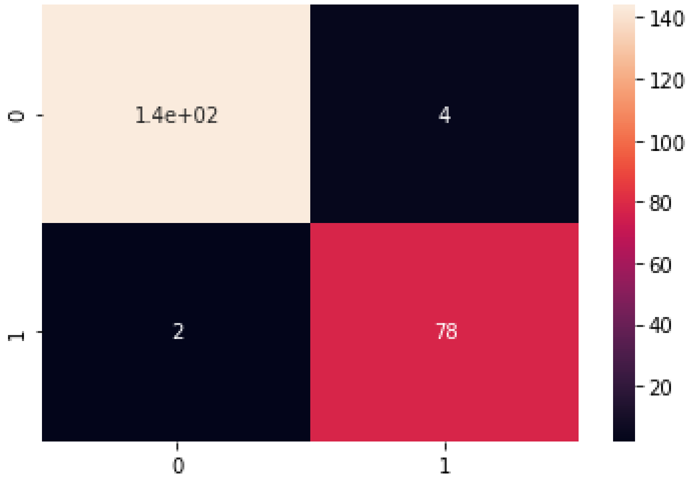
the value of the accuracy changes for different penalties.

Confusion Matrix for different iteration values and penalties:

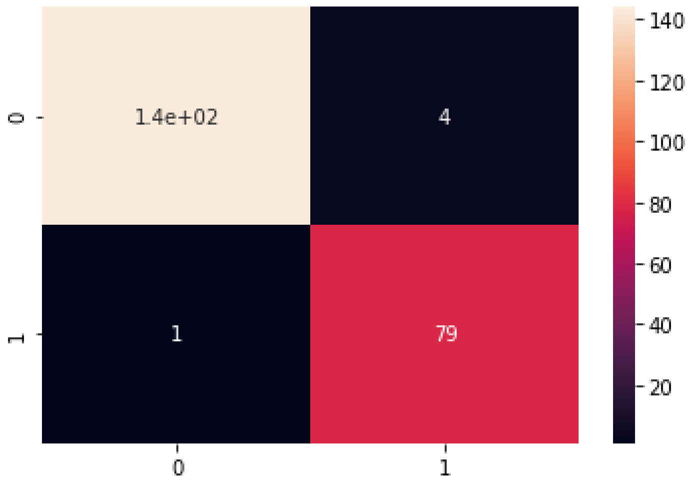
In [26]:

```
l2=['l1','l2','none']
pen=[]
accuracy=[]
iteration=[]
l1=[1, 2, 5, 10, 20, 50, 100, 500,1000]
for i in l1:
    for j in l2:
        print(f"\033[1m for penalty:{j} iteration :{i} \033[0m")
        classifer = LogisticRegression(penalty=j,solver='saga',max_iter=i)
        classifer.fit(X_train, y_train)
        predictions = classifer.predict(X_test)
        cm = confusion_matrix(y_test, predictions)
        sns.heatmap(cm, annot=True,)
        plt.show()
```

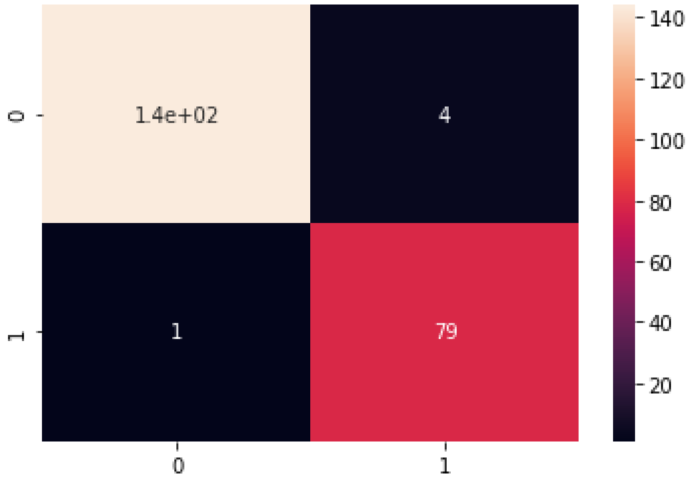
for penalty:l1 iteration :1



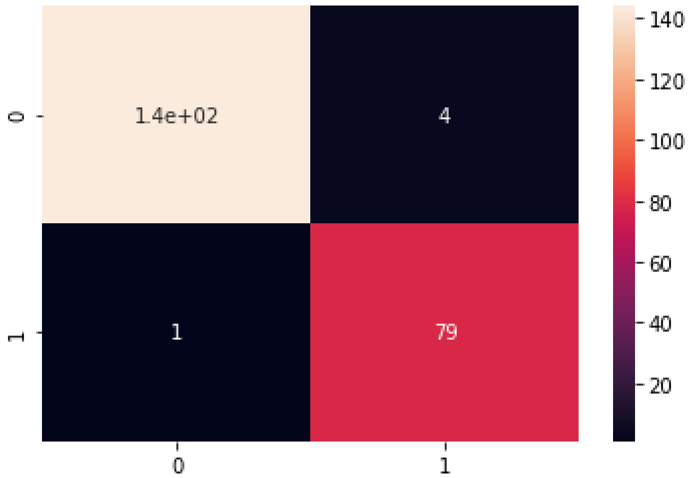
for penalty:l2 iteration :1



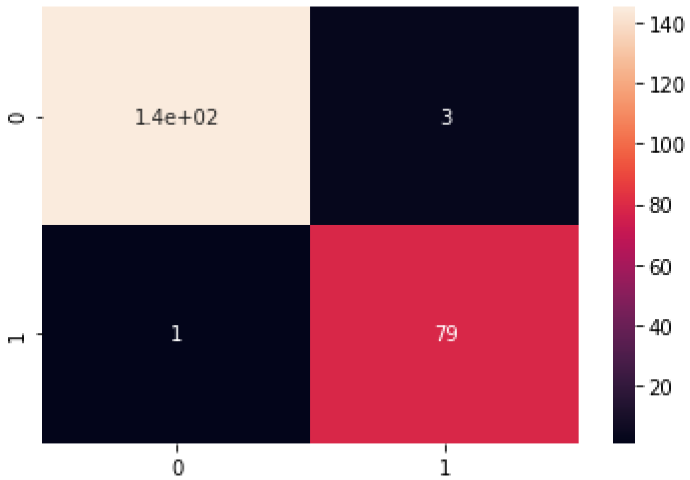
for penalty:none iteration :1



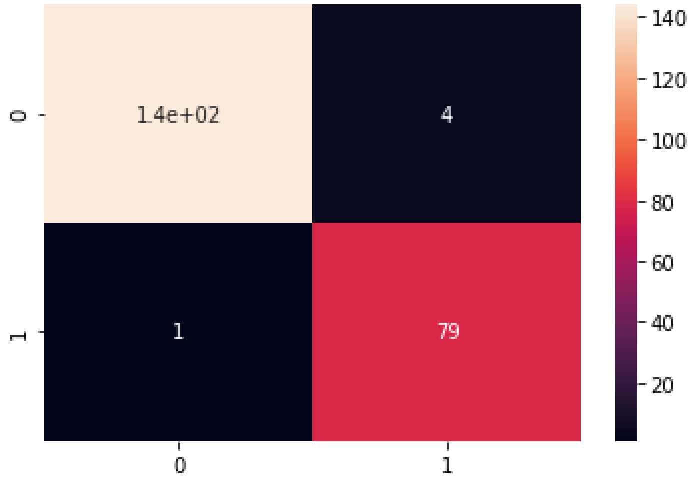

```
for penalty:l1 iteration :2
```



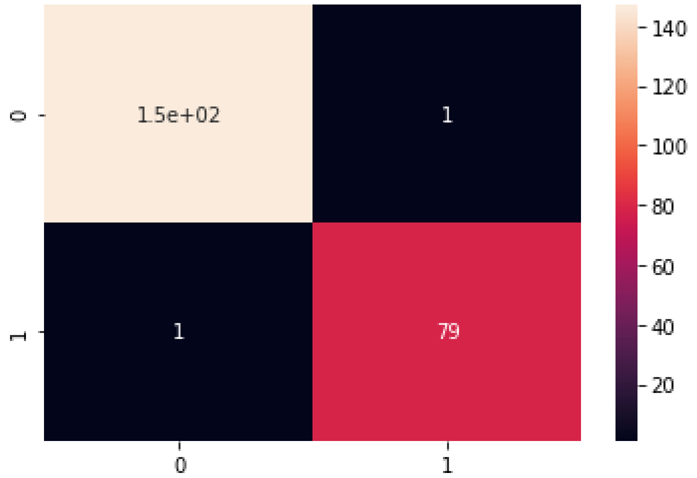

```
for penalty:l2 iteration :2
```



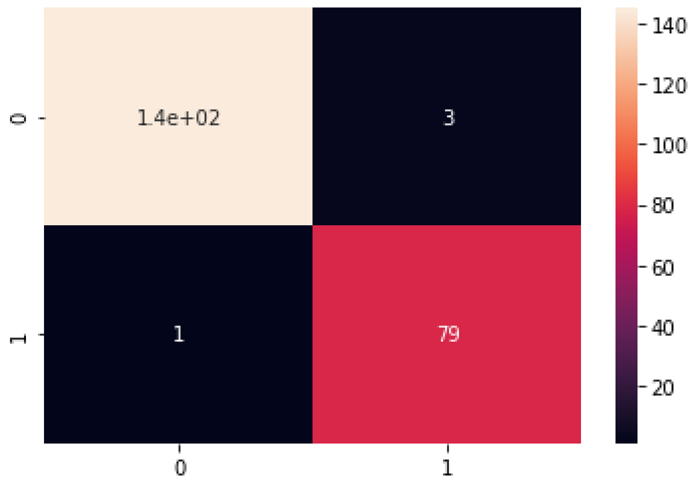

```
for penalty:none iteration :2
```



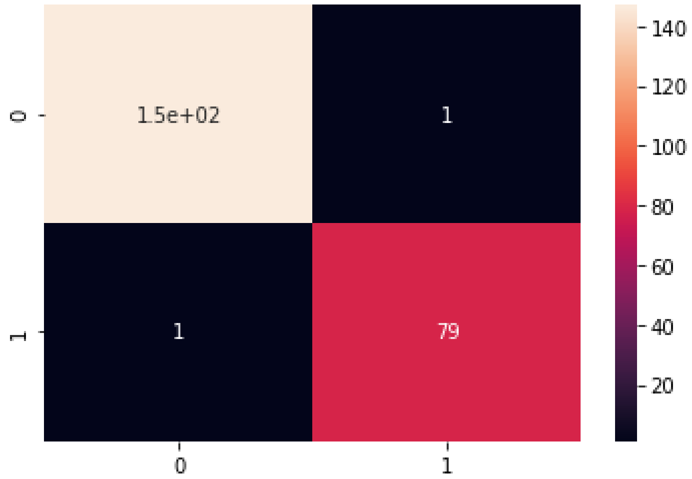
for penalty:l1 iteration :5



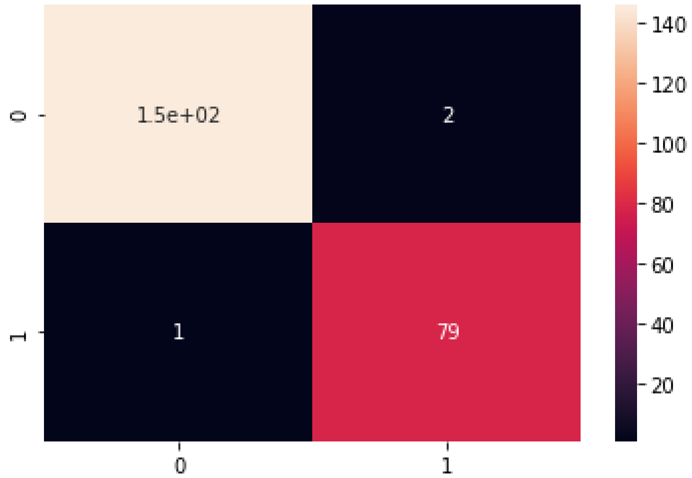
for penalty:l2 iteration :5



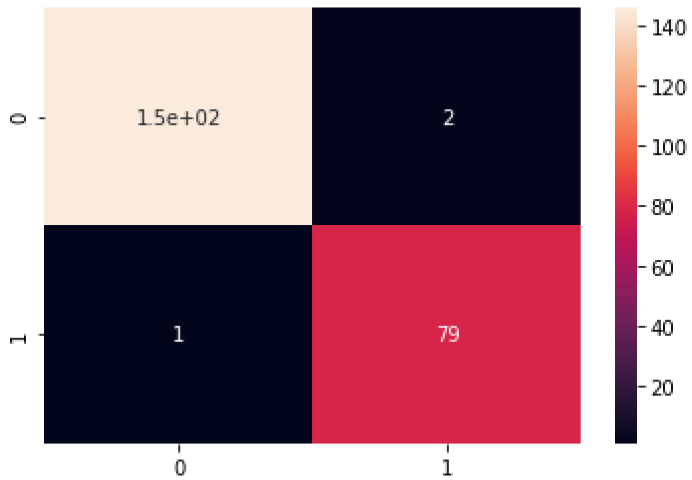
for penalty:none iteration :5



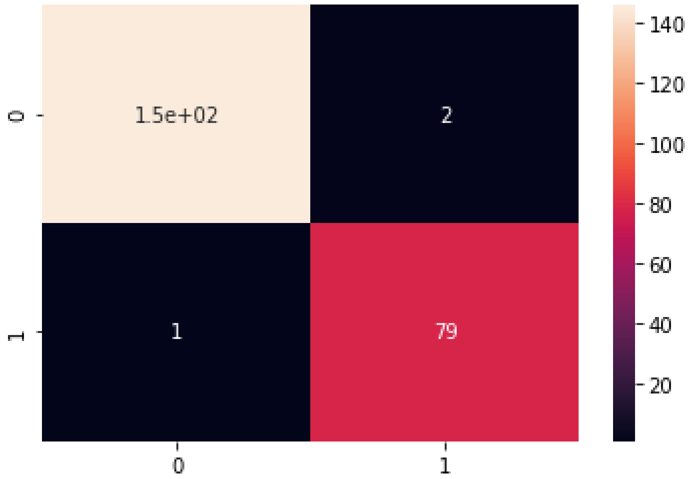
for penalty:l1 iteration :10



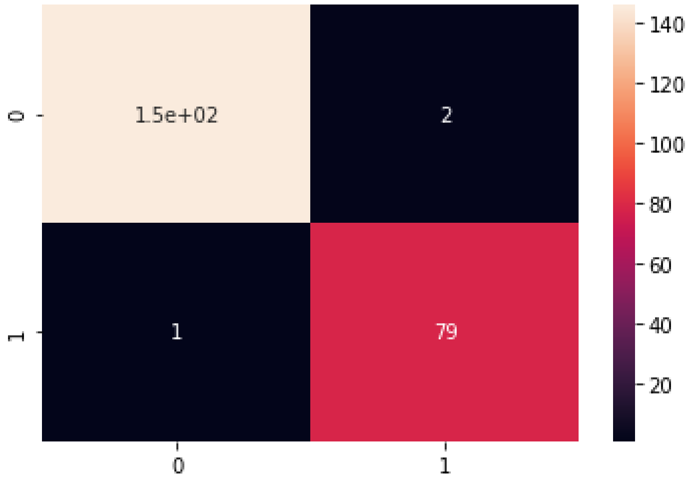
for penalty:l2 iteration :10



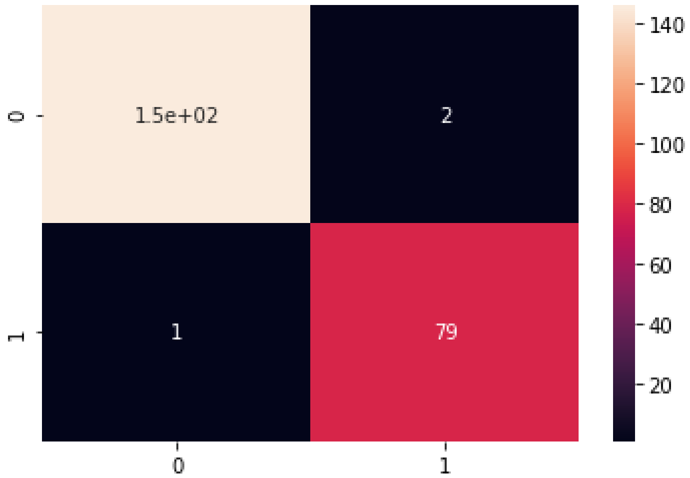
for penalty:none iteration :10



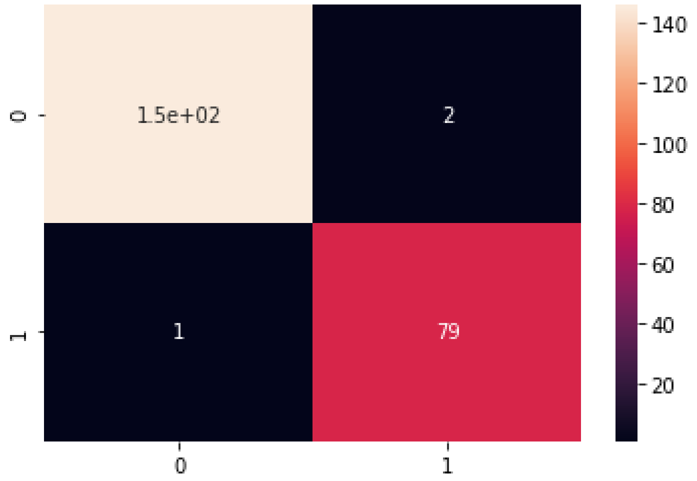
for penalty:l1 iteration :20



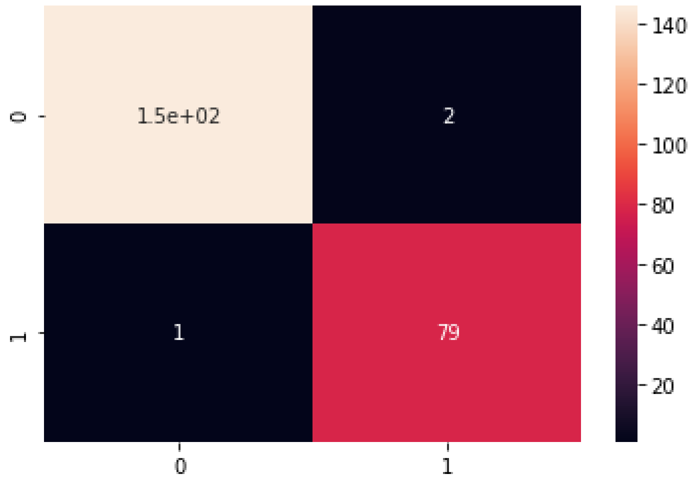
for penalty:l2 iteration :20



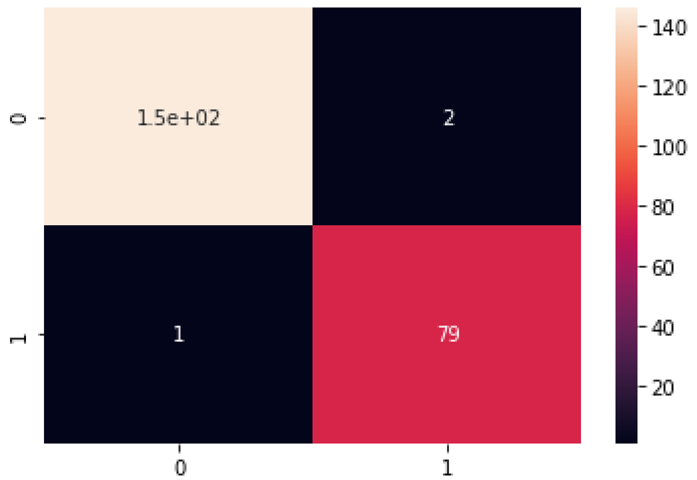
for penalty:none iteration :20



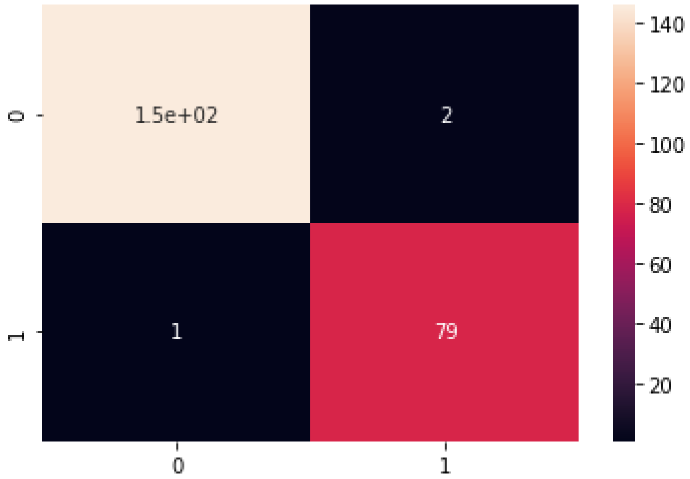

```
for penalty:l1 iteration :50
```



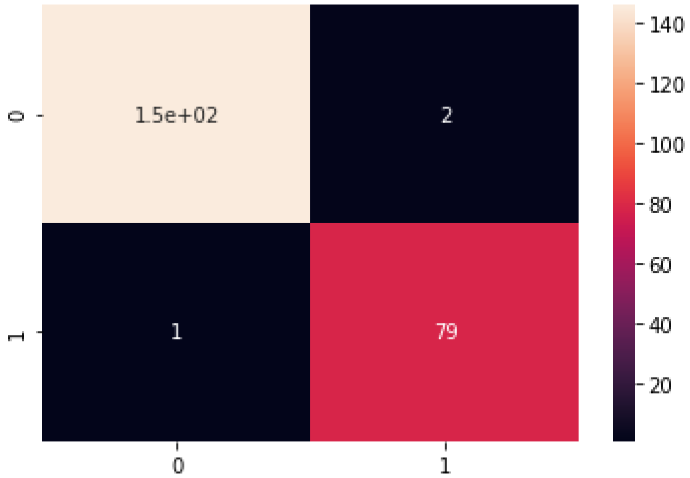

```
for penalty:l2 iteration :50
```



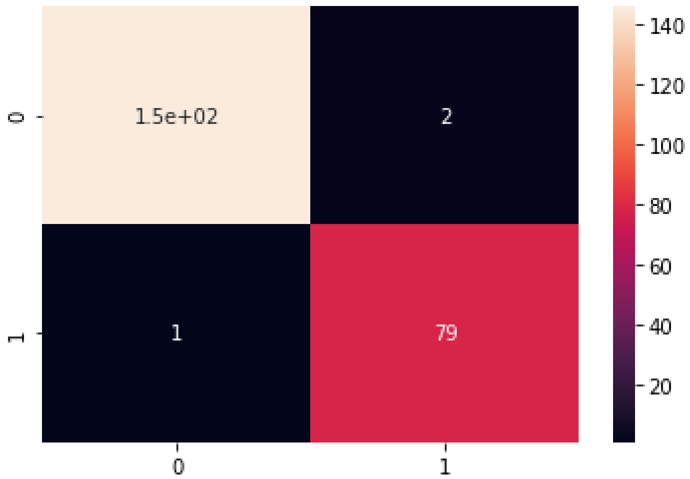

```
for penalty:none iteration :50
```



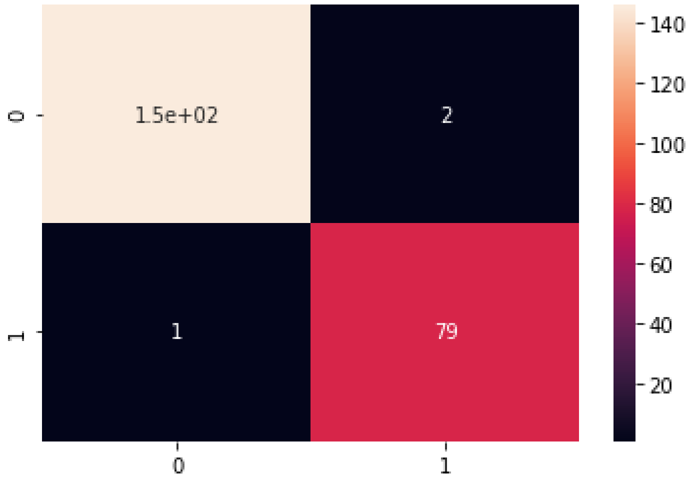

```
for penalty:l1 iteration :100
```



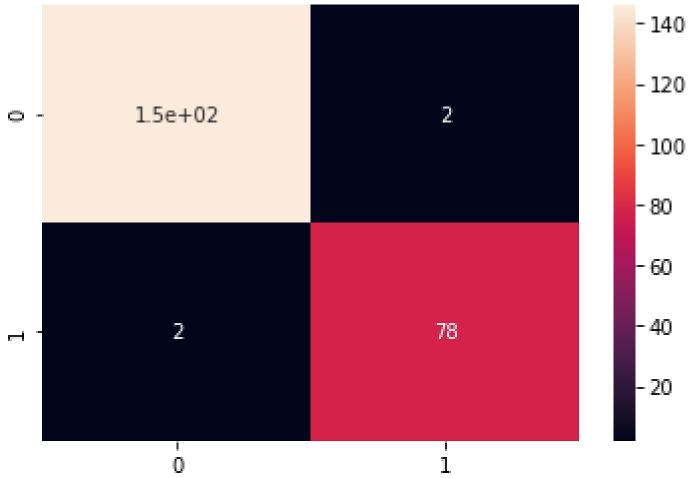

```
for penalty:l2 iteration :100
```



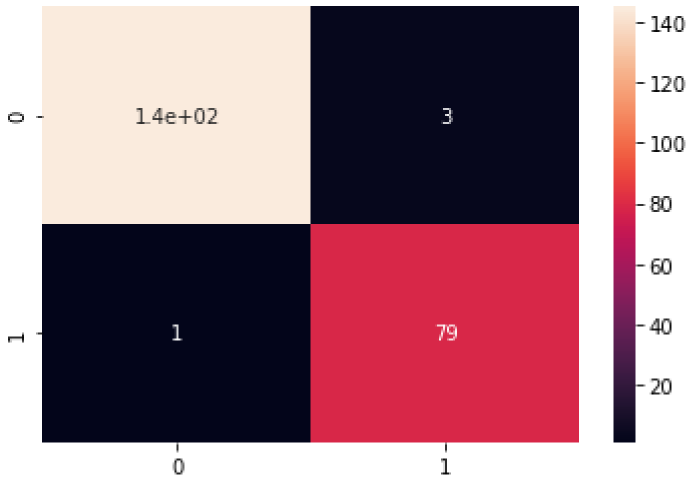

```
for penalty:none iteration :100
```



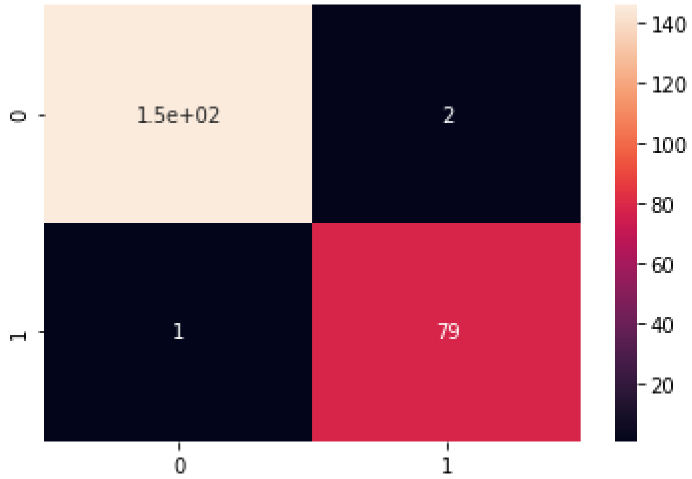

```
for penalty:l1 iteration :500
```



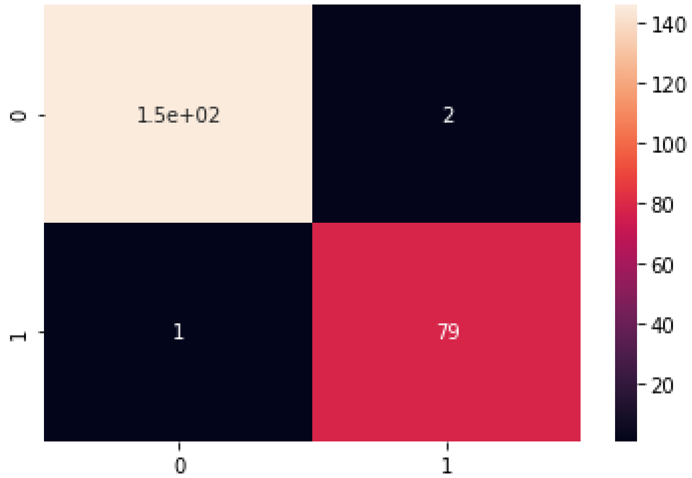

```
for penalty:l2 iteration :500
```



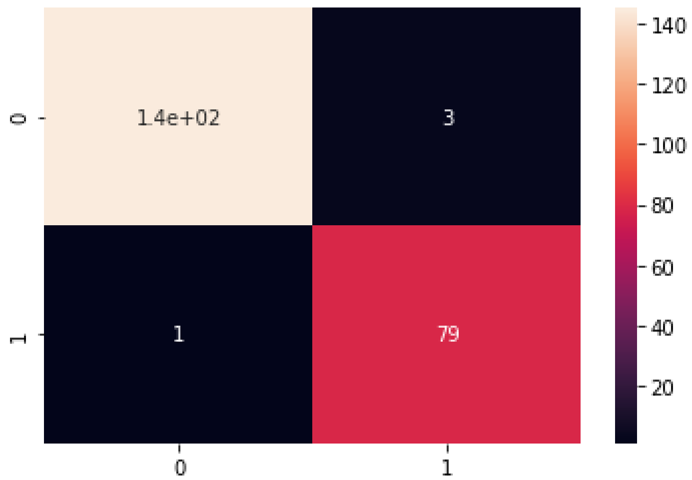

```
for penalty:none iteration :500
```



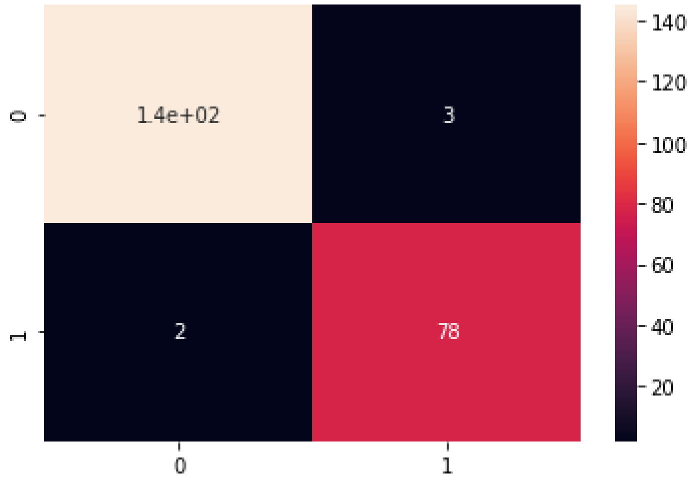
for penalty:l1 iteration :1000



for penalty:l2 iteration :1000



for penalty:none iteration :1000



Q3)

Get the attributes: classes, *coef* and intercept_ and print the same in the above case.

In [27]:

```
print(classifer.classes_)
```

```
[0 1]
```

Class will give the classification criteria's

In [28]:

```
print(classifer.coef_)
```

```
[[-0.08209231 -0.24271427 -0.14039311  0.14068608 -0.02724844 -2.60382948
   1.64202895  2.92180359 -1.0494043   0.60192373  2.99381621 -0.59572542
   1.14023774  2.03985847  0.31522363 -0.32408321 -0.74506811  1.4220966
  -1.07014165 -1.37504414  1.29477287  2.91693097  0.43178706  1.27633799
   0.3692476  -1.03686318  2.32426085  1.45587324  2.99661001  0.00993696]]
```

coef_ attribute is also used to view the model's coefficients.

In [29]:

```
print(classifer.intercept_)
```

```
[0.50431558]
```

The value of b_0, also called the intercept, shows the point where the estimated regression line crosses the y axis.

coeff,class and intercept in a single dataframe:

In [30]:

```
l2=['l1','l2','none']
pen=[]
accuracy=[]
iteration=[]
cls=[]
intercept=[]
l1=[1, 2, 5, 10, 20, 50, 100, 500,1000]
for i in l1:
    for j in l2:
        classifer = LogisticRegression(penalty=j,solver='saga',max_iter=i)
        classifer.fit(X_train, y_train)
        predictions = classifer.predict(X_test)
        pen.append(j)
        accuracy.append(accuracy_score(y_test,predictions))
        iteration.append(i)
```

```
        cls.append(classifer.classes_)
        intercept.append(classifer.intercept_)
```

In [31]:

```
score=pd.DataFrame()
score['penalty']=pen
score['iteration']=iteration
score['class']=cls
score['intercept']=intercept
score['accuracy']=accuracy
score
```

Out[31]:

	penalty	iteration	class	intercept	accuracy
0	l1	1	[0, 1]	[-0.23668199745488105]	0.978070
1	l2	1	[0, 1]	[-0.13829212325660264]	0.973684
2	none	1	[0, 1]	[-0.2183925010149449]	0.969298
3	l1	2	[0, 1]	[-0.2884212079162961]	0.978070
4	l2	2	[0, 1]	[-0.2475204446078854]	0.978070
5	none	2	[0, 1]	[-0.22035714382121543]	0.973684
6	l1	5	[0, 1]	[-0.316585280847195]	0.991228
7	l2	5	[0, 1]	[-0.2892900699067467]	0.991228
8	none	5	[0, 1]	[-0.3403412463853296]	0.991228
9	l1	10	[0, 1]	[-0.32936068945772773]	0.986842
10	l2	10	[0, 1]	[-0.3380908986415193]	0.986842
11	none	10	[0, 1]	[-0.32709034019805494]	0.986842
12	l1	20	[0, 1]	[-0.35457779897449415]	0.986842
13	l2	20	[0, 1]	[-0.33119958977486996]	0.986842
14	none	20	[0, 1]	[-0.3051862912161733]	0.986842
15	l1	50	[0, 1]	[-0.289659664434936]	0.986842
16	l2	50	[0, 1]	[-0.290007174746684]	0.986842
17	none	50	[0, 1]	[-0.23370248527047907]	0.986842
18	l1	100	[0, 1]	[-0.22760880047779694]	0.986842
19	l2	100	[0, 1]	[-0.24405867192623548]	0.986842
20	none	100	[0, 1]	[-0.18748281702627992]	0.986842
21	l1	500	[0, 1]	[-0.11619733973993761]	0.982456
22	l2	500	[0, 1]	[-0.16782756752390357]	0.982456
23	none	500	[0, 1]	[0.21018414315765624]	0.986842
24	l1	1000	[0, 1]	[-0.1211145418156155]	0.986842
25	l2	1000	[0, 1]	[-0.1680448619854572]	0.982456
26	none	1000	[0, 1]	[0.508756889918008]	0.978070

In [36]:

```
l2=['l1','l2','none']
```

```
l1=[1, 2, 5, 10, 20, 50, 100, 500,1000]
for i in l1:
    for j in l2:
        print(f"\033[1m for penalty:{j} iteration :{i} \033[0m")
        classifer = LogisticRegression(penalty=j,solver='saga',max_iter=i)
        classifer.fit(X_train, y_train)
        print(classifer.coef_)
```

```
 for penalty:l1 iteration :1
[[ 0.29939923  0.18428545  0.29840871  0.29303173  0.11491086  0.14196168
   0.20783063  0.30922839  0.02041179 -0.1108061   0.27789855 -0.04464459
   0.25630959  0.26725932  0.01504474 -0.00784784 -0.04295042  0.06628419
  -0.0061287  -0.12106751  0.34664178  0.19631491  0.34333632  0.33076436
   0.20199335  0.1976854   0.21550931  0.31057107  0.14607356  0.05067992]]
 for penalty:l2 iteration :1
[[ 0.30772722  0.1476935   0.30861533  0.2923931   0.12821557  0.16855408
   0.2158799   0.32387895  0.11171781 -0.09178241  0.20290227  0.02332918
   0.18086412  0.1852561  -0.00465121 -0.01015706 -0.06885493  0.06500782
   0.02061968 -0.1127541   0.3404487   0.22098434  0.33728278  0.31071569
   0.18703813  0.19074379  0.21417831  0.32611309  0.24138634  0.06907217]]
 for penalty:none iteration :1
[[ 0.27469465  0.17003401  0.27766449  0.26539881  0.08244562  0.14305233
   0.2106261   0.28895639  0.07347412 -0.11867997  0.248041   -0.00448318
   0.22349573  0.21140011 -0.02798544  0.00621922 -0.05187633  0.02132737
   0.0438492  -0.10621651  0.31875083  0.21893434  0.31568956  0.29270389
   0.16854376  0.18382287  0.21094366  0.27244141  0.22007961  0.04526099]]
 for penalty:l1 iteration :2
[[ 0.33829162  0.23386688  0.33997084  0.32466593  0.11034657  0.15426222
   0.22922791  0.35662294  0.07947149 -0.13017008  0.25399705  0.01460276
   0.22677289  0.22824076 -0.00691679 -0.03925665 -0.09660454  0.07728208
  -0.04025172 -0.12436324  0.37279925  0.31008387  0.36246884  0.33871322
   0.27269515  0.20763707  0.21387935  0.36341307  0.22129434  0.05979718]]
 for penalty:l2 iteration :2
[[ 0.35339251  0.25375568  0.35327903  0.33624326  0.12707273  0.14536483
   0.2677804   0.37462159  0.12556224 -0.15581954  0.25383968  0.00723874
   0.22094989  0.23172696 -0.00814101 -0.02348042 -0.09185266  0.03898864
  -0.06912468 -0.13841692  0.38674038  0.30696819  0.37606186  0.35532
   0.22746563  0.21789741  0.27724091  0.3739801   0.2179219   0.10371223]]
 for penalty:none iteration :2
[[ 0.34636964  0.28303896  0.34564975  0.33273381  0.1414206   0.14343421
   0.23222568  0.3694578   0.11979331 -0.13730083  0.28051877  0.04720744
   0.23608029  0.24434868  0.03979101 -0.03449645 -0.07998958  0.05726532
  -0.06196175 -0.11797087  0.40164194  0.35272008  0.38987062  0.3725973
   0.2420742   0.21341227  0.21589479  0.35471664  0.23804865  0.07598705]]
 for penalty:l1 iteration :5
[[ 0.38376144  0.31847653  0.38021393  0.37761412  0.13902705  0.08906467
   0.3116361   0.42769034  0.07161173 -0.17012565  0.35313265  0.0442934
   0.29387375  0.31709502  0.01954113 -0.08724767 -0.06803709  0.02787578
  -0.06943368 -0.16886369  0.45528505  0.44307697  0.44142588  0.43274744
   0.31623088  0.20032659  0.32290752  0.4262394   0.27317331  0.07085504]]
 for penalty:l2 iteration :5
[[ 0.38731817  0.36316944  0.38556597  0.379903    0.17552827  0.13919266
   0.32536381  0.43818923  0.10728577 -0.20040481  0.36124387  0.04538398
   0.30513521  0.30046977 -0.0075254  -0.08150228 -0.10530781  0.05939097
  -0.04696101 -0.22868053  0.44915094  0.45112733  0.43469771  0.41764376
   0.28103829  0.2235578   0.32880656  0.41668911  0.31793707  0.0491977 ]]
 for penalty:none iteration :5
[[ 0.38182557  0.36907094  0.37944693  0.36362404  0.14580082  0.12626525
   0.30885855  0.44705691  0.08933435 -0.19638564  0.33644054  0.02210982
   0.28101901  0.28773488  0.03927148 -0.07534254 -0.11662489  0.07708085
  -0.09465232 -0.23699369  0.45344206  0.47118153  0.43907709  0.41060821
   0.29274929  0.23483244  0.33607566  0.45341343  0.30615813  0.06767348]]
 for penalty:l1 iteration :10
[[ 3.96198087e-01  4.29313939e-01  3.89488140e-01  3.73520465e-01
   1.62168341e-01  2.77885076e-02  3.31437041e-01  4.89165432e-01
   5.36389576e-06 -2.24976803e-01  4.46503432e-01  1.15557048e-05
   3.51934374e-01  3.31731656e-01  4.27418514e-04 -1.27967479e-01
  -4.40104435e-02  6.32273181e-02 -6.56277994e-02 -2.42253911e-01
```

```
   5.06388081e-01  5.78824716e-01  4.76260724e-01  4.42588176e-01
   3.98618906e-01  2.20170725e-01  3.79931010e-01  5.23235520e-01
   4.04723175e-01  2.31771985e-02]]
 for penalty:l2 iteration :10
[[ 0.41941747  0.45906641  0.41591248  0.40829462  0.15201517  0.07948994
   0.36403677  0.5103782   0.04736961 -0.24404907  0.42348776  0.02029938
   0.34029301  0.34363196  0.03064147 -0.18155487 -0.08759354  0.10859683
  -0.07349085 -0.26170407  0.51832857  0.58082694  0.49106679  0.47388565
   0.35350444  0.20813737  0.39276255  0.52096417  0.43538451  0.05148711]]
 for penalty:none iteration :10
[[ 0.44562897  0.45404026  0.44035359  0.43108006  0.1834904   0.07161801
   0.36914814  0.51829503  0.09525705 -0.28954151  0.43681873  0.02381722
   0.35952995  0.3583238  -0.00227082 -0.16151479 -0.06033966  0.1287562
  -0.11350016 -0.29276203  0.54621978  0.6138418   0.52347148  0.49767698
   0.40855762  0.22093002  0.39580888  0.52865719  0.44238544  0.02967586]]
 for penalty:l1 iteration :20
[[ 4.40231065e-01  4.83507648e-01  4.31520036e-01  4.25472065e-01
   1.42543068e-01 -1.07736534e-02  3.84543448e-01  5.79690147e-01
   1.61830094e-02 -2.48705473e-01  5.35336874e-01  0.00000000e+00
   3.97174372e-01  4.01690552e-01  4.66080095e-02 -1.96592686e-01
  -1.30749350e-02  5.30828443e-02 -9.87159256e-02 -3.08594964e-01
   5.74466784e-01  7.15424040e-01  5.33484730e-01  5.05804623e-01
   4.73794135e-01  1.63345385e-01  4.74153052e-01  5.88118714e-01
   5.27715492e-01  9.82069535e-05]]
 for penalty:l2 iteration :20
[[ 0.45320678  0.52238668  0.44675951  0.44972474  0.18082247 -0.01920401
   0.44304546  0.60029309  0.05200331 -0.31702633  0.56299018  0.006834
   0.44257784  0.43514854  0.06507066 -0.26195109 -0.08136733  0.13702584
  -0.17635533 -0.34060727  0.58567172  0.73454268  0.5463635   0.53618722
   0.47633752  0.18480659  0.49566197  0.59197061  0.55606115  0.06760426]]
 for penalty:none iteration :20
[[ 0.4812579   0.54851965  0.47389464  0.46508425  0.21185817 -0.01300231
   0.4792999   0.64689421  0.08059723 -0.34523651  0.57042719  0.03447933
   0.44533364  0.43981714  0.04256684 -0.3033888  -0.06383586  0.1725
  -0.19539964 -0.37932804  0.6330252   0.78914767  0.59426062  0.56749893
   0.49661036  0.18421017  0.51991502  0.64614214  0.57225619  0.03577191]]
 for penalty:l1 iteration :50
[[ 4.07802743e-01  4.70901319e-01  3.92249050e-01  3.97854700e-01
   5.83361077e-02 -6.84250529e-02  5.08376664e-01  7.51027775e-01
  -1.31203743e-05 -2.31467336e-01  7.78258347e-01  0.00000000e+00
   5.12561124e-01  4.90102309e-01  3.35801518e-02 -2.97584947e-01
   0.00000000e+00  6.42312377e-02 -1.82266287e-01 -3.61662175e-01
   6.72013201e-01  9.17622286e-01  5.80426760e-01  5.81364368e-01
   5.77159439e-01  0.00000000e+00  6.16856900e-01  7.00986417e-01
   6.95174159e-01  0.00000000e+00]]
 for penalty:l2 iteration :50
[[ 0.44303254  0.54755447  0.4328543   0.44511794  0.15235449 -0.19709656
   0.58352925  0.73021129 -0.02786279 -0.3556872   0.78642064 -0.02437329
   0.58427343  0.56178474  0.12734249 -0.44032951 -0.04426773  0.23467365
  -0.27890388 -0.4603874   0.67345329  0.91743104  0.60301318  0.61066675
   0.57040928  0.11717825  0.69809937  0.71178847  0.79475155  0.07464551]]
 for penalty:none iteration :50
[[ 0.49736723  0.60050208  0.48436341  0.51241812  0.17646815 -0.25905724
   0.65912335  0.82611253 -0.01418145 -0.36740842  0.87106548 -0.01307887
   0.64643991  0.63211252  0.13566394 -0.5372055  -0.01788823  0.22316128
  -0.32604859 -0.4878151   0.75634013  1.03374472  0.6782791   0.69757944
   0.64854579  0.09284599  0.76250045  0.79270128  0.88354061  0.06307404]]
 for penalty:l1 iteration :100
[[ 0.34673037  0.39013184  0.31975111  0.34007073  0.         -0.14422924
   0.51596789  0.88928443  0.         -0.24512859  1.00425995 -0.00118011
   0.57647357  0.55766497  0.08603144 -0.38698624  0.          0.09192926
  -0.26978916 -0.37586444  0.73879984  1.04496375  0.58859516  0.61304358
   0.58652575  0.          0.70540062  0.79782131  0.80094965  0.        ]]
 for penalty:l2 iteration :100
[[ 0.40773037  0.50351243  0.39651623  0.42629377  0.11180759 -0.36125847
   0.69663402  0.84946515 -0.07820985 -0.32840243  0.94861132 -0.08788116
   0.68033605  0.65674329  0.17889198 -0.52975795 -0.03296619  0.31328743
  -0.36249553 -0.53089677  0.69788103  1.05381051  0.59555572  0.63627551
   0.58105029  0.02788771  0.85165356  0.76657564  0.96172491  0.08620203]]
```

```
 for penalty:none iteration :100
[[ 0.45597881  0.60078908  0.43793606  0.48479447  0.13265761 -0.53134129
   0.82267167  1.01097332 -0.11331906 -0.38163552  1.18491151 -0.13059215
   0.814603    0.81699166  0.17775634 -0.72368638  0.04104597  0.40030454
  -0.47304065 -0.61282205  0.83981487  1.27748661  0.70283242  0.76989035
   0.74410104 -0.02593649  1.02538436  0.93400569  1.19681534  0.07729285]]
 for penalty:l1 iteration :500
[[ 0.09942307  0.          0.          0.04756677  0.         -0.28736338
   0.3936746   1.63038357 -0.0395516  -0.14574306  1.74672597 -0.18748129
   0.24451607  0.51952105  0.33650585 -0.42395256  0.          0.
  -0.39581297 -0.43395239  1.09434836  1.52538061  0.51265529  0.6969349
   0.14729482  0.          0.86241045  0.7920336   0.94297659  0.        ]]
 for penalty:l2 iteration :500
[[ 0.34632665  0.37593378  0.33215519  0.38072418  0.06697511 -0.61452759
   0.79834015  1.07957626 -0.18307293 -0.17284692  1.13252735 -0.17570577
   0.68716378  0.78960385  0.19360474 -0.53876739 -0.05320761  0.4611827
  -0.4782114  -0.58505165  0.73161406  1.25926256  0.54701638  0.67650765
   0.52108261 -0.12780369  0.97321302  0.79245267  1.19078126  0.08734025]]
 for penalty:none iteration :500
[[ 0.15825705  0.13183933  0.11697332  0.30636312 -0.01937243 -1.69493282
   1.30402939  1.98165392 -0.67607374  0.0985928   2.22831296 -0.43161465
   1.13253674  1.53398264  0.28653934 -0.72643076 -0.225322    0.97175132
  -0.83651605 -1.02728228  1.12916501  2.29472773  0.63525771  1.09596056
   0.64666011 -0.60473457  1.80938616  1.33323996  2.29431496  0.08096472]]
 for penalty:l1 iteration :1000
[[ 0.          0.          0.          0.          0.         -0.36682026
   0.06761165  2.17993447 -0.03879955 -0.05006121  2.21331147 -0.25453858
   0.          0.20951252  0.37734838 -0.35459297  0.          0.
  -0.45424632 -0.48182749  1.36315278  1.58902282  0.34668349  0.6730379
   0.02692501  0.          1.06868851  0.58762021  0.98973026  0.        ]]
 for penalty:l2 iteration :1000
[[ 0.34639001  0.37610794  0.33208593  0.37980665  0.06685985 -0.61406818
   0.79777856  1.07962569 -0.18332651 -0.17276324  1.13363819 -0.17550361
   0.68672152  0.7879627   0.19351172 -0.53841164 -0.0534161   0.46142304
  -0.47860888 -0.5848871   0.73212821  1.25905338  0.54745748  0.67646847
   0.521094   -0.12826589  0.97366871  0.79277043  1.19130537  0.08675133]]
 for penalty:none iteration :1000
[[-0.07882635 -0.24498037 -0.13768613  0.14231281 -0.03167762 -2.60117816
   1.64039222  2.92240727 -1.05215044  0.60767148  2.98600451 -0.59782787
   1.14886751  2.04957068  0.31126595 -0.31466219 -0.73844323  1.42676336
  -1.07581279 -1.38631151  1.28951001  2.92226389  0.43060565  1.27314487
   0.37501298 -1.0472406   2.31729758  1.45818651  3.00350223  0.01099899]]
```

Conclusion:

Basic idea about logistic regression was obtained.The accuracy and cionfusion matrix were obtained.