CAPSTONE PROJECT 2

PROJECT PROPOSAL

Title Of Project

Credit Card Fraud Detection

Breif on the Project

The Credit Card Fraud Detection project aims to develop a machine learning system that can automatically identify fraudulent credit card transactions. By analyzing transaction data, including transaction amount, location, and user behavior, the system detects anomalies and patterns indicative of fraud. This project helps financial institutions protect customers from unauthorized transactions and minimize financial losses due to fraudulent activities.

Deliverables of the Project

The deliverables for the Credit Card Fraud Detection project are training a machine learning model capable of detecting fraudulent credit card transactions accurately

References

Data set: Source https://www.kaggle.com/datasets/arslanali4343/credit-card-cheating-detection-cccd/data

Software used: Jupyter notebook(Python)

Individual Deteils

Name : Nanda Kishore Sai C

Email id : nandakishoresaic@gmail.com

Phone no : +91 9618208987

CODE

In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
```

In [2]:
```
cc= pd.read_csv('creditcard.csv')
```

In [3]:
```
cc.sample(10)
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
185442	126662.0	-2.288573	-0.559528	0.915183	-1.636016	-0.725922	0.346375	-1.044857	-0.427654	-0.375932	...	0.724005	-0.092057	-0.902260	-0.376783	-0.377529	-0.395773	0.015098	-0.425850	104.08	0
28992	35271.0	1.023741	-0.129899	-0.106309	0.643756	-0.371161	-1.220651	0.560901	-0.324336	-0.114157	...	-0.309345	-1.260786	0.036201	0.376469	0.216880	0.125637	-0.093186	0.031675	140.80	0
67851	52722.0	1.020509	-0.171286	0.577751	0.989772	-0.744944	-0.845530	0.163884	-0.222829	0.321929	...	-0.143105	-0.418597	-0.041346	0.780861	0.398879	0.235710	-0.035757	0.034859	104.73	0
158496	111341.0	1.811944	-0.587333	-1.269727	0.102317	0.554379	1.096034	-0.421841	0.305310	2.331001	...	-0.206089	-0.194654	0.186838	-1.708168	-0.309159	-0.017072	-0.024308	-0.082538	60.00	0
141339	84246.0	1.295971	-0.789478	0.941957	-0.658280	-1.381149	-0.270316	-1.103493	0.089308	-0.346611	...	0.415853	1.132351	-0.084432	0.107912	0.334598	-0.020113	0.048226	0.024751	29.99	0
177028	123011.0	1.814272	-1.242594	-0.936168	-0.701742	0.023657	1.861587	-1.060943	0.539884	-0.123995	...	-0.449405	-0.843300	0.414105	-1.028848	-0.706093	-0.590623	0.081192	-0.031287	97.94	0
46723	42901.0	-1.314087	-0.208476	1.837879	-1.722664	-1.001906	-0.312185	-0.972617	0.862320	-1.082684	...	0.579729	1.297096	-0.149255	0.227068	-0.020199	-0.181202	0.227491	0.029201	24.99	0
88785	62290.0	-0.237403	-4.270558	0.341060	-0.349177	-2.736765	1.345044	-0.847679	0.279936	-0.642250	...	0.080128	-0.829645	-0.678028	-0.280601	-0.023688	-0.263114	-0.050677	0.153483	835.00	0
174072	121754.0	2.162261	-1.605107	-0.292134	-1.537620	-1.795396	-0.492119	-1.508029	0.058044	-1.092701	...	-0.338793	-0.620562	0.463724	-0.077837	-0.707151	-0.502446	0.027246	-0.042797	35.99	0
209704	137668.0	-0.467776	1.022628	1.024089	1.219077	-0.204212	0.201866	0.101760	0.606401	0.087010	...	-0.352801	-0.929642	0.294259	0.996672	-0.856743	-1.057057	0.134449	0.160213	11.54	0

10 rows × 31 columns

In [4]:
```
cc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [5]:
```
cc.isnull().sum()
```

Out[5]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [6]:
```
#distribution of legitimate transactions and fraudulent transactions
cc['Class'].value_counts()
```

Out[6]:
```
0    284315
1       492
Name: Class, dtype: int64
```

We can see that the data is imbalanced

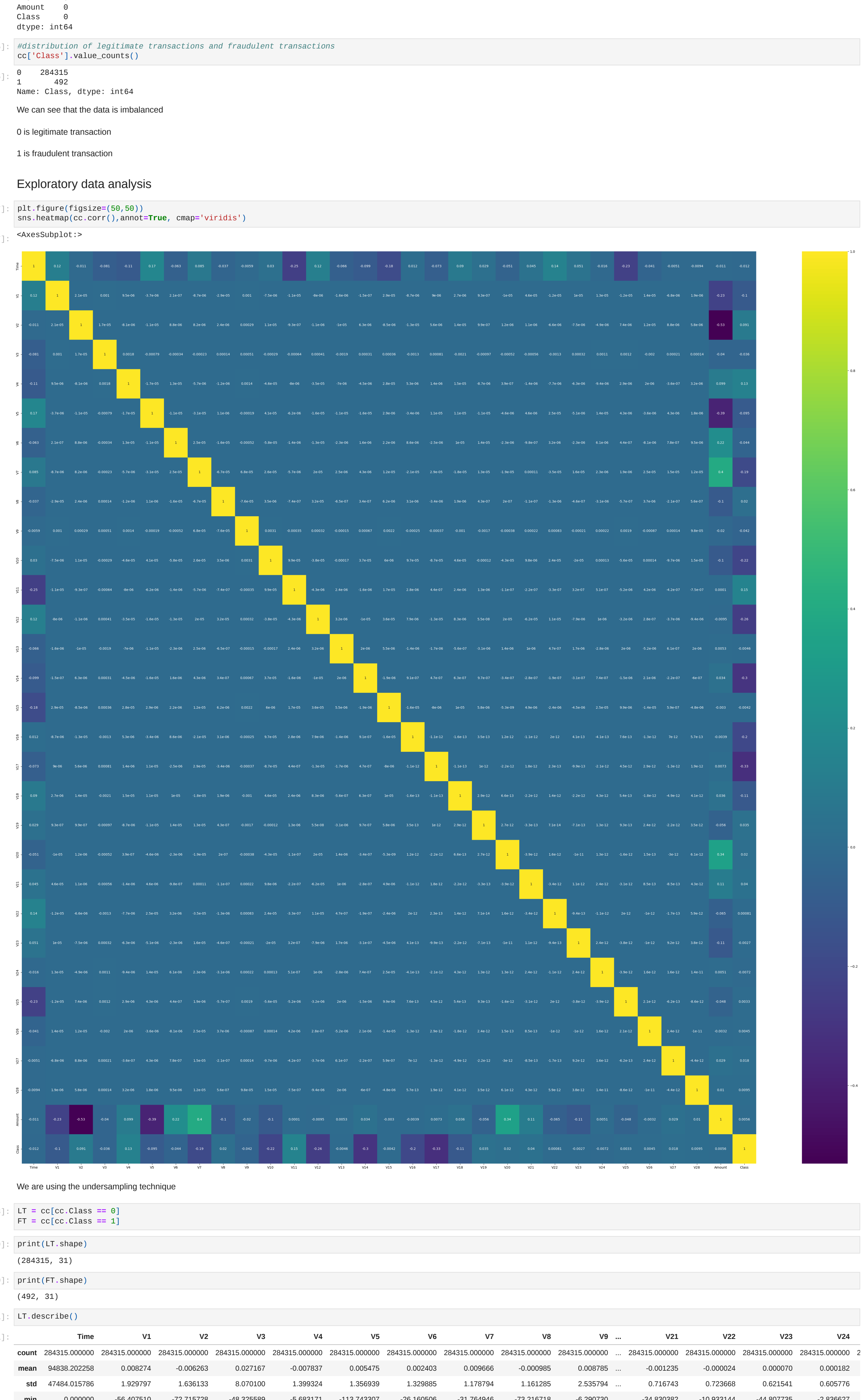
0 is legitimate transaction

1 is fraudulent transaction

Exploratory data analysis

In [7]:
```
plt.figure(figsize=(50,50))
sns.heatmap(cc.corr(),annot=True, cmap='viridis')
```

Out[7]:
```
<AxesSubplot:>
```



We are using the undersampling technique

In [8]:
```
LT = cc[cc.Class == 0]
FT = cc[cc.Class == 1]
```

In [9]:
```
print(LT.shape)
```

```
(284315, 31)
```

In [10]:
```
print(FT.shape)
```

```
(492, 31)
```

In [11]:
```
LT.describe()
```

Out[11]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	
count	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	...	284315.000000	284315.000000	284315.000000	284315.000000	2
mean	94838.202258	0.008274	-0.006263	0.027167	-0.007837	0.005475	0.002403	0.009666	-0.000985	0.008785	...	-0.001235	-0.000024	0.000070	0.000182	
std	47484.015786	1.929797	1.636133	8.070100	1.399324	1.356939	1.329885	1.178794	1.161285	2.535794	...	0.716743	0.723668	0.621541	0.605776	
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307	-26.160506	-31.764946	-73.216718	-6.290730	...	-34.830382	-10.933144	-44.807735	-2.836627	
25%	54230.000000	-0.917508	-0.599442	-0.884541	-0.850031	-0.689335	-0.766846	-0.551411	-0.208633	-0.640388	...	-0.228509	-0.542403	-0.161702	-0.354425	
50%	84711.000000	0.019958	0.064063	0.182131	-0.022388	-0.053390	-0.273091	0.041139	0.022041	-0.049869	...	-0.029821	0.006736	-0.011147	0.041082	
75%	139333.000000	1.316209	0.800446	1.028386	0.737654	0.612181	0.399581	0.571027	0.326205	0.598230	...	0.185626	0.528407	0.147522	0.439869	
max	172792.000000	2.454930	18.902453	4232.000000	16.875344	34.801666	73.301626	120.589494	18.709255	1221.000000	...	22.614889	10.503090	22.528412	4.584549	

8 rows × 31 columns

In [12]:
```
FT.describe()
```

Out[12]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	
count	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	...	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	492.00
mean	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.713588	0.014049	-0.040308	-0.105130	0.041449	0.051648	0.170575	0.075
std	47835.365138	6.783687	4.291216	7.110937	2.873318	5.372468	1.858124	7.206773	6.797831	2.500896	...	3.869304	1.494602	1.579642	0.515577	0.797205	0.471679	1.376766	0.547
min	406.000000	-30.552380	-8.402154	-31.103685	-1.313275	-22.105532	-6.406267	-43.557242	-41.044261	-13.434066	...	-22.797604	-8.887017	-19.254328	-2.028024	-4.781606	-1.152671	-7.263482	-1.869
25%	41241.500000	-6.036063	1.188226	-8.643489	2.373050	-4.792835	-2.501511	-7.965295	-0.195336	-3.872383	...	0.041787	-0.533764	-0.342175	-0.436809	-0.314348	-0.259416	-0.020025	-0.108
50%	75568.500000	-2.342497	2.717869	-5.075257	4.177147	-1.522962	-1.424616	-3.034402	0.621508	-2.208768	...	0.592146	0.048434	-0.073135	-0.060795	0.088371	0.004321	0.394926	0.146
75%	128483.000000	-0.419200	4.971257	-2.276185	6.348729	0.214562	-0.413216	-0.945954	1.764879	-0.787850	...	1.244611	0.617474	0.308378	0.285328	0.456515	0.396733	0.826029	0.381
max	170348.000000	2.132386	22.057729	2.250210	12.114672	11.095089	6.474115	5.802537	20.007208	3.353525	...	27.202839	8.361985	5.466230	1.091435	2.208209	2.745261	3.052358	1.779

8 rows × 31 columns

In [13]:
```
LT_2 = LT.sample(n=492)
```

In [14]:
```
newdata = pd.concat([LT_2, FT],axis=0)
```

In [15]:
```
newdata.sample(10)
```

Out[15]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
150601	93742.0	-3.291125	4.401194	-8.394212	4.453580	-4.790055	-4.240182	-9.219001	1.974030	-2.912943	...	2.102343	0.597370	-0.328086	0.445752	0.585281	-0.399005	2.116004	1.050744	1.00	1
282654	171050.0	-0.750947	1.186439	-0.510031	-0.225151	0.163673	-1.022722	0.912529	0.273084	-0.034971	...	0.099051	0.369633	-0.069414	-0.117385	-0.140902	0.584685	0.225555	0.204921	79.00	0
38457	39396.0	-14.839074	9.628396	-9.870326	-0.010024	-9.079928	-0.788339	-6.827138	7.419759	0.791993	...	1.606483	-1.696751	0.768079	-0.166424	1.362359	0.276440	-1.575058	-0.053105	166.14	0
41569	40742.0	-2.377533	0.520539	-8.094139	8.005351	2.640750	-3.381586	-1.934372	0.562322	-3.104027	...	0.148284	0.721100	2.661291	-0.508620	-0.401657	0.587611	0.500326	0.551760	1.00	1
263274	160870.0	-0.644278	5.002352	-8.252739	7.756915	-0.216267	-2.751496	-3.358857	1.406268	-4.403852	...	0.587728	-0.605759	0.033746	-0.756170	-0.008172	0.532772	0.663970	0.192067	0.77	1
43061	41353.0	-15.020981	8.075240	-16.298091	5.664820	-11.918153	-4.246957	-14.716668	9.435084	-6.795398	...	2.525115	-0.832074	-0.186117	0.429781	0.697103	0.056031	-1.310888	-0.707403	34.12	1
86155	61108.0	-2.756007	0.683821	-1.390169	1.501887	-1.165614	-0.131207	-1.478741	-0.246922	-0.100523	...	0.320474	0.611027	0.174864	-0.502151	-0.174713	1.179242	-1.166315	0.821215	101.50	1
23234	32649.0	0.615798	-1.047711	0.306984	0.363691	-1.099156	-0.861050	0.263381	-0.250433	0.240689	...	-0.038624	-0.890292	-0.123255	0.434377	-0.115927	0.707293	-0.129494	0.073062	342.50	0
140786	83934.0	-0.433222	2.428379	-3.996454	4.871299	-1.796308	-0.586868	-4.654543	1.285230	-2.743539	...	0.713559	-0.408954	-0.320890	-0.804230	0.962852	0.199558	1.094533	0.541148	1.00	1
252988	156050.0	-1.448751	2.042931	-1.082058	-1.246040	1.206367	-0.254818	1.259731	-0.324888	1.451941	...	-0.707690	-0.862826	-0.013075	-0.031462	-0.087419	0.048107	0.165287	-0.411996	8.98	0

10 rows × 31 columns

In [16]:
```
newdata['Class'].value_counts()
```

Out[16]:
```
0    492
1    492
Name: Class, dtype: int64
```

In [17]:
```
x = newdata.drop(columns = 'Class', axis = 1)
y = newdata['Class']
```

MODEL

In [18]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split (x,y,test_size = 0.2,random_state = 2)
```

LOGISTIC_REGRESSION_MODEL

In [19]:
```
from sklearn.linear_model import LogisticRegression
```

In [20]:
```
Lreg=LogisticRegression()
```

In [21]:
```
Lreg.fit(x_train, y_train)
```

Out[21]:
```
LogisticRegression()
```

In [22]:
```
y_pred = Lreg.predict(x_test)
y_pred
```

Out[22]:
```
array([0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0],
      dtype=int64)
```

In [23]:
```
from sklearn.metrics import precision_score,accuracy_score,recall_score,confusion_matrix
from sklearn.metrics import classification_report
from sklearn import metrics
```

In [24]:
```
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
confusion_matrix
```

Out[24]:
```
array([[91,  4],
       [12, 90]], dtype=int64)
```

In [25]:
```
print("Accuracy : ", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy :  0.9187817258883249
```

In [26]:
```
report=classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.88      0.96      0.92        95
           1       0.96      0.88      0.92       102

    accuracy                           0.92       197
   macro avg       0.92      0.92      0.92       197
weighted avg       0.92      0.92      0.92       197
```

DECISON_TREE_MODEL

In [27]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.model_selection import train_test_split, GridSearchCV
```

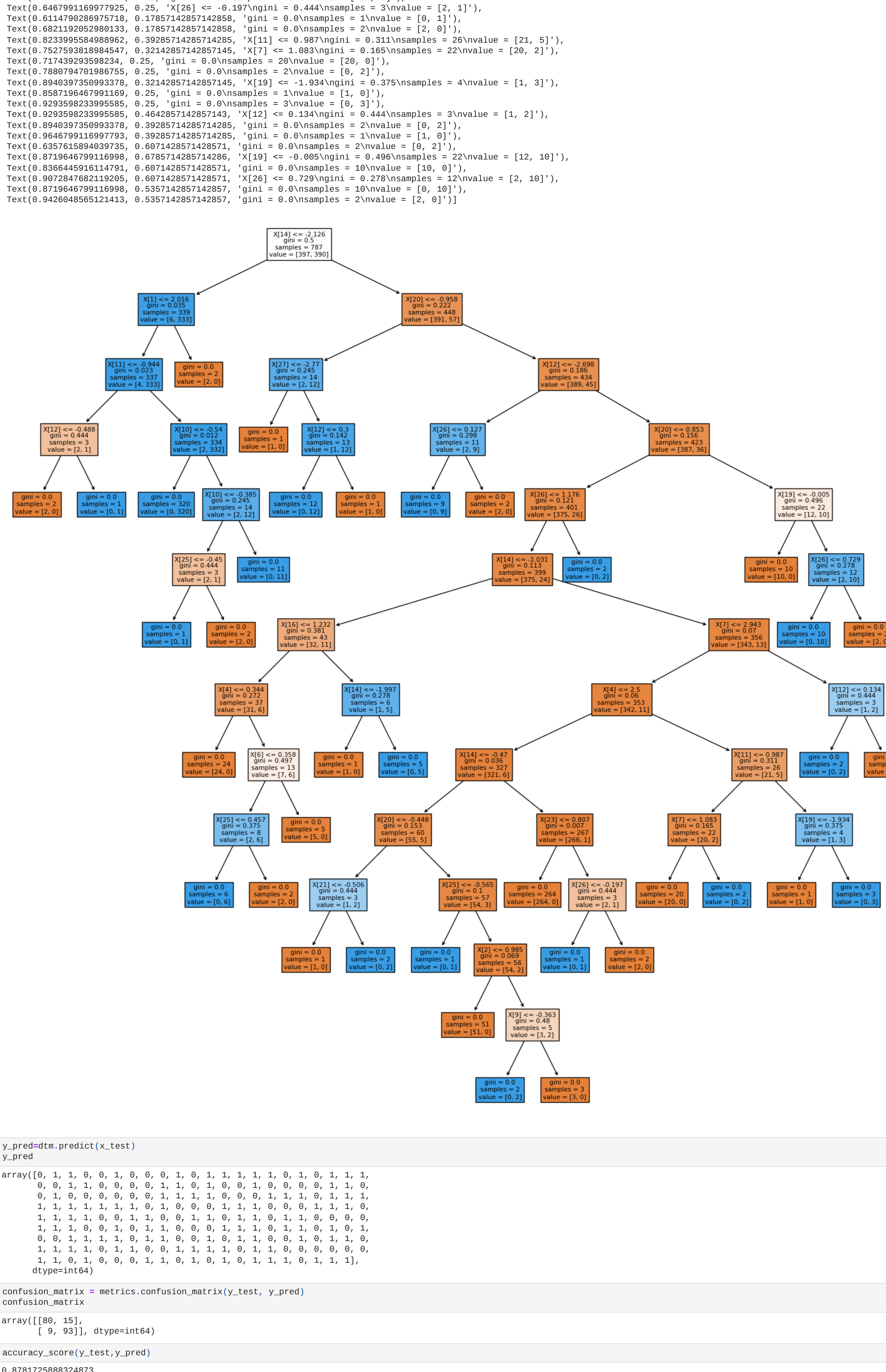
In [29]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=2)
```

In [30]:
```
dtm=DecisionTreeClassifier()
dtm.fit(x_train,y_train)
```

Out[30]:
```
DecisionTreeClassifier()
```

In [31]:
```
plt.figure(figsize=[20,20])
tree.plot_tree(dtm, filled=True)
```

Out[31]:
```
[Text(0.3216059602649007, 0.9642857142857143, 'X[14] <= -2.126\ngini = 0.5\nsamples = 787\nvalue = [397, 390]'),
 Text(0.17660044150110377, 0.8928571428571429, 'X[1] <= 2.016\ngini = 0.035\nsamples = 339\nvalue = [6, 333]'),
 Text(0.141280353200883, 0.8214285714285714, 'X[11] <= -0.944\ngini = 0.023\nsamples = 337\nvalue = [4, 333]'),
 Text(0.0706401766004415, 0.75, 'X[12] <= -0.488\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
 Text(0.03532008830022075, 0.6785714285714286, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.10596026490066225, 0.6785714285714286, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.2119205298013245, 0.75, 'X[10] <= -0.54\ngini = 0.012\nsamples = 334\nvalue = [2, 332]'),
 Text(0.17660044150110377, 0.6785714285714286, 'gini = 0.0\nsamples = 320\nvalue = [0, 320]'),
 Text(0.24724061810154527, 0.6785714285714286, 'X[10] <= -0.385\ngini = 0.245\nsamples = 14\nvalue = [2, 12]'),
 Text(0.2119205298013245, 0.6071428571428571, 'X[25] <= -0.45\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
 Text(0.17660044150110377, 0.5357142857142857, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.24724061810154527, 0.5357142857142857, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.282560706401766, 0.6071428571428571, 'gini = 0.0\nsamples = 11\nvalue = [0, 11]'),
 Text(0.2119205298013245, 0.8214285714285714, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.4666114790286976, 0.8928571428571429, 'X[20] <= -0.958\ngini = 0.222\nsamples = 448\nvalue = [391, 57]'),
 Text(0.31788079470198677, 0.8214285714285714, 'X[27] <= -2.77\ngini = 0.245\nsamples = 14\nvalue = [2, 12]'),
 Text(0.282560706401766, 0.75, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.35320088300220753, 0.75, 'X[12] <= 0.3\ngini = 0.142\nsamples = 13\nvalue = [1, 12]'),
 Text(0.31788079470198677, 0.6785714285714286, 'gini = 0.0\nsamples = 12\nvalue = [0, 12]'),
 Text(0.38852097130242824, 0.6785714285714286, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.6153421633554084, 0.8214285714285714, 'X[12] <= -2.698\ngini = 0.186\nsamples = 434\nvalue = [389, 45]'),
 Text(0.49448123620309054, 0.75, 'X[26] <= 0.127\ngini = 0.298\nsamples = 11\nvalue = [2, 9]'),
 Text(0.45916114790286977, 0.6785714285714286, 'gini = 0.0\nsamples = 9\nvalue = [0, 9]'),
 Text(0.5298013245033113, 0.6785714285714286, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.7362030905077263, 0.75, 'X[20] <= 0.853\ngini = 0.156\nsamples = 423\nvalue = [387, 36]'),
 Text(0.6004415011037527, 0.6785714285714286, 'X[26] <= 1.176\ngini = 0.121\nsamples = 401\nvalue = [375, 26]'),
 Text(0.565121412803532, 0.6071428571428571, 'X[14] <= -1.031\ngini = 0.113\nsamples = 399\nvalue = [375, 24]'),
 Text(0.32891832229580575, 0.5357142857142857, 'X[16] <= 1.232\ngini = 0.381\nsamples = 43\nvalue = [32, 11]'),
 Text(0.2582781456953642, 0.4642857142857143, 'X[4] <= 0.344\ngini = 0.272\nsamples = 37\nvalue = [31, 6]'),
 Text(0.2229580573951435, 0.39285714285714285, 'gini = 0.0\nsamples = 24\nvalue = [24, 0]'),
 Text(0.293598233995585, 0.39285714285714285, 'X[6] <= 0.358\ngini = 0.497\nsamples = 13\nvalue = [7, 6]'),
 Text(0.2582781456953642, 0.32142857142857145, 'X[25] <= 0.457\ngini = 0.375\nsamples = 8\nvalue = [2, 6]'),
 Text(0.2229580573951435, 0.25, 'gini = 0.0\nsamples = 6\nvalue = [0, 6]'),
 Text(0.293598233995585, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.32891832229580575, 0.32142857142857145, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(0.3995584988962472, 0.4642857142857143, 'X[14] <= -1.997\ngini = 0.278\nsamples = 6\nvalue = [1, 5]'),
 Text(0.36423841059602646, 0.39285714285714285, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.434878587196468, 0.39285714285714285, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.8013245033112583, 0.5357142857142857, 'X[7] <= 2.943\ngini = 0.07\nsamples = 356\nvalue = [343, 13]'),
 Text(0.673289183222958, 0.4642857142857143, 'X[4] <= 2.5\ngini = 0.06\nsamples = 353\nvalue = [342, 11]'),
 Text(0.5231788079470199, 0.39285714285714285, 'X[14] <= -0.47\ngini = 0.036\nsamples = 327\nvalue = [321, 6]'),
 Text(0.434878587196468, 0.32142857142857145, 'X[20] <= -0.448\ngini = 0.153\nsamples = 60\nvalue = [55, 5]'),
 Text(0.36423841059602646, 0.25, 'X[21] <= -0.506\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.32891832229580575, 0.17857142857142858, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.3995584988962472, 0.17857142857142858, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.5055187637969095, 0.25, 'X[25] <= -0.565\ngini = 0.1\nsamples = 57\nvalue = [54, 3]'),
 Text(0.47019867549668876, 0.17857142857142858, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.5408388520971302, 0.17857142857142858, 'X[2] <= 0.985\ngini = 0.069\nsamples = 56\nvalue = [54, 2]'),
 Text(0.5055187637969095, 0.10714285714285714, 'gini = 0.0\nsamples = 51\nvalue = [51, 0]'),
 Text(0.5761589403973509, 0.10714285714285714, 'X[9] <= -0.363\ngini = 0.48\nsamples = 5\nvalue = [3, 2]'),
 Text(0.5408388520971302, 0.03571428571428571, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.6114790286975718, 0.03571428571428571, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.6114790286975718, 0.32142857142857145, 'X[23] <= 0.807\ngini = 0.007\nsamples = 267\nvalue = [266, 1]'),
 Text(0.5761589403973509, 0.25, 'gini = 0.0\nsamples = 264\nvalue = [264, 0]'),
 Text(0.6467991169977925, 0.25, 'X[26] <= -0.197\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
 Text(0.6114790286975718, 0.17857142857142858, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.6821192052980133, 0.17857142857142858, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.8233995584988962, 0.39285714285714285, 'X[11] <= 0.987\ngini = 0.311\nsamples = 26\nvalue = [21, 5]'),
 Text(0.7527593818984547, 0.32142857142857145, 'X[7] <= 1.083\ngini = 0.165\nsamples = 22\nvalue = [20, 2]'),
 Text(0.717439293598234, 0.25, 'gini = 0.0\nsamples = 20\nvalue = [20, 0]'),
 Text(0.7880794701986755, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.8940397350993378, 0.32142857142857145, 'X[19] <= -1.934\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
 Text(0.8587196467991169, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.9293598233995585, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.9293598233995585, 0.4642857142857143, 'X[12] <= 0.134\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.8940397350993378, 0.39285714285714285, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.9646799116997793, 0.39285714285714285, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.6357615894039735, 0.6071428571428571, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.8719646799116998, 0.6785714285714286, 'X[19] <= -0.005\ngini = 0.496\nsamples = 22\nvalue = [12, 10]'),
 Text(0.8366445916114791, 0.6071428571428571, 'gini = 0.0\nsamples = 10\nvalue = [10, 0]'),
 Text(0.9072847682119205, 0.6071428571428571, 'X[26] <= 0.729\ngini = 0.278\nsamples = 12\nvalue = [2, 10]'),
 Text(0.8719646799116998, 0.5357142857142857, 'gini = 0.0\nsamples = 10\nvalue = [0, 10]'),
 Text(0.9426048565121413, 0.5357142857142857, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]')]
```



In [32]:
```
y_pred=dtm.predict(x_test)
y_pred
```

Out[32]:
```
array([0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1],
      dtype=int64)
```

In [33]:
```
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
confusion_matrix
```

Out[33]:
```
array([[80, 15],
       [ 9, 93]], dtype=int64)
```

In [34]:
```
accuracy_score(y_test,y_pred)
```

Out[34]:
```
0.8781725888324873
```

In [35]:
```
report=classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.90      0.84      0.87        95
           1       0.86      0.91      0.89       102

    accuracy                           0.88       197
   macro avg       0.88      0.88      0.88       197
weighted avg       0.88      0.88      0.88       197
```

RANDOM_FOREST

In [36]:
```
from sklearn.ensemble import RandomForestClassifier
np.random.seed(0)
```

In [37]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=2)
```

In [38]:
```
rfm=RandomForestClassifier()
rfm.fit(x_train,y_train)
```

Out[38]:
```
RandomForestClassifier()
```

In [39]:
```
y_pred=rfm.predict(x_test)
```

In [40]:
```
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
confusion_matrix
```

Out[40]:
```
array([[91,  4],
       [ 9, 93]], dtype=int64)
```

In [41]:
```
accuracy_score(y_test,y_pred)
```

Out[41]:
```
0.934010152284264
```

In [42]:
```
report=classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.91      0.96      0.93        95
           1       0.96      0.91      0.93       102

    accuracy                           0.93       197
   macro avg       0.93      0.93      0.93       197
weighted avg       0.94      0.93      0.93       197
```

CONCLUSION

From the machine learning models that we have used we have got the following accuracy

Logistics Regression: 91.87%

Decision tree: 87.81%

Random Forest: 93.4%