Credit Card Fraud Detection

In [105]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
log_model = LogisticRegression(solver='lbfgs', max_iter=100000)
from sklearn.metrics import accuracy_score
```

In [72]:

```
# loading the dataset to a Pandas DataFrame
credit_card_data = pd.read_csv('/content/Credit_Card_Fraud_Detection.csv')
```

In [73]:

```
# first 5 rows of the dataset
credit_card_data.head()
```

Out[73]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	0.6178
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.0652
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.0660
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.1782
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.5381

In [74]:

```
credit_card_data.tail()
```

Out[74]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
45641	42436	-2.481639	-2.439949	0.363642	1.216827	2.572442	-1.264220	-0.443652	0.075853	0.073188	0.097421	-1.339838
45642	42436	1.223475	0.014944	0.471312	-0.038410	-0.566793	-0.867970	-0.058213	-0.144080	0.164904	-0.248839	0.363145
45643	42436	1.258657	0.421016	0.325437	0.684259	-0.292529	-1.052786	0.145228	-0.253567	-0.100521	-0.308072	0.083964
45644	42437	-0.500147	1.000770	1.809639	-0.114551	0.333865	-0.577076	1.062325	-0.513050	-0.048285	0.314582	0.369958
45645	42437	-0.652459	0.177290	1.955607	-1.879724	-0.368457	NaN	NaN	NaN	NaN	NaN	NaN

In [48]:

```
# dataset informations
credit_card_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 45646 entries, 0 to 45645
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Time    45646 non-null  int64  
 1   V1      45646 non-null  float64
 2   V2      45646 non-null  float64
 3   V3      45646 non-null  float64
 4   V4      45646 non-null  float64
 5   V5      45646 non-null  float64
 6   V6      45645 non-null  float64
 7   V7      45645 non-null  float64
 8   V8      45645 non-null  float64
 9   V9      45645 non-null  float64
 10  V10     45645 non-null  float64
 11  V11     45645 non-null  float64
 12  V12     45645 non-null  float64
 13  V13     45645 non-null  float64
 14  V14     45645 non-null  float64
 15  V15     45645 non-null  float64
 16  V16     45645 non-null  float64
 17  V17     45645 non-null  float64
 18  V18     45645 non-null  float64
 19  V19     45645 non-null  float64
 20  V20     45645 non-null  float64
 21  V21     45645 non-null  float64
 22  V22     45645 non-null  float64
 23  V23     45645 non-null  float64
 24  V24     45645 non-null  float64
 25  V25     45645 non-null  float64
 26  V26     45645 non-null  float64
 27  V27     45645 non-null  float64
 28  V28     45645 non-null  float64
 29  Amount  45645 non-null  float64
 30  Class   45645 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 10.8 MB
```

In [75]:

```
# checking the number of missing values in each column
credit_card_data.isnull().sum()
```

Out[75]:

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        1
V7        1
V8        1
V9        1
V10       1
V11       1
V12       1
V13       1
V14       1
V15       1
V16       1
V17       1
V18       1
V19       1
V20       1
V21       1
V22       1
V23       1
V24       1
V25       1
V26       1
```

```
V27         1
V28         1
Amount      1
Class       1
dtype: int64
```

In [76]:

```
# distribution of legit transactions & fraudulent transactions
credit_card_data['Class'].value_counts()
```

Out[76]:

```
0.0    45503
1.0      142
Name: Class, dtype: int64
```

This Dataset is highly unblanced

0 --> Normal Transaction

1 --> fraudulent transaction

In [77]:

```
# separating the data for analysis
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

In [78]:

```
print(legit.shape)
print(fraud.shape)
```

```
(45503, 31)
(142, 31)
```

In [80]:

```
# statistical measures of the data
legit.Amount.describe()
```

Out[80]:

```
count    45503.000000
mean        90.808470
std        240.322652
min          0.000000
25%          7.580000
50%         24.990000
75%         82.360000
max       7879.420000
Name: Amount, dtype: float64
```

In [79]:

```
fraud.Amount.describe()
```

Out[79]:

```
count     142.000000
mean       97.592183
std       233.185192
min         0.000000
25%         1.000000
50%         8.370000
75%        99.990000
max      1809.680000
Name: Amount, dtype: float64
```

In [81]:

```python
# compare the values for both transactions
credit_card_data.groupby('Class').mean()
```

Out[81]:

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	
0.0	27549.332857	-0.21371	0.011525	0.732168	0.173212	-0.230885	0.106122	-0.092498	0.041321	0.169777	-0.042999	0.37
1.0	26193.556338	-7.87188	5.609155	-10.671851	6.067972	-5.862266	-2.315720	-8.269674	3.901566	-3.650345	-7.653117	5.48

Under-Sampling

Build a sample dataset containing similar distribution of normal transactions and Fraudulent Transactions

Number of Fraudulent Transactions --> 492

In [83]:

```python
legit_sample = legit.sample(n=492)
```

Concatenating two DataFrames

In [84]:

```python
new_dataset = pd.concat([legit_sample, fraud], axis=0)
```

In [85]:

```python
new_dataset.head()
```

Out[85]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
13309	23451	-0.808883	1.056344	0.202809	0.839825	1.362333	5.599274	-2.230701	-2.845794	2.223159	-0.463253	-0.143944
27857	34736	1.267014	-0.716659	0.679277	-0.746548	-1.168415	-0.308703	-0.887086	0.155982	-0.802779	0.764146	1.689626
15996	27437	-1.156895	-0.537961	1.358034	1.096137	2.082972	-0.985832	-0.022456	-0.150626	-0.509194	0.054979	-0.402237
19357	30210	-0.436689	0.071842	2.543647	-0.000031	-0.558228	1.474524	-0.685809	0.584585	0.971230	-0.717426	-0.780312
20076	30769	-0.732469	1.540226	0.277131	1.025435	-0.702923	-0.837118	0.083999	0.638158	-0.793751	-0.493672	-0.795968

In [86]:

```python
new_dataset.tail()
```

Out[86]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	
44091	41791	-7.222731	6.155773	-10.826460	4.180779	-6.123555	-3.114136	-6.895112	5.161516	-2.516477	-6.403371	3.186
44223	41851	-19.139733	9.286847	-20.134992	7.818673	-15.652208	-1.668348	-21.340478	0.641900	-8.550110	-16.649628	4.818
44270	41870	-20.906908	9.843153	-19.947726	6.155789	-15.142013	-2.239566	-21.234463	1.151795	-8.739670	-18.271168	4.677

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	
44556	41991	-4.566342	3.353451	-4.572028	3.616119	-2.493138	1.0900[illegible]	-5.551433	0.447783	2.4244[illegible]	-5.699922	3.586
45203	42247	-2.524012	2.098152	-4.946075	6.456588	3.173921	-3.058806	-0.184710	-0.390420	-3.649812	-4.077585	4.389

In [87]:

```
new_dataset['Class'].value_counts()
```

Out[87]:

```
0.0    492
1.0    142
Name: Class, dtype: int64
```

In [89]:

```
new_dataset.groupby('Class').mean()
```

Out[89]:

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	
0.0	27764.721545	-0.169966	0.025639	0.780178	0.261962	-0.182046	0.012629	-0.170985	-0.072877	0.250426	0.024435	0.4
1.0	26193.556338	-7.871880	5.609155	-10.671851	6.067972	-5.862266	-2.315720	-8.269674	3.901566	-3.650345	-7.653117	5.4

Splitting the data into Features & Targets

In [90]:

```
X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']
```

In [91]:

```
print(X)
```

```
        Time         V1        V2  ...       V27       V28  Amount
13309  23451  -0.808883  1.056344  ...  0.366779  0.206565   48.04
27857  34736   1.267014 -0.716659  ...  0.018891  0.010734   26.80
15996  27437  -1.156895 -0.537961  ... -0.021691 -0.034374    9.71
19357  30210  -0.436689  0.071842  ...  0.100384  0.083696    1.00
20076  30769  -0.732469  1.540226  ... -0.293289 -0.089762   31.32
...      ...        ...       ...  ...       ...       ...     ...
44091  41791  -7.222731  6.155773  ...  1.193695  0.257468   99.99
44223  41851 -19.139733  9.286847  ... -3.381843 -1.256524  139.90
44270  41870 -20.906908  9.843153  ... -3.765371 -1.071238    1.00
44556  41991  -4.566342  3.353451  ...  0.195985  0.141115    1.00
45203  42247  -2.524012  2.098152  ...  0.456023 -0.405682    1.00

[634 rows x 30 columns]
```

In [92]:

```
print(Y)
```

```
13309    0.0
27857    0.0
15996    0.0
19357    0.0
20076    0.0
        ...
44091    1.0
44223    1.0
44270    1.0
```

```
44556    1.0
45203    1.0
Name: Class, Length: 634, dtype: float64
```

Split the data into Training data & Testing Data

In [93]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, ran
dom_state=2)
```

In [94]:

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(634, 30) (507, 30) (127, 30)
```

Model Training

Logistic Regression

In [103]:

```
model = LogisticRegression()
```

In [ ]:

```
# training the Logistic Regression Model with Training Data
model.fit(X_train, Y_train)
```

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=100, multi_class='auto', n_jobs=None, penalty='l2', random_state=None, solver='lbfgs', tol=0.0001, verbose=0, warm_start=False)

Model Evaluation

Accuracy Score

In [102]:

```
# accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

In [111]:

```
print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.9861932938856016
```

In [37]:

```
# accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

In [109]:

```
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.984251968503937
```

In [ ]:

```

```