In [7]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [8]:
```
# loading the dataset with pandas
credit_card_data = pd.read_csv('D:\study\datasets\creditcard.csv')
```

In [9]:
```
# first five rows
credit_card_data.head()
```

Out[9]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458

5 rows × 31 columns

In [10]:
```
# last five rows
credit_card_data.tail()
```

Out[10]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078

5 rows × 31 columns

In [11]:
```
credit_card_data.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 [12]:
```
credit_card_data.isnull()
```

Out[12]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
284803	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
284804	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
284805	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False
284806	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False

284807 rows × 31 columns

In [13]:
```
# count of fraud transactions and correct ones
credit_card_data['Class'].value_counts()
```

Out[13]:
```
0    284315
1       492
Name: Class, dtype: int64
```

0 = Correct transactions

1 = Fraud transactions

In [14]:
```
# separating both the transactions
correct = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

In [15]:
```
print(correct.shape)
print(fraud.shape)
```

```
(284315, 31)
(492, 31)
```

In [16]:
```
correct.Amount.describe()
```

Out[16]:
```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

In [17]:
```
fraud.Amount.describe()
```

Out[17]:
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

In [18]:
```
# comparing the values for both transactions
credit_card_data.groupby('Class').mean()
```

Out[18]:

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	...	-0.000644	-0.001235
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588

2 rows × 30 columns

Building a sample dataset

Building a sample dataset

Number of fraud transactions = 492

In [20]:
```
correct_sample = correct.sample(n=492)
```

Concatinating two dataframes

In [24]:
```
new_dataset = pd.concat([correct_sample, fraud], axis=0)
```

In [25]:
```
new_dataset.head()
```

Out[25]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
106192	69866.0	-1.534438	1.500176	0.857480	1.400216	-0.378992	-0.303781	0.019725	0.689968	-0.238097	...	-0.016327	0.321613	(
204651	135379.0	1.859176	-2.211704	-1.295073	-1.853877	-0.624809	1.694614	-1.515967	0.508669	-1.116411	...	0.020730	0.214671	(
10291	16215.0	-0.252448	0.630190	1.455456	1.424390	-0.006792	2.246304	-1.744867	-2.257114	0.957178	...	-1.437323	0.212130	-(
269948	163849.0	-0.498010	0.752878	1.490247	0.894997	0.667065	-0.204548	0.778543	-0.341155	-1.103842	...	-0.230945	-0.367021	-(
45828	42513.0	1.158967	0.104368	0.576218	0.510226	-0.399121	-0.331660	-0.162663	0.107543	-0.124281	...	-0.172794	-0.538541	(

5 rows × 31 columns

In [26]:
```
new_dataset.tail()
```

Out[26]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	...	0.778584	-0.319189	0.(
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	...	0.370612	0.028234	-0.
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...	0.751826	0.834108	0.
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	...	0.583276	-0.269209	-0.(
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	...	-0.164350	-0.295135	-0.(

5 rows × 31 columns

In [27]:
```
new_dataset['Class'].value_counts()
```

Out[27]:
```
0    492
1    492
Name: Class, dtype: int64
```

In [32]:
```
new_dataset.groupby('Class').mean()
```

Out[32]:

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	
0	94238.609756	0.025103	0.014805	0.031649	-0.016647	-0.010814	0.116369	-0.021979	0.031426	0.022931	...	-0.019893	0.014618	(
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588	(

2 rows × 30 columns

Spilling the data

In [33]:
```
x = new_dataset.drop(columns='Class',axis=1)
y = new_dataset['Class']
```

In [34]:
```
print(x)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
106192   69866.0 -1.534438  1.500176  0.857480  1.400216 -0.378992 -0.303781
204651  135379.0  1.859176 -2.211704 -1.295073 -1.853877 -0.624809  1.694614
10291    16215.0 -0.252448  0.630190  1.455456  1.424390 -0.006792  2.246304
269948  163849.0 -0.498010  0.752878  1.490247  0.894997  0.667065 -0.204548
45828    42513.0  1.158967  0.104368  0.576218  0.510226 -0.399121 -0.331660
...          ...       ...       ...       ...       ...       ...       ...
279863  169142.0 -1.927883  1.125653 -4.518331  1.749293 -1.566487 -2.010494
280143  169347.0  1.378559  1.289381 -5.004247  1.411850  0.442581 -1.326536
280149  169351.0 -0.676143  1.126366 -2.213700  0.468308 -1.120541 -0.003346
281144  169966.0 -3.113832  0.585864 -5.399730  1.817092 -0.840618 -2.943548
281674  170348.0  1.991976  0.158476 -2.583441  0.408670  1.151147 -0.096695

               V7        V8        V9  ...       V20       V21       V22  \
106192  0.019725  0.689968 -0.238097  ...  0.031253 -0.016327  0.321613
204651 -1.515967  0.508669 -1.116411  ... -0.217601  0.020730  0.214671
10291  -1.744867 -2.257114  0.957178  ...  0.881607 -1.437323  0.212130
269948  0.778543 -0.341155 -1.103842  ...  0.429207 -0.230945 -0.367021
45828  -0.162663  0.107543 -0.124281  ... -0.165912 -0.172794 -0.538541
...          ...       ...       ...  ...       ...       ...       ...
279863 -0.882850  0.697211 -2.064945  ...  1.252967  0.778584 -0.319189
280143 -1.413170  0.248525 -1.127396  ...  0.226138  0.370612  0.028234
280149 -2.234739  1.210158 -0.652250  ...  0.247968  0.751826  0.834108
281144 -2.208002  1.058733 -1.632333  ...  0.306271  0.583276 -0.269209
281674  0.223050 -0.068384  0.577829  ... -0.017652 -0.164350 -0.295135

              V23       V24       V25       V26       V27       V28  Amount
106192  0.030343  0.424600 -0.050812 -0.271094  0.020517 -0.104559    3.92
204651  0.155931 -0.988112 -0.513024 -0.107899  0.029605 -0.040035  174.00
10291  -0.407351 -1.362397  0.905785 -0.126134  0.117920  0.217801   74.97
269948 -0.151650  0.086119  0.101803  0.710173 -0.253347 -0.138629    0.99
45828   0.187180  0.199368  0.063062  0.096554 -0.020827  0.005999    1.98
...          ...       ...       ...       ...       ...       ...     ...
279863  0.639419 -0.294885  0.537503  0.788395  0.292680  0.147968  390.00
280143 -0.145640 -0.081049  0.521875  0.739467  0.389152  0.186637    0.76
280149  0.190944  0.032070 -0.739695  0.471111  0.385107  0.194361   77.89
281144 -0.456108 -0.183659 -0.328168  0.606116  0.884876 -0.253700  245.00
281674 -0.072173 -0.450261  0.313267 -0.289617  0.002988 -0.015309   42.53

[984 rows x 30 columns]
```

In [35]:
```
print(y)
```

```
106192    0
204651    0
10291     0
269948    0
45828     0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

Spilling data into training and testing data

In [39]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, stratify=y, random_state=2)
```

In [40]:
```
print(x.shape, x_train.shape, x_test.shape)
```

```
(984, 30) (787, 30) (197, 30)
```

Model training

Logistic regression

In [43]:
```
model = LogisticRegression()
```

In [45]:
```
# training logistic regression with training data
model.fit(x_train, y_train)
```

Out[45]:
```
▾ LogisticRegression
LogisticRegression()
```

Accuracy score

In [46]:
```
x_train_prediction = model.predict(x_train)
training_data_prediction = accuracy_score(x_train_prediction, y_train)
```

In [50]:
```
print('Accuracy on training data = ', training_data_prediction)
```

```
Accuracy on training data =  0.9479034307496823
```

In [54]:
```
x_test_prediction = model.predict(x_test)
test_data_prediction = accuracy_score(x_test_prediction, y_test)
```

In [55]:
```
print('Accuracy on test data = ', test_data_prediction)
```

```
Accuracy on test data =  0.9289340101522843
```

In [ ]: