```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
# loading the dataset to Pandas DataFrame
cdd=pd.DataFrame()
cdd=pd.read_csv("/content/crd_data.csv")
cdd
```

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
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	1.014480
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	0.012463
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	-0.037501
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	-0.163298
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	0.376777

284807 rows × 31 columns

```
# First 5 rows of the dataset
cdd.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798

5 rows × 31 columns

```
# Last 5 rows of the dataset
cdd.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.2134
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.2142
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.2320
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.2652
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.2610

5 rows × 31 columns

```
# information about the dataset
cdd.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
```

```
# Checking the missing values in each column
cdd.isna().sum()
```

```
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
```

```
# Distribution of legit transaction and fradulant transaction
cdd['Class'].value_counts()
```

```
0    284315
1       492
Name: Class, dtype: int64
```

The dataset is highly unbalanced.

0 :- Normal transaction

1 :- Fradulant transaction

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

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

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

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

```
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
```

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

```
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
```

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

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
Class												
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	...	-0.(
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.3

2 rows × 30 columns

Under Sampling

Build a sample dataset containing similar distribution of Normal Transaction and Fradulant Transaction

Number of Fradulant Transaction :- 492

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

```
# Concatinating two dataframes

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

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V
208301	137056.0	2.142093	0.648832	-3.614811	0.347483	1.857524	-0.707254	0.725239	-0.291990	-0.217878	...	-0.084
162107	114828.0	-0.104303	1.107666	-0.352884	-0.474848	0.473886	-1.151622	0.867878	-0.033408	0.371437	...	-0.362
274298	165946.0	-0.479484	1.323058	0.163166	0.391006	1.669099	-0.315946	1.700516	-0.490564	-1.289008	...	0.214
238296	149607.0	1.995117	-0.201209	-1.085143	0.017048	-0.027372	-0.027326	-0.613795	0.184730	1.097077	...	-0.251
79897	58228.0	1.364625	-0.849058	0.360596	-0.524372	-1.196476	-0.523307	-0.778607	0.030202	0.015735	...	-0.131

5 rows × 31 columns

```
new_dataset.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V2
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	...	0.7785
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	...	0.3706
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...	0.7518
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	...	0.5832
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	...	-0.1643

5 rows × 31 columns

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

```
0    492
1    492
Name: Class, dtype: int64
```

Now, the data is uniformly distributed.

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

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
0	94003.871951	-0.048218	-0.041500	0.041474	-0.048248	0.013213	0.055639	0.051425	-0.002005	0.071083	...	0.0
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.3

2 rows × 30 columns

```
# Splitting the data into features and targets

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

```
print(x)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
208301  137056.0  2.142093  0.648832 -3.614811  0.347483  1.857524 -0.707254
162107  114828.0 -0.104303  1.107666 -0.352884 -0.474848  0.473886 -1.151622
274298  165946.0 -0.479484  1.323058  0.163166  0.391006  1.669099 -0.315946
238296  149607.0  1.995117 -0.201209 -1.085143  0.017048 -0.027372 -0.027326
79897    58228.0  1.364625 -0.849058  0.360596 -0.524372 -1.196476 -0.523307
...          ...       ...       ...       ...       ...       ...       ...
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  \
208301  0.725239 -0.291990 -0.217878  ... -0.057340 -0.084325 -0.048553
162107  0.867878 -0.033408  0.371437  ...  0.018130 -0.362419 -0.913134
274298  1.700516 -0.490564 -1.289008  ...  0.002507  0.214844  0.750064
238296 -0.613795  0.184730  1.097077  ... -0.150184 -0.251018 -0.645947
79897  -0.778607  0.030202  0.015735  ... -0.033862 -0.131643 -0.476452
...          ...       ...       ...  ...       ...       ...       ...
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
```

```
208301 -0.191678 -0.509830  0.526800  0.726535 -0.084407 -0.033064    0.76
162107  0.180855  0.992046 -0.489897  0.094163  0.319381  0.143981    4.49
274298 -0.414222  0.728890  0.475903 -0.437212 -0.320117 -0.112640    3.50
238296  0.344530  0.136674 -0.464391 -0.307700  0.012947 -0.017958    3.69
79897  -0.026151 -0.166936  0.416095 -0.296530 -0.000416  0.011999   37.00
...          ...       ...       ...       ...       ...       ...     ...
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]
```

```
print(y)
```

```
208301    0
162107    0
274298    0
238296    0
79897     0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

```
# Splitting the data into training data and testing data

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=2, stratify=y)
```

```
print(x.shape, x_train.shape, x_test.shape)
```

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

Model training

Logistic Regression

```
model = LogisticRegression()
# Training the Logistic Regression model with Training Data
 model.fit(x_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

model evaluation on the basis of the accuracy score

```
# Accuracy on training data

x_train_prediction = model.predict(x_train)
training_data_accuracy = accuracy_score(x_train_prediction, y_train)


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

```
Accuracy on training data : 0.9148665819567979
```

```
# Accuracy on test data

x_test_prediction = model.predict(x_test)
test_data_accuracy = accuracy_score(x_test_prediction, y_test)


print('Accuracy on test data :', test_data_accuracy)
```

```
Accuracy on test data : 0.9137055837563451
```