```
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


credit_card_data = pd.read_csv("/content/creditcard.csv")
credit_card_data.head()
```

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

5 rows × 31 columns

```
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
```

```
credit_card_data["Class"].value_counts()
# 0 is the normal data
# 1 is the fraud data
```

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

```
normal = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

```
normal.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
```

```
normal.shape
```

```
(284315, 31)
```

```
fraud.shape
```

```
(492, 31)
```

```
credit_card_data.groupby("Class").mean()
```

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	(
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2

2 rows × 30 columns

Under Sampling

Building a distribution which will contain same number of data from normal and fraud data i.e. 282

```
normal_sample = normal.sample(n = 492)
```

Concatenating 2 data frames

```
df = pd.concat([normal_sample,fraud],axis = 0)
```

```
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8

```
df["Class"].value_counts()
```

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

177767	123335.0	2.107141	-0.095715	-2.667412	-0.345582	1.118200	-0.633853	0.698127	-0.354241	0.08

```
df.groupby("Class").mean()
```

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	93456.400407	0.119693	-0.061463	0.009113	-0.004934	0.029485	0.028105	0.018522	0.070291	0
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2

2 rows × 30 columns

Splitting the data into features and targets

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

```
x.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
81876	59139.0	-0.906358	1.040641	0.876514	1.057104	-0.564895	-0.975746	0.646942	0.294364	-0.67
270195	163957.0	-8.524914	7.499069	-7.047385	-1.970566	-3.238241	-2.159327	-2.667546	4.665639	1.78
96968	66024.0	-0.837262	1.386247	1.271542	2.915289	0.271627	0.083552	0.093877	0.452096	-1.49
177767	123335.0	2.107141	-0.095715	-2.667412	-0.345582	1.118200	-0.633853	0.698127	-0.354241	0.08
170514	120214.0	0.717559	-3.280086	-2.895946	-1.120625	-0.724015	-0.330075	0.881767	-0.484274	-0.64

5 rows × 30 columns

```
y.head()
```

```
81876     0
270195    0
96968     0
177767    0
170514    0
Name: Class, dtype: int64
```

Split 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 , stratify = y , random_state = 2)
```

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

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

Model Training

```
model = LogisticRegression()
```

Feeding the training data to the model

```
model.fit(x_train,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

Model Performance

```
x_train_predict = model.predict(x_train)
```

```
#accuracy for training data

training_data_accuracy = accuracy_score(x_train_predict,y_train)
training_data_accuracy.round(3)
```

```
0.925
```

```
x_test_predict = model.predict(x_test)
```

```
#accuracy for test data

test_data_accuracy = accuracy_score(x_test_predict,y_test)
test_data_accuracy.round(3)
```

```
0.924
```