

✓ Credit Card Fraud Detection - Logistic Regression

Importing Libraries

```
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
```

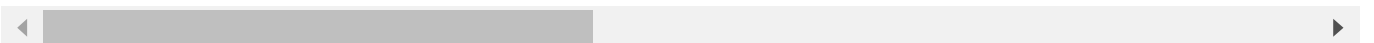
```
# loading the dataset to a Pandas DataFrame
credit_card_data = pd.read_csv('creditcard.csv')
```

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



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

5 rows × 31 columns

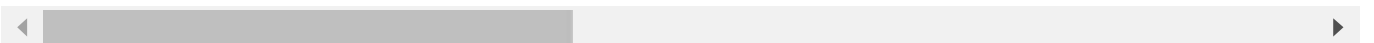


```
credit_card_data.tail()
```



	Time	V1	V2	V3	V4	V5	V6	V7
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006

5 rows × 31 columns



```
# dataset informations
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
```

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



	0
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 transactions & fraudulent transactions
credit_card_data['Class'].value_counts()
```



	count
Class	
0	284315
1	492

dtype: int64

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

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



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

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



	Amount
count	284315.000000
mean	88.291022
std	250.105092
min	0.000000
25%	5.650000
50%	22.000000
75%	77.050000
max	25691.160000

dtype: float64

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



	Amount
count	492.000000
mean	122.211321
std	256.683288
min	0.000000
25%	1.000000
50%	9.250000
75%	105.890000
max	2125.870000

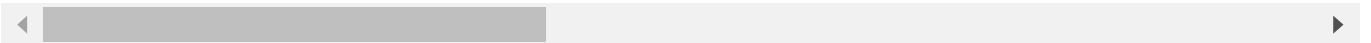
dtype: float64

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



	Time	V1	V2	V3	V4	V5	V6	
Class								
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.0096
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.5687

2 rows × 30 columns



Under-Sampling Build a sample dataset containing similar distribution of normal transactions and Fraudulent Transactions Number of Fraudulent Transactions --> 492

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

Concatenating two DataFrames

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

```
new_dataset.head()
```



	Time	V1	V2	V3	V4	V5	V6	V7
222644	143064.0	-0.204156	0.789728	0.676419	-0.702966	0.646070	-1.062063	1.261606
170948	120421.0	-0.907712	-0.236815	0.907539	-2.955275	-0.168306	1.180725	-0.707736
22404	32227.0	-1.113087	1.379736	1.213365	1.055965	0.181941	0.334371	0.534518
20513	31093.0	-1.192693	0.567001	1.018639	-2.027937	0.195278	-0.084362	0.468799
63843	50889.0	-0.653976	1.318063	1.506234	0.211357	0.025224	-0.798924	0.946490

5 rows × 31 columns



new_dataset.tail()



	Time	V1	V2	V3	V4	V5	V6	V7
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050

5 rows × 31 columns



new_dataset['Class'].value_counts()



	count
Class	
0	492
1	492

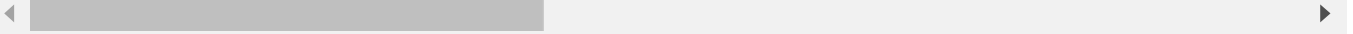
dtype: int64

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



	Time	V1	V2	V3	V4	V5	V6	V
Class								
0	93213.233740	0.129194	-0.141559	0.093879	0.010826	-0.007418	-0.054803	-0.07346
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.56873

2 rows × 30 columns



Splitting the data into Features & Targets

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

```
print(X)
```



	Time	V1	V2	V3	V4	V5	V6	\
222644	143064.0	-0.204156	0.789728	0.676419	-0.702966	0.646070	-1.062063	
170948	120421.0	-0.907712	-0.236815	0.907539	-2.955275	-0.168306	1.180725	
22404	32227.0	-1.113087	1.379736	1.213365	1.055965	0.181941	0.334371	
20513	31093.0	-1.192693	0.567001	1.018639	-2.027937	0.195278	-0.084362	
63843	50889.0	-0.653976	1.318063	1.506234	0.211357	0.025224	-0.798924	
...	
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 \
222644	1.261606	-0.382036	0.066857	...	0.037529	-0.287439	-0.528936	
170948	-0.707736	0.888734	-2.719511	...	-0.516192	-0.014660	0.275362	
22404	0.534518	0.244474	-0.078077	...	0.408270	-0.116449	0.404205	
20513	0.468799	0.076647	1.302181	...	-0.189550	0.118729	0.648605	
63843	0.946490	-0.347420	-0.803105	...	0.080555	-0.405211	-1.338554	
...	
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
222644	-0.012390	-0.044981	-0.644623	0.110487	0.182384	0.013292	6.79	
170948	-0.307543	-1.442482	0.431331	0.066443	-0.043824	-0.049160	10.00	
22404	-0.164734	0.031346	-0.033780	-0.284219	0.208148	-0.206163	9.00	
20513	-0.206028	-0.266573	0.160577	-0.770116	-0.093414	0.193993	5.50	
63843	0.025747	0.386570	-0.020912	0.071301	-0.453309	-0.017510	18.00	
...	
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)
```

```

222644    0
170948    0
22404     0
20513     0
63843     0
..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64

```

Split the data into Training data & Testing Data

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_
```

```
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

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 : 0.9250317662007624

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

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

➞ Accuracy score on Test Data : 0.9187817258883249

Start coding or [generate](#) with AI.