Credit Card Fraud Detection

> About This Project

In this project, we aimed to detect fraudulent credit card transactions using a logistic regression model. We began by importing necessary libraries, loading the dataset, and performing initial data exploration. The dataset contained highly imbalanced classes, with a majority of normal transactions (Class 0) and a minority of fraudulent transactions (Class 1).

In future iterations of this project, you could consider the following enhancements

1. **Data Loading and Exploration:**
    - The code begins by importing necessary libraries, including `numpy`, `pandas`, and specific modules from `sklearn`.
    - The credit card dataset is loaded using `pd.read_csv('/content/creditcard.csv')`.
    - The dataset's structure and basic information are displayed using `.head()`, `.tail()`, and `.info()` methods.
    - Missing values are checked using `.isnull().sum()` and are found to be absent in the dataset.
    - The distribution of the target variable 'Class' is shown using `.value_counts()`.
2. **Data Preparation and Separation:**
    - The dataset is divided into two subsets: legitimate transactions (`legit`) and fraudulent transactions (`fraud`) based on the 'Class' column.
    - Descriptive statistics of the 'Amount' feature are displayed for both legitimate and fraudulent transactions.
3. **Feature Engineering and Exploration:**
    - Summary statistics are shown for the features grouped by the 'Class' variable.
4. **Under-Sampling and Balancing:**
    - Under-sampling is performed to balance the dataset by randomly sampling an equal number of legitimate and fraudulent transactions.
    - The concatenated dataset of balanced samples is stored in `new_dataset`.
5. **Feature and Target Separation:**
    - The features (`x`) are extracted by dropping the 'Class' column from the `new_dataset`.
    - The target variable (`y`) is extracted, containing the 'Class' column from `new_dataset`.
6. **Data Splitting:**
    - The dataset is split into training and test sets using the `train_test_split` function.

- The dataset is split into training and test sets using the `train_test_split` function.
- The training set is assigned to `x_train` and `y_train`, and the test set is assigned to `x_test` and `y_test`.

7. **Model Training and Evaluation:**
   - A logistic regression model is created using `LogisticRegression()`.
   - The model is trained using the training data with `model.fit(x_train, y_train)`.
   - The accuracy of the model is evaluated on both the training and test sets using `accuracy_score`.
8. **Results:**
   - The accuracy of the model on the training data is approximately 92.4%.
   - The accuracy of the model on the test data is approximately 91.4%.

In conclusion, the provided code focuses on building a simple logistic regression model for credit card fraud detection. The model is trained and evaluated using accuracy as the evaluation metric. While accuracy provides an overview of the model's performance, it is important to consider other evaluation metrics, especially for imbalanced datasets like this one. Further improvements could involve exploring alternative models, utilizing more sophisticated evaluation metrics, and considering other techniques to handle class imbalance.

Importing Necessary 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 Dataset

```
Credit_Card_data = pd.read_csv('/content/creditcard.csv')
```

```
# first 5 rows
Credit_Card_data.head()
```

1 to 5 of 5 entries Filter

index	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807134	-0.072781173	2.536346738	1.378155224	-0.33832077	0.462387778	0.239598554	0.09869
1	0.0	1.191857111	0.266150712	0.166480113	0.448154078	0.060017649	-0.082360809	-0.078802983	0.08510
2	1.0	-1.358354062	-1.340163075	1.773209343	0.379779593	-0.503198133	1.800499381	0.791460956	0.24767
3	1.0	-0.966271712	-0.185226008	1.79299334	-0.863291275	-0.01030888	1.247203168	0.23760894	0.37743
4	2.0	-1.158233093	0.877736755	1.548717847	0.403033934	-0.407193377	0.095921462	0.592940745	-0.27053

```
# Last rows
Credit_Card_data.tail()
```

1 to 5 of 5 entries Filter

index	Time	V1	V2	V3	V4	V5	V6	V7	
284802	172786.0	-11.88111789	10.07178497	-9.834783457	-2.066655685	-5.364472781	-2.606837331	-4.918215431	7.
284803	172787.0	-0.732788671	-0.05508049	2.035029745	-0.738588584	0.868229399	1.058415272	0.024329696	0.
284804	172788.0	1.91956501	-0.301253846	-3.249639814	-0.557828125	2.63051512	3.031260098	-0.296826527	0.
284805	172788.0	-0.24044005	0.530482513	0.70251023	0.689799168	-0.377961134	0.623707722	-0.686179986	(
284806	172792.0	-0.596412522	-0.189733337	0.703337367	-0.50627124	-0.012545679	-0.649616686	1.577006254	-0.

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

```
Warning: Total number of columns (31) exceeds max_columns (20) limiting to first (20) columns.
```

```
# info
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
```

```
# Check missing values

Credit_Card_data.isnull().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
```

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

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

This Dataset is Highly Unblanced

0 = Noramal Transaction

1 = Fraudulent Transaction

```
# 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)
```

```
# Statical 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
```

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

1 to 2 of 2 entries Filter

Class	Time	V1	V2	V3	V4
0	94838.20225805884	0.008257515901953819	-0.0062708574165133745	0.012170917030775724	-0.007859867819657
1	80746.80691056911	-4.771948441388211	3.6237781019126016	-7.033281048536585	4.542029104436

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

```
Warning: Total number of columns (30) exceeds max_columns (20) limiting to first (20) columns.
```

Under-Sampling

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

```
# Concatenating two Data
```

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

```
new_dataset.head()
```

1 to 5 of 5 entries Filter

index	Time	V1	V2	V3	V4	V5	V6	V7	
249671	154524.0	0.138621231	0.504223165	0.226955681	-0.628245845	0.807858252	-1.095610497	1.290402334	-0.
193201	130035.0	2.135805959	-1.01803093	-0.757191627	-0.766406588	-1.094454941	-0.920283835	-0.811455336	-(
131528	79634.0	1.211782313	0.041224299	0.329701555	1.144392329	-0.20881684	-0.072755671	-0.087446824	
97391	66199.0	-1.077886975	1.098046354	1.18535212	-0.388477331	0.871209159	0.052171301	0.929343359	-0.
118899	75249.0	1.081609486	-1.797639018	0.569293681	-1.134711787	-2.099889495	-0.743406984	-0.998303228	-0

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

```
Warning: Total number of columns (31) exceeds max_columns (20) limiting to first (20) columns.
```

```
new_dataset.tail()
```

1 to 5 of 5 entries Filter

index	Time	V1	V2	V3	V4	V5	V6	V7	
279863	169142.0	-1.927883321	1.125652664	-4.518330641	1.749292533	-1.566487292	-2.010494231	-0.882849831	0.6
280143	169347.0	1.378558997	1.289380937	-5.004246784	1.411849842	0.442580636	-1.326535934	-1.413169956	0.2
280149	169351.0	-0.676142671	1.126366062	-2.213699523	0.468308388	-1.120541044	-0.003346296	-2.234739296	1.2
281144	169966.0	-3.113831607	0.585864172	-5.399730211	1.817092473	-0.840618466	-2.943547791	-2.20800192	1.0
281674	170348.0	1.991976096	0.158475887	-2.583440645	0.408669993	1.151147061	-0.096694744	0.223050267	-0.0

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

```
Warning: Total number of columns (31) exceeds max_columns (20) limiting to first (20) columns.
```

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

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

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

1 to 2 of 2 entries Filter

Class	Time	V1	V2	V3	V4
0	94868.6361788618	0.044826651589430895	-0.06486907317479675	-0.04177646449390244	-0.11030721061991869
1	80746.80691056911	-4.771948441388211	3.6237781019126016	-7.033281048536585	4.542029104436992

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

```
Warning: Total number of columns (30) exceeds max_columns (20) limiting to first (20) columns.
```

Splitting the data into features & Target

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

```
print(x)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
249671  154524.0  0.138621  0.504223  0.226956 -0.628246  0.807858 -1.095610
193201  130035.0  2.135806 -1.018031 -0.757192 -0.766407 -1.094455 -0.920284
```

```
131528   79634.0  1.211782  0.041224  0.329702  1.144392 -0.208817 -0.072756
97391    66199.0 -1.077887  1.098046  1.185352 -0.388477  0.871209  0.052171
118899   75249.0  1.081609 -1.797639  0.569294 -1.134712 -2.099889 -0.743407
...          ...       ...       ...       ...       ...       ...       ...
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  \
249671  1.290402 -0.503858  0.022366  ... -0.046210 -0.145491 -0.215729
193201 -0.811455 -0.148290  0.122525  ... -0.074043  0.202852  0.567046
131528 -0.087447  0.010871  0.655480  ... -0.127124 -0.319999 -0.820429
97391   0.929343 -0.004249 -0.564004  ...  0.079059 -0.514485 -1.465811
118899 -0.998303 -0.171101 -1.548716  ...  0.089188  0.064507  0.088093
...          ...       ...       ...  ...       ...       ...       ...
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
249671  0.235163 -0.110294 -1.435347 -0.055238  0.037253  0.045597   26.99
193201  0.168626 -0.088614 -0.190652 -0.170339 -0.011791 -0.051808   32.00
131528 -0.072292 -0.482841  0.577632 -0.493104  0.030474  0.021828   28.27
97391   0.038850 -0.982764  0.157070 -0.006196 -0.002640  0.129799   14.65
118899 -0.185738  0.369997  0.245524 -0.124256  0.008085  0.063739  228.00
...          ...       ...       ...       ...       ...       ...     ...
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)
```

```
249671    0
193201    0
131528    0
97391     0
118899    0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

Split the data into Training data & Test

```
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

Logistic Regression

```
model = LogisticRegression()
```

```
# Train 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:  0.9237611181702668
```

```
# 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
```

0s completed at 1:29 PM