Credit Card Fraud Detection.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

Importing the Dependencies

```
[1] 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
```

```
[2] # loading the dataset to a Pandas DataFrame
    credit_card_data = pd.read_csv('/content/creditcard.csv')
```

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

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
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	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0.0
1	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	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0.0
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0.0
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0.0
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0.0

5 rows × 31 columns

```
[4] credit_card_data.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
97137	66085	0.998571	-0.046655	0.644378	1.114925	-0.665698	-0.627024	-0.002057	0.018110	-0.052156	...	-0.171808	-0.784227	0.095822	0.455048	0.167887	-0.729060	0.004417	0.037612	89.0	0.0
97138	66085	-1.326193	0.549467	1.220272	1.286509	0.473532	-0.681876	-0.249255	0.444731	-0.768583	...	0.088777	0.029885	-0.123943	-0.092548	-0.159851	-0.360097	0.318036	0.007246	3.6	0.0
97139	66086	1.230983	-0.224520	-0.345196	0.212802	1.586953	3.997378	-1.145351	1.068038	0.584379	...	0.067612	0.229977	-0.119921	1.019614	0.667317	-0.226637	0.071064	0.028365	1.0	0.0
97140	66086	1.241193	0.767604	-0.210715	1.297487	0.152102	-1.162435	0.389686	-0.321743	-0.288129	...	-0.036601	0.032307	-0.136263	0.308814	0.738340	-0.331821	0.040823	0.054137	1.0	0.0
97141	66087	0.310485	-2.576074	1.002015	0.011196	-2.280745	0.465648	-0.860224	0.156411	0.087629	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 31 columns

```
[5] # dataset informations
    credit_card_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97142 entries, 0 to 97141
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Time    97142 non-null  int64
 1   V1      97142 non-null  float64
 2   V2      97142 non-null  float64
 3   V3      97142 non-null  float64
 4   V4      97142 non-null  float64
 5   V5      97142 non-null  float64
 6   V6      97142 non-null  float64
 7   V7      97142 non-null  float64
 8   V8      97142 non-null  float64
 9   V9      97142 non-null  float64
 10  V10     97142 non-null  float64
 11  V11     97142 non-null  float64
 12  V12     97142 non-null  float64
 13  V13     97142 non-null  float64
 14  V14     97142 non-null  float64
 15  V15     97142 non-null  float64
 16  V16     97142 non-null  float64
 17  V17     97142 non-null  float64
 18  V18     97142 non-null  float64
 19  V19     97141 non-null  float64
 20  V20     97141 non-null  float64
 21  V21     97141 non-null  float64
 22  V22     97141 non-null  float64
 23  V23     97141 non-null  float64
 24  V24     97141 non-null  float64
 25  V25     97141 non-null  float64
 26  V26     97141 non-null  float64
 27  V27     97141 non-null  float64
 28  V28     97141 non-null  float64
 29  Amount  97141 non-null  float64
 30  Class   97141 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 23.0 MB
```

```
[6] # checking the number of missing values in each column
    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       1
V20       1
V21       1
V22       1
V23       1
V24       1
V25       1
V26       1
V27       1
V28       1
Amount    1
Class     1
dtype: int64
```

```
[7] # distribution of legit transactions & fraudulent transactions
    credit_card_data['Class'].value_counts()
```

```
Class
0.0    96919
1.0      222
Name: count, dtype: int64
```

This Dataset is highly unblanced

0 --> Normal Transaction

1 --> fraudulent transaction

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

```
[9] print(legit.shape)
    print(fraud.shape)
```

```
(96919, 31)
(222, 31)
```

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

```
count    96919.000000
mean        98.310270
std        265.983851
min          0.000000
25%          7.580000
50%         26.610000
75%         89.345000
max      19656.530000
Name: Amount, dtype: float64
```

```
[11] fraud.Amount.describe()
```

```
count     222.000000
mean      114.488243
std       255.373074
min         0.000000
25%         1.000000
50%         7.805000
75%        99.990000
max      1809.680000
Name: Amount, dtype: float64
```

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

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount
0.0	41730.096658	-0.249732	-0.043534	0.694778	0.151455	-0.269573	0.098435	-0.093862	0.050744	-0.036243	...	0.043675	-0.031999	-0.108137	-0.036618	0.009741	0.131925	0.026549	-0.000700	0.001378	98.310270
1.0	36541.941441	-6.044462	4.134072	-7.932926	4.915738	-4.386432	-1.796113	-6.300490	2.722455	-2.896811	...	0.345305	0.715188	-0.125165	-0.265450	-0.105791	0.205945	0.103589	0.523395	0.037908	114.488243

2 rows × 30 columns

```
[13] legit_sample = legit.sample(n=492)
```

Concatenating two DataFrames

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

```
[15] new_dataset.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
33968	37457	1.177690	-0.103608	0.392021	0.591672	-0.233191	0.120053	-0.180798	0.066232	0.573304	...	-0.296459	-0.701009	0.024720	-0.427385	0.305461	0.346533	-0.011820	0.010740	29.69	0.0
30875	36114	0.891118	-2.063481	0.503965	-1.256899	-1.902577	0.071625	-1.102155	0.096096	-1.830302	...	0.177594	0.293194	-0.216890	0.012810	0.125049	-0.118184	0.011053	0.059742	282.00	0.0
49041	43888	-0.120126	-2.382475	1.242680	2.394520	-2.241805	0.499088	-0.278874	0.203564	1.784094	...	0.312191	-0.042700	-0.583453	0.753939	0.249634	-0.306432	-0.013948	0.146500	623.02	0.0
13529	24000	1.207892	-0.098400	0.472470	0.041344	-0.659200	-0.890118	-0.245740	-0.160615	1.667604	...	-0.100598	-0.080898	0.048033	0.421557	0.112275	1.423444	-0.135263	-0.010233	24.99	0.0
9853	14522	-3.685646	-4.034000	2.340454	0.971496	3.988201	-2.461228	-3.146273	0.669894	1.242409	...	0.399882	-0.010333	0.591191	-0.343474	0.128209	1.204143	-0.190769	0.128554	15.95	0.0

5 rows × 31 columns

```
[16] new_dataset.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
95534	65358	1.193916	-0.571085	0.742522	-0.014588	-0.624561	0.832162	-0.833350	0.272897	1.169425	...	-0.049502	0.207265	-0.265272	-0.679294	0.511812	1.246604	-0.028671	-0.006112	31.91	1.0
95597	65385	-2.923827	1.524837	-3.018758	3.289291	-5.755542	2.218276	-0.509995	-3.569444	-1.016592	...	-0.511657	-0.122724	-4.288639	0.563797	-0.949451	-0.204532	1.510206	-0.324706	1354.25	1.0
96341	65728	1.227614	-0.668974	-0.271785	-0.589440	-0.604795	-0.350285	-0.486365	-0.010809	-0.794944	...	-0.026055	-0.295255	-0.180459	-0.436539	0.494649	-0.283738	-0.001128	0.035075	98.01	1.0
96789	65936	-3.593476	0.781442	-1.822448	0.605761	-1.194656	-0.517195	-1.722523	0.128890	0.014963	...	0.351792	0.391249	-0.252875	-0.498042	0.010172	0.909929	-1.478767	0.722673	101.50	1.0
96994	66037	0.286302	1.399345	-1.682503	3.864377	-1.185373	-0.341732	-2.539380	0.768378	-1.547882	...	0.352456	-0.243678	-0.194079	-0.172201	0.742237	0.127790	0.569731	0.291206	7.53	1.0

5 rows × 31 columns

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

```
Class
0.0    492
1.0    222
Name: count, dtype: int64
```

```
[18] new_dataset.groupby('Class').mean()
```

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount
0.0	41432.353659	-0.379484	-0.175702	0.671829	0.172682	-0.272168	0.090532	-0.10443	0.091357	0.002649	...	0.074892	-0.032420	-0.116489	0.004871	0.020850	0.153029	0.049331	0.002452	0.011798	114.814715
1.0	36541.941441	-6.044462	4.134072	-7.932926	4.915738	-4.386432	-1.796113	-6.30049	2.722455	-2.896811	...	0.345305	0.715188	-0.125165	-0.265450	-0.105791	0.205945	0.103589	0.523395	0.037908	114.488243

2 rows × 30 columns

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

```
[20] print(X)
```

```
        Time        V1        V2        V3        V4        V5        V6  \
33968  37457  1.177690 -0.103608  0.392021  0.591672 -0.233191  0.120053
30875  36114  0.891118 -2.063481  0.503965 -1.256899 -1.902577  0.071625
49041  43888 -0.120126 -2.382475  1.242680  2.394520 -2.241805  0.499088
13529  24000  1.207892 -0.098400  0.472470  0.041344 -0.659200 -0.890118
9853   14522 -3.685646 -4.034000  2.340454  0.971496  3.988201 -2.461228
...      ...       ...       ...       ...       ...       ...       ...
95534  65358  1.193916 -0.571085  0.742522 -0.014588 -0.624561  0.832162
95597  65385 -2.923827  1.524837 -3.018758  3.289291 -5.755542  2.218276
96341  65728  1.227614 -0.668974 -0.271785 -0.589440 -0.604795 -0.350285
96789  65936 -3.593476  0.781442 -1.822448  0.605761 -1.194656 -0.517195
96994  66037  0.286302  1.399345 -1.682503  3.864377 -1.185373 -0.341732

             V7        V8        V9  ...       V20       V21       V22  \
33968 -0.180798  0.066232  0.573304  ... -0.078592 -0.296459 -0.701009
30875 -1.102155  0.096096 -1.830302  ...  0.183339  0.177594  0.293194
49041 -0.278874  0.203564  1.784094  ...  0.997603  0.312191 -0.042700
13529 -0.245740 -0.160615  1.667604  ... -0.170898 -0.100598 -0.080898
9853  -3.146273  0.669894  1.242409  ...  1.175229  0.399882 -0.010333
...         ...       ...       ...  ...       ...       ...       ...
95534 -0.833350  0.272897  1.169425  ...  0.062908 -0.049502  0.207265
95597 -0.509995 -3.569444 -1.016592  ... -0.447039 -0.511657 -0.122724
96341 -0.486365 -0.010809 -0.794944  ...  0.273799 -0.026055 -0.295255
96789 -1.722523  0.128890  0.014963  ... -0.478219  0.351792  0.391249
96994 -2.539380  0.768378 -1.547882  ...  0.270360  0.352456 -0.243678

            V23       V24       V25       V26       V27       V28   Amount
33968  0.024720 -0.427385  0.305461  0.346533 -0.011820  0.010740    29.69
30875 -0.216890  0.012810  0.125049 -0.118184  0.011053  0.059742   282.00
49041 -0.583453  0.753939  0.249634 -0.306432 -0.013948  0.146500   623.02
13529  0.048033  0.421557  0.112275  1.423444 -0.135263 -0.010233    24.99
9853   0.591191 -0.343474  0.128209  1.204143 -0.190769  0.128554    15.95
...         ...       ...       ...       ...       ...       ...      ...
95534 -0.265272 -0.679294  0.511812  1.246604 -0.028671 -0.006112    31.91
95597 -4.288639  0.563797 -0.949451 -0.204532  1.510206 -0.324706  1354.25
96341 -0.180459 -0.436539  0.494649 -0.283738 -0.001128  0.035075    98.01
96789 -0.252875 -0.498042  0.010172  0.909929 -1.478767  0.722673   101.50
96994 -0.194079 -0.172201  0.742237  0.127790  0.569731  0.291206     7.53

[714 rows x 30 columns]
```

```
[21] print(Y)
```

```
33968    0.0
30875    0.0
49041    0.0
13529    0.0
9853     0.0
        ...
95534    1.0
95597    1.0
96341    1.0
96789    1.0
96994    1.0
Name: Class, Length: 714, dtype: float64
```

Split the data into Training data & Testing Data

```
[22] X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)
```

```
[23] print(X.shape, X_train.shape, X_test.shape)
```

```
(714, 30) (571, 30) (143, 30)
```

Model Training

Logistic Regression

```
[24] model = LogisticRegression()
```

```
[25] # training the Logistic Regression Model with Training Data
     model.fit(X_train, Y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
▾ LogisticRegression
LogisticRegression()
```

Model Evaluation

Accuracy Score

```
[26] # accuracy on training data
     X_train_prediction = model.predict(X_train)
     training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
[27] print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.957968476357268
```

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

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

```
Accuracy score on Test Data :  0.9230769230769231
```