credit-card-fraud

September 11, 2023

1 Importing the Dependecies

[2]:
```
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
```

1.1 Loading the dataset to a Pandas DataFrame

[3]:
```
credit_card_data = pd.read_csv('creditcard.csv')
```

1.2 first 5 rows of the dataset

[4]:
```
credit_card_data.head()
```

[4]:

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

	V8	V9	…	V21	V22	V23	V24	V25	\
0	0.098698	0.363787	…	-0.018307	0.277838	-0.110474	0.066928	0.128539	
1	0.085102	-0.255425	…	-0.225775	-0.638672	0.101288	-0.339846	0.167170	
2	0.247676	-1.514654	…	0.247998	0.771679	0.909412	-0.689281	-0.327642	
3	0.377436	-1.387024	…	-0.108300	0.005274	-0.190321	-1.175575	0.647376	
4	-0.270533	0.817739	…	-0.009431	0.798278	-0.137458	0.141267	-0.206010	

	V26	V27	V28	Amount	Class
0	-0.189115	0.133558	-0.021053	149.62	0
1	0.125895	-0.008983	0.014724	2.69	0
2	-0.139097	-0.055353	-0.059752	378.66	0
3	-0.221929	0.062723	0.061458	123.50	0
4	0.502292	0.219422	0.215153	69.99	0

```
[5 rows x 31 columns]
```

[5]:
```
credit_card_data.tail()
```

[5]:
```
            Time         V1         V2        V3        V4        V5  \
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

              V6        V7        V8        V9  …       V21       V22  \
284802 -2.606837 -4.918215  7.305334  1.914428  …  0.213454  0.111864
284803  1.058415  0.024330  0.294869  0.584800  …  0.214205  0.924384
284804  3.031260 -0.296827  0.708417  0.432454  …  0.232045  0.578229
284805  0.623708 -0.686180  0.679145  0.392087  …  0.265245  0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  …  0.261057  0.643078

             V23       V24       V25       V26       V27       V28  Amount  \
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

        Class
284802      0
284803      0
284804      0
284805      0
284806      0

[5 rows x 31 columns]
```

1.3 dataset informations

[6]:
```
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
```

1.4 checking the number of missing values in each column

[7]:
```
credit_card_data.isnull().sum()
```

[7]:
```
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
```

1.5 distribution of legit transactions & fraudulent transactions

[8]:
```
credit_card_data['Class'].value_counts()
```

[8]:
```
0    284315
1       492
Name: Class, dtype: int64
```

1.6 This Dataset is highly unblanced

1.6.1 0 –> Normal Transaction

1.6.2 1 –> fraudulent transaction

1.7 separating the data for analysis

[9]:
```
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

[10]:
```
print(legit.shape)
print(fraud.shape)
```

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

1.8 statistical measures of the data

[11]:
```
legit.Amount.describe()
```

[11]:
```
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
```

[12]:
```
fraud.Amount.describe()
```

[12]:
```
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
```

1.9 compare the values for both transactions

[13]:
```
credit_card_data.groupby('Class').mean()
```

[13]:
```
                Time        V1        V2        V3        V4        V5  \
Class
0      94838.202258  0.008258 -0.006271  0.012171 -0.007860  0.005453
1      80746.806911 -4.771948  3.623778 -7.033281  4.542029 -3.151225

               V6        V7        V8        V9  …       V20       V21  \
Class                                            …
0       0.002419  0.009637 -0.000987  0.004467  … -0.000644 -0.001235
1      -1.397737 -5.568731  0.570636 -2.581123  …  0.372319  0.713588

              V22       V23       V24       V25       V26       V27       V28  \
Class
0      -0.000024  0.000070  0.000182 -0.000072 -0.000089 -0.000295 -0.000131
1       0.014049 -0.040308 -0.105130  0.041449  0.051648  0.170575  0.075667

            Amount
Class
```

```
0       88.291022
1      122.211321

[2 rows x 30 columns]
```

1.10 Under-Sampling

1.10.1 Build a sample dataset containing similar distribution of normal transactions and Fraudulent Transactions

1.10.2 Number of Fraudulent Transactions –> 492

[14]:
```
legit_sample = legit.sample(n=492)
```

1.11 Concatenating two DataFrames

[15]:
```
new_dataset = pd.concat([legit_sample, fraud], axis=0)
```

[16]:
```
new_dataset.head()
```

[16]:
```
            Time        V1        V2        V3        V4        V5        V6  \
66362    52010.0 -0.765908  0.493080  1.618109  0.136591  0.446126 -1.007659
282100  170652.0 -0.698308  1.007052 -0.146513 -0.886018  0.714503 -0.638046
146629   87788.0  2.013142  0.051367 -1.956487  1.072391  0.857206 -0.167450
230388  146313.0 -0.898384  1.325619 -1.178800 -1.172976  3.025053  0.592774
207056  136498.0 -0.097266 -0.836552  1.009212 -2.747376 -0.732408 -1.028021

              V7        V8        V9  …       V21       V22       V23  \
66362   1.181387 -0.303829 -0.288825  …  0.106675  0.203843 -0.139368
282100  0.815012 -0.053728  0.199456  … -0.319738 -0.574539  0.126682
146629  0.404509 -0.082408  0.137153  …  0.056767  0.245756 -0.027406
230388  1.412550  0.326854 -1.099763  …  0.014995  0.056688 -0.690199
207056 -0.396264 -0.070058 -1.677403  … -0.111016 -0.035635 -0.051760

             V24       V25       V26       V27       V28  Amount  Class
66362   0.346814  0.271382 -0.498993 -0.118815 -0.108953   65.85      0
282100 -0.504955 -0.332802  0.155413  0.366311  0.185687    2.58      0
146629  0.193188  0.465498 -0.509647 -0.033106 -0.071018   15.99      0
230388 -1.147418  1.083427  0.838384 -0.039554  0.055762    0.76      0
207056 -0.086683 -0.055488 -0.266398  0.022441  0.020056   19.60      0

[5 rows x 31 columns]
```

[17]:
```
new_dataset.tail()
```

[17]:
```
            Time        V1        V2        V3        V4        V5        V6  \
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  …       V21       V22       V23  \
279863 -0.882850  0.697211 -2.064945  …  0.778584 -0.319189  0.639419
280143 -1.413170  0.248525 -1.127396  …  0.370612  0.028234 -0.145640
280149 -2.234739  1.210158 -0.652250  …  0.751826  0.834108  0.190944
281144 -2.208002  1.058733 -1.632333  …  0.583276 -0.269209 -0.456108
281674  0.223050 -0.068384  0.577829  … -0.164350 -0.295135 -0.072173

              V24       V25       V26       V27       V28  Amount  Class
279863 -0.294885  0.537503  0.788395  0.292680  0.147968  390.00      1
280143 -0.081049  0.521875  0.739467  0.389152  0.186637    0.76      1
280149  0.032070 -0.739695  0.471111  0.385107  0.194361   77.89      1
281144 -0.183659 -0.328168  0.606116  0.884876 -0.253700  245.00      1
281674 -0.450261  0.313267 -0.289617  0.002988 -0.015309   42.53      1

[5 rows x 31 columns]
```

[18]:
```
new_dataset['Class'].value_counts()
```

[18]:
```
0    492
1    492
Name: Class, dtype: int64
```

[19]:
```
new_dataset.groupby('Class').mean()
```

[19]:
```
               Time        V1        V2        V3        V4        V5  \
Class
0      95003.662602  0.145020 -0.025266 -0.007142  0.058877  0.129349
1      80746.806911 -4.771948  3.623778 -7.033281  4.542029 -3.151225

              V6        V7        V8        V9  …       V20       V21  \
Class                                           …
0     -0.040760 -0.011592  0.009452  0.030291  …  0.004375  0.034190
1     -1.397737 -5.568731  0.570636 -2.581123  …  0.372319  0.713588

             V22       V23       V24       V25       V26       V27       V28  \
Class
0     -0.047864 -0.008563  0.026288  0.024393  0.004171  0.004060  0.003908
1      0.014049 -0.040308 -0.105130  0.041449  0.051648  0.170575  0.075667

            Amount
Class
0       80.556870
```

```
1      122.211321

[2 rows x 30 columns]
```

1.12 *Splitting* the data into Features & Targets

[20]:
```
X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']
```

[21]:
```
print(X)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
66362    52010.0 -0.765908  0.493080  1.618109  0.136591  0.446126 -1.007659
282100  170652.0 -0.698308  1.007052 -0.146513 -0.886018  0.714503 -0.638046
146629   87788.0  2.013142  0.051367 -1.956487  1.072391  0.857206 -0.167450
230388  146313.0 -0.898384  1.325619 -1.178800 -1.172976  3.025053  0.592774
207056  136498.0 -0.097266 -0.836552  1.009212 -2.747376 -0.732408 -1.028021
…            …         …         …         …         …         …         …
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  \
66362    1.181387 -0.303829 -0.288825  …  0.048358  0.106675  0.203843
282100   0.815012 -0.053728  0.199456  …  0.128607 -0.319738 -0.574539
146629   0.404509 -0.082408  0.137153  … -0.305170  0.056767  0.245756
230388   1.412550  0.326854 -1.099763  … -0.141616  0.014995  0.056688
207056  -0.396264 -0.070058 -1.677403  … -0.433075 -0.111016 -0.035635
…             …         …         …  …       …         …         …
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
66362    -0.139368  0.346814  0.271382 -0.498993 -0.118815 -0.108953   65.85
282100    0.126682 -0.504955 -0.332802  0.155413  0.366311  0.185687    2.58
146629   -0.027406  0.193188  0.465498 -0.509647 -0.033106 -0.071018   15.99
230388   -0.690199 -1.147418  1.083427  0.838384 -0.039554  0.055762    0.76
207056   -0.051760 -0.086683 -0.055488 -0.266398  0.022441  0.020056   19.60
…              …         …         …         …         …         …       …
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]
```

[22]:
```
print(Y)
```

```
66362     0
282100    0
146629    0
230388    0
207056    0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

1.13 *Split* the data into Training data & Testing Data

[23]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
↪stratify=Y, random_state=2)
```

[24]:
```
print(X.shape, X_train.shape, X_test.shape)
```

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

2 Model Training

2.1 Logistic Regression

[25]:
```
model = LogisticRegression()
```

2.2 training the Logistic Regression Model with Training Data

[ ]:
```
model.fit(X_train, Y_train)
```

3 Model Evaluation

3.1 Accuracy Score

3.2 accuracy on training data

[28]:
```
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

[29]:
```
print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.9453621346886912
```

3.3 accuracy on test data

[30]:
```
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

[31]:
```
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.9289340101522843
```