Credit_Card_Fraud_Detection

January 30, 2025

[1]:
```python
import pandas as pd
import numpy as np
```

[2]:
```python
df = pd.read_csv("creditcard.csv")
df.head()
```

[2]:

```
   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]
```

[3]:
```python
pd.options.display.max_columns = None
```

[4]:
```python
df.tail()
```

[4]:

```
            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       V10       V11       V12  \
284802 -2.606837 -4.918215  7.305334  1.914428  4.356170 -1.593105  2.711941
284803  1.058415  0.024330  0.294869  0.584800 -0.975926 -0.150189  0.915802
284804  3.031260 -0.296827  0.708417  0.432454 -0.484782  0.411614  0.063119
284805  0.623708 -0.686180  0.679145  0.392087 -0.399126 -1.933849 -0.962886
284806 -0.649617  1.577006 -0.414650  0.486180 -0.915427 -1.040458 -0.031513

             V13       V14       V15       V16       V17       V18       V19  \
284802 -0.689256  4.626942 -0.924459  1.107641  1.991691  0.510632 -0.682920
284803  1.214756 -0.675143  1.164931 -0.711757 -0.025693 -1.221179 -1.545556
284804 -0.183699 -0.510602  1.329284  0.140716  0.313502  0.395652 -0.577252
284805 -1.042082  0.449624  1.962563 -0.608577  0.509928  1.113981  2.897849
284806 -0.188093 -0.084316  0.041333 -0.302620 -0.660377  0.167430 -0.256117

             V20       V21       V22       V23       V24       V25       V26  \
284802  1.475829  0.213454  0.111864  1.014480 -0.509348  1.436807  0.250034
284803  0.059616  0.214205  0.924384  0.012463 -1.016226 -0.606624 -0.395255
284804  0.001396  0.232045  0.578229 -0.037501  0.640134  0.265745 -0.087371
284805  0.127434  0.265245  0.800049 -0.163298  0.123205 -0.569159  0.546668
284806  0.382948  0.261057  0.643078  0.376777  0.008797 -0.473649 -0.818267

             V27       V28  Amount  Class
284802  0.943651  0.823731    0.77      0
284803  0.068472 -0.053527   24.79      0
284804  0.004455 -0.026561   67.88      0
284805  0.108821  0.104533   10.00      0
284806 -0.002415  0.013649  217.00      0
```

[5]:
```python
df.shape
```

[5]:
```
(284807, 31)
```

[6]:
```python
df.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
```

[7]:

```python
## make the dataset to a standarized format
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Assuming df is your DataFrame and it has a column 'Amount'
sc = StandardScaler()

# Correcting the use of DataFrame to standardize the 'Amount' column
df["Amount"] = sc.fit_transform(df[["Amount"]])

df.head()
```

[7]:

```
   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       V10       V11       V12       V13       V14  \
```

```
0  0.098698  0.363787  0.090794 -0.551600 -0.617801 -0.991390 -0.311169
1  0.085102 -0.255425 -0.166974  1.612727  1.065235  0.489095 -0.143772
2  0.247676 -1.514654  0.207643  0.624501  0.066084  0.717293 -0.165946
3  0.377436 -1.387024 -0.054952 -0.226487  0.178228  0.507757 -0.287924
4 -0.270533  0.817739  0.753074 -0.822843  0.538196  1.345852 -1.119670

        V15       V16       V17       V18       V19       V20       V21  \
0  1.468177 -0.470401  0.207971  0.025791  0.403993  0.251412 -0.018307
1  0.635558  0.463917 -0.114805 -0.183361 -0.145783 -0.069083 -0.225775
2  2.345865 -2.890083  1.109969 -0.121359 -2.261857  0.524980  0.247998
3 -0.631418 -1.059647 -0.684093  1.965775 -1.232622 -0.208038 -0.108300
4  0.175121 -0.451449 -0.237033 -0.038195  0.803487  0.408542 -0.009431

        V22       V23       V24       V25       V26       V27       V28  \
0  0.277838 -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053
1 -0.638672  0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724
2  0.771679  0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752
3  0.005274 -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458
4  0.798278 -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153

     Amount  Class
0  0.244964      0
1 -0.342475      0
2  1.160686      0
3  0.140534      0
4 -0.073403      0
```

[8]:
```python
df.drop(["Time"],axis = 1)
```

[8]:
```
               V1         V2        V3        V4        V5        V6  \
0       -1.359807  -0.072781  2.536347  1.378155 -0.338321  0.462388
1        1.191857   0.266151  0.166480  0.448154  0.060018 -0.082361
2       -1.358354  -1.340163  1.773209  0.379780 -0.503198  1.800499
3       -0.966272  -0.185226  1.792993 -0.863291 -0.010309  1.247203
4       -1.158233   0.877737  1.548718  0.403034 -0.407193  0.095921
…             …          …         …         …         …         …
284802 -11.881118  10.071785 -9.834783 -2.066656 -5.364473 -2.606837
284803  -0.732789  -0.055080  2.035030 -0.738589  0.868229  1.058415
284804   1.919565  -0.301254 -3.249640 -0.557828  2.630515  3.031260
284805  -0.240440   0.530483  0.702510  0.689799 -0.377961  0.623708
284806  -0.533413  -0.189733  0.703337 -0.506271 -0.012546 -0.649617

              V7        V8        V9       V10       V11       V12       V13  \
0       0.239599  0.098698  0.363787  0.090794 -0.551600 -0.617801 -0.991390
1      -0.078803  0.085102 -0.255425 -0.166974  1.612727  1.065235  0.489095
2       0.791461  0.247676 -1.514654  0.207643  0.624501  0.066084  0.717293
3       0.237609  0.377436 -1.387024 -0.054952 -0.226487  0.178228  0.507757
```

```
4       0.592941 -0.270533  0.817739  0.753074 -0.822843  0.538196  1.345852
…          …         …         …         …         …         …         …
284802 -4.918215  7.305334  1.914428  4.356170 -1.593105  2.711941 -0.689256
284803  0.024330  0.294869  0.584800 -0.975926 -0.150189  0.915802  1.214756
284804 -0.296827  0.708417  0.432454 -0.484782  0.411614  0.063119 -0.183699
284805 -0.686180  0.679145  0.392087 -0.399126 -1.933849 -0.962886 -1.042082
284806  1.577006 -0.414650  0.486180 -0.915427 -1.040458 -0.031513 -0.188093

             V14       V15       V16       V17       V18       V19       V20  \
0      -0.311169  1.468177 -0.470401  0.207971  0.025791  0.403993  0.251412
1      -0.143772  0.635558  0.463917 -0.114805 -0.183361 -0.145783 -0.069083
2      -0.165946  2.345865 -2.890083  1.109969 -0.121359 -2.261857  0.524980
3      -0.287924 -0.631418 -1.059647 -0.684093  1.965775 -1.232622 -0.208038
4      -1.119670  0.175121 -0.451449 -0.237033 -0.038195  0.803487  0.408542
…          …         …         …         …         …         …         …
284802  4.626942 -0.924459  1.107641  1.991691  0.510632 -0.682920  1.475829
284803 -0.675143  1.164931 -0.711757 -0.025693 -1.221179 -1.545556  0.059616
284804 -0.510602  1.329284  0.140716  0.313502  0.395652 -0.577252  0.001396
284805  0.449624  1.962563 -0.608577  0.509928  1.113981  2.897849  0.127434
284806 -0.084316  0.041333 -0.302620 -0.660377  0.167430 -0.256117  0.382948

             V21       V22       V23       V24       V25       V26       V27  \
0      -0.018307  0.277838 -0.110474  0.066928  0.128539 -0.189115  0.133558
1      -0.225775 -0.638672  0.101288 -0.339846  0.167170  0.125895 -0.008983
2       0.247998  0.771679  0.909412 -0.689281 -0.327642 -0.139097 -0.055353
3      -0.108300  0.005274 -0.190321 -1.175575  0.647376 -0.221929  0.062723
4      -0.009431  0.798278 -0.137458  0.141267 -0.206010  0.502292  0.219422
…          …         …         …         …         …         …         …
284802  0.213454  0.111864  1.014480 -0.509348  1.436807  0.250034  0.943651
284803  0.214205  0.924384  0.012463 -1.016226 -0.606624 -0.395255  0.068472
284804  0.232045  0.578229 -0.037501  0.640134  0.265745 -0.087371  0.004455
284805  0.265245  0.800049 -0.163298  0.123205 -0.569159  0.546668  0.108821
284806  0.261057  0.643078  0.376777  0.008797 -0.473649 -0.818267 -0.002415

             V28    Amount  Class
0      -0.021053  0.244964      0
1       0.014724 -0.342475      0
2      -0.059752  1.160686      0
3       0.061458  0.140534      0
4       0.215153 -0.073403      0
…          …         …       …
284802  0.823731 -0.350151      0
284803 -0.053527 -0.254117      0
284804 -0.026561 -0.081839      0
284805  0.104533 -0.313249      0
284806  0.013649  0.514355      0
```

```
[284807 rows x 30 columns]
```

[9]:
```
df.duplicated().any()
```

[9]:
```
True
```

[10]:
```
df = df.drop_duplicates()
```

[11]:
```
df.shape
```

[11]:
```
(283726, 31)
```

[12]:
```
# Checking for imbalance

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

[12]:
```
Class
0    283253
1       473
Name: count, dtype: int64
```

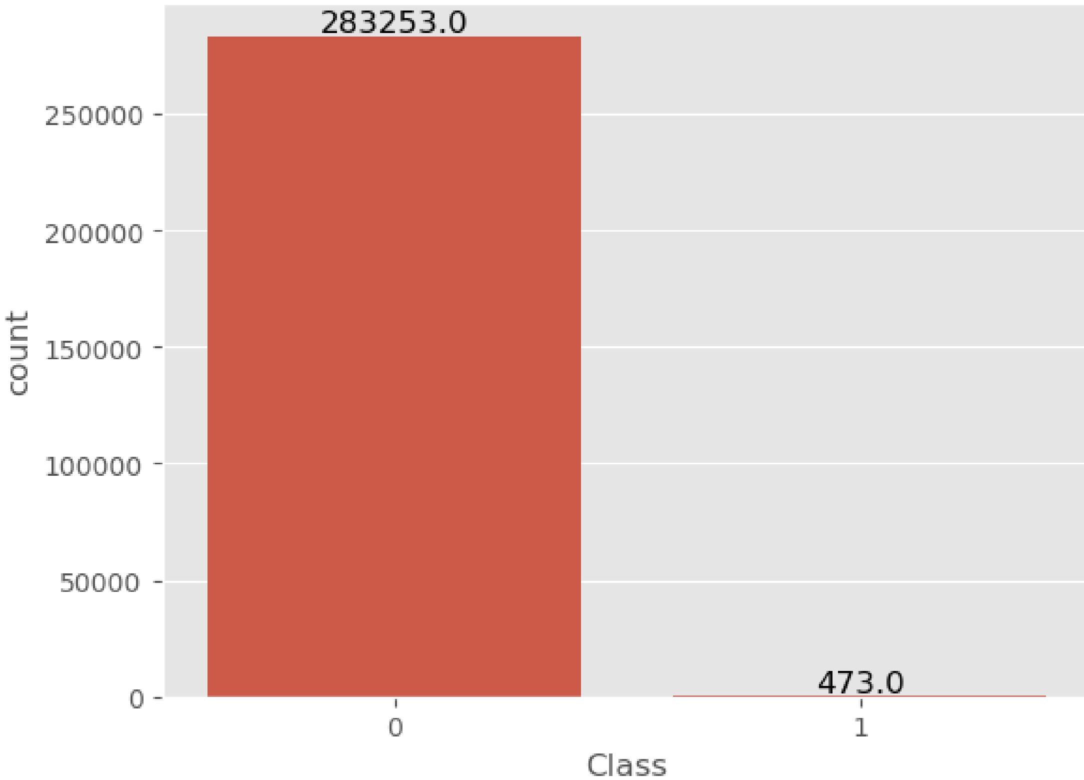
[13]:
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

[14]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Create the countplot
ax = sns.countplot(x=df['Class'])

# Add data labels
for p in ax.patches:
    ax.annotate(f'{p.get_height()}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center',
                fontsize=12, color='black',
                xytext=(0, 5), textcoords='offset points')

plt.show()
```



[15]:
```
X = df.drop('Class',axis=1)
Y = df['Class']
```

[19]:
```
from sklearn.model_selection import train_test_split
```

[20]:
```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.
 ↪2,random_state = 42)
```

[21]:
```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
 ↪recall_score
```

[22]:
```
classifier = {
    "Logistic Regression" : LogisticRegression(),
    "Decision Tree Classifier" : DecisionTreeClassifier()
}

for name,clf in classifier.items():
```

```
    print(f"\n==========={name}======")
    clf.fit(X_train,Y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(Y_test,y_pred)
    print(f"\n Accuracy: {accuracy_score(Y_test,y_pred)}")
    print(f"\n Precision: {precision_score(Y_test,y_pred)}")
    print(f"\n Recall: {recall_score(Y_test,y_pred)}")
    print(f"\n F1 Score: {f1_score(Y_test,y_pred)}")
```

```
===========Logistic Regression======

C:\Users\afros\anaconda3\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: 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(


 Accuracy: 0.999048391076023

 Precision: 0.7647058823529411

 Recall: 0.5777777777777777

 F1 Score: 0.6582278481012658

===========Decision Tree Classifier======

 Accuracy: 0.9990307686885419

 Precision: 0.6842105263157895

 Recall: 0.7222222222222222

 F1 Score: 0.7027027027027027
```

[23]:
```
## Checking with Under sampling to match the lower side data
normal = df[df['Class'] == 0]
fraud = df[df['Class'] == 1]
```

[24]:
```
normal_sample = normal.sample(n=473)
```

[25]:
```python
normal_sample.shape
```

[25]:
```
(473, 31)
```

[26]:
```python
new_under_Sampled = pd.concat([normal_sample,fraud],ignore_index = True)
```

[27]:
```python
new_under_Sampled
```

[27]:
```
         Time        V1        V2        V3        V4        V5        V6  \
0     41639.0  1.161922 -0.476883  1.091284  0.211155 -1.094138  0.063974
1    134971.0 -0.100642  1.234425  0.024438  1.032598  0.744236 -0.360728
2    159612.0  2.035526 -0.085170 -1.173378  0.211055  0.144427 -0.606359
3     29024.0 -0.407065  0.788201  1.551483  1.181414  0.202893 -0.205186
4      2185.0 -0.553649  0.700380  1.134316 -0.070443  0.377868  1.762290
..        …         …         …         …         …         …         …
941  169142.0 -1.927883  1.125653 -4.518331  1.749293 -1.566487 -2.010494
942  169347.0  1.378559  1.289381 -5.004247  1.411850  0.442581 -1.326536
943  169351.0 -0.676143  1.126366 -2.213700  0.468308 -1.120541 -0.003346
944  169966.0 -3.113832  0.585864 -5.399730  1.817092 -0.840618 -2.943548
945  170348.0  1.991976  0.158476 -2.583441  0.408670  1.151147 -0.096695

           V7        V8        V9       V10       V11       V12       V13  \
0   -0.786758  0.242821  1.336610 -0.471299 -0.724728  0.186454 -0.937451
1    0.608881 -0.127264 -0.520564 -0.055720 -0.458310 -0.390206  0.138354
2    0.087886 -0.164929  0.238581  0.222041  0.786657  1.264790  0.457782
3    0.449064  0.081953 -0.578149 -0.142392 -0.410134 -0.478763 -0.682878
4   -0.094326 -0.232484 -0.766323 -0.147612  2.400641  0.905801 -0.151857
..        …         …         …         …         …         …         …
941 -0.882850  0.697211 -2.064945 -5.587794  2.115795 -5.417424 -1.235123
942 -1.413170  0.248525 -1.127396 -3.232153  2.858466 -3.096915 -0.792532
943 -2.234739  1.210158 -0.652250 -3.463891  1.794969 -2.775022 -0.418950
944 -2.208002  1.058733 -1.632333 -5.245984  1.933520 -5.030465 -1.127455
945  0.223050 -0.068384  0.577829 -0.888722  0.491140  0.728903  0.380428

          V14       V15       V16       V17       V18       V19       V20  \
0   -0.348797  0.047390 -0.370529  0.474205 -0.978150  0.112303 -0.160556
1   -0.796478  1.465520 -0.835594  1.467085  0.175582  2.762432  0.395922
2    0.373220 -0.635767  0.108402 -0.626169 -0.408307  0.484958 -0.183367
3    0.462423  1.613420 -0.548843  0.198135 -0.125325  0.293668  0.034179
4    0.661012  2.116675 -1.575973  1.366089 -1.700175  0.667103 -0.109729
..        …         …         …         …         …         …         …
941 -6.665177  0.401701 -2.897825 -4.570529 -1.315147  0.391167  1.252967
942 -5.210141 -0.613803 -2.155297 -3.267116 -0.688505  0.737657  0.226138
943 -4.057162 -0.712616 -1.603015 -5.035326 -0.507000  0.266272  0.247968
944 -6.416628  0.141237 -2.549498 -4.614717 -1.478138 -0.035480  0.306271
945 -1.948883 -0.832498  0.519436  0.903562  1.197315  0.593509 -0.017652
```

```
          V21       V22       V23       V24       V25       V26       V27  \
0   -0.214742 -0.372933  0.118152  0.136635  0.032242  0.974516 -0.023640
1   -0.299127 -0.620174 -0.008826  0.604631 -0.705073  0.583574  0.293141
2   -0.248886 -0.588268  0.289497 -0.383927 -0.285043  0.202634 -0.068878
3    0.189416  0.534401 -0.086630  0.056051 -0.295431 -0.226073  0.171159
4    0.820552  0.083399  0.270325 -0.975188 -1.111250  1.132361  0.208529
..        …         …         …         …         …         …         …
941  0.778584 -0.319189  0.639419 -0.294885  0.537503  0.788395  0.292680
942  0.370612  0.028234 -0.145640 -0.081049  0.521875  0.739467  0.389152
943  0.751826  0.834108  0.190944  0.032070 -0.739695  0.471111  0.385107
944  0.583276 -0.269209 -0.456108 -0.183659 -0.328168  0.606116  0.884876
945 -0.164350 -0.295135 -0.072173 -0.450261  0.313267 -0.289617  0.002988

          V28    Amount  Class
0    0.009608 -0.307331      0
1    0.256360 -0.346073      0
2   -0.073233 -0.349271      0
3    0.150457 -0.277666      0
4    0.115114 -0.198503      0
..        …         …     …
941  0.147968  1.206024      1
942  0.186637 -0.350191      1
943  0.194361 -0.041818      1
944 -0.253700  0.626302      1
945 -0.015309 -0.183191      1

[946 rows x 31 columns]
```

[29]:
```
new_under_Sampled['Class'].value_counts()

## Problem for this is we have lost around 250000 data record
## so I would suggest to go for over sampling
```

[29]:
```
Class
0    473
1    473
Name: count, dtype: int64
```

[30]:
```
X = new_under_Sampled.drop('Class',axis=1)
Y = new_under_Sampled['Class']
```

[31]:
```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.
 ↪2,random_state = 42)
```

[32]:
```
classifier = {
    "Logistic Regression" : LogisticRegression(),
    "Decision Tree Classifier" : DecisionTreeClassifier()
```

```
}

for name,clf in classifier.items():
    print(f"\n==========={name}======")
    clf.fit(X_train,Y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(Y_test,y_pred)
    print(f"\n Accuracy: {accuracy_score(Y_test,y_pred)}")
    print(f"\n Precision: {precision_score(Y_test,y_pred)}")
    print(f"\n Recall: {recall_score(Y_test,y_pred)}")
    print(f"\n F1 Score: {f1_score(Y_test,y_pred)}")



## Problem for this is we have lost around 250000 data record
## so I would suggest to go for over sampling
```

```
===========Logistic Regression======

 Accuracy: 0.9526315789473684

 Precision: 0.979381443298969

 Recall: 0.9313725490196079

 F1 Score: 0.9547738693467337

===========Decision Tree Classifier======

 Accuracy: 0.9052631578947369

 Precision: 0.92

 Recall: 0.9019607843137255

 F1 Score: 0.9108910891089109
C:\Users\afros\anaconda3\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: 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(
```

[33]:
```python
## Now lets try for over sampling lets use SMOT


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

[34]:
```python
from imblearn.over_sampling import SMOTE
```

[35]:
```python
X_res ,Y_res = SMOTE().fit_resample(X,Y)
```

[37]:
```python
Y_res.value_counts()
```

[37]:
```
Class
0    283253
1    283253
Name: count, dtype: int64
```

[38]:
```python
X_train,X_test,Y_train,Y_test = train_test_split(X_res,Y_res,test_size = 0.
↪2,random_state = 42)
```

[39]:
```python
classifier = {
    "Logistic Regression" : LogisticRegression(),
    "Decision Tree Classifier" : DecisionTreeClassifier()
}

for name,clf in classifier.items():
    print(f"\n==========={name}======")
    clf.fit(X_train,Y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(Y_test,y_pred)
    print(f"\n Accuracy: {accuracy_score(Y_test,y_pred)}")
    print(f"\n Precision: {precision_score(Y_test,y_pred)}")
    print(f"\n Recall: {recall_score(Y_test,y_pred)}")
    print(f"\n F1 Score: {f1_score(Y_test,y_pred)}")




## Problem for this is we have lost around 250000 data record
## so I would suggest to go for over sampling
```

```
===========Logistic Regression======
```

```
C:\Users\afros\anaconda3\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: 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(


 Accuracy: 0.9731160968032339

 Precision: 0.9837761030271418

 Recall: 0.9622794208202115

 F1 Score: 0.9729090326941549

===========Decision Tree Classifier======

 Accuracy: 0.9982789359411132

 Precision: 0.9974182444061962

 Recall: 0.999155509421348

 F1 Score: 0.9982861210965309
```

[40]:
```python
dtc = DecisionTreeClassifier()
dtc.fit(X_res,Y_res)
```

[40]: DecisionTreeClassifier()

[42]:
```python
import joblib

# Assuming dtc is your trained DecisionTreeClassifier model
joblib.dump(dtc, 'credit_card_model.pkl')
```

[42]: ['credit_card_model.pkl']

[43]:
```python
model = joblib.load("credit_card_model.pkl")
```

[45]:
```python
# Assuming `model` is your trained model and you're predicting on one row

pred = model.predict([[
    -1.3598071336738, -0.0727811733098497, 2.53634673796914, 1.37815522427443,
    -0.338320769942518, 0.462387777762292, 0.239598554061257, 0.
 ↪0986979012610507,
```

```
    0.363786969611213, 0.0907941719789316, -0.551599533260813, -0.
↪617800855762348,
    -0.991389847235408, -0.311169353699879, 1.46817697209427, -0.
↪470400525259478,
    0.207971241929242, 0.0257905801985591, 0.403992960255733, 0.251412098239705,
    -0.018306777944153, 0.277837575558899, -0.110473910188767, 0.
↪0669280749146731,
    0.128539358273528, -0.189114843888824, 0.133558376740387, -0.
↪0210530534538215,
    149.62,  # Make sure all features are included, like V29, V30, if any
    0.5       # Example of another missing feature, just as an illustration
]])
```

```
C:\Users\afros\anaconda3\Lib\site-packages\sklearn\base.py:493: UserWarning: X
does not have valid feature names, but DecisionTreeClassifier was fitted with
feature names
  warnings.warn(
```

[46]:

```
pred
```

[46]:

```
array([0], dtype=int64)
```

[47]:

```
if pred == 0:
    print("Normal Transaction")
else:
    print("Fraud Transaction")
```

```
Normal Transaction
```