Unbalance Dataset

May 20, 2022

1 Load data

[60]:
```python
import pandas as pd
```

[61]:
```python
df = pd.read_csv("creditcard.csv")
```

2 EDA

[62]:
```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
```

[63]:

```python
df.describe()
```

[63]:

```
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  3.918649e-15  5.682686e-16 -8.761736e-15  2.811118e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean  -1.552103e-15  2.040130e-15 -1.698953e-15 -1.893285e-16 -3.147640e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       …           V21           V22           V23           V24  \
count  …  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   …  1.473120e-16  8.042109e-16  5.282512e-16  4.456271e-15
std    …  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    … -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    … -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    … -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    …  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    …  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   1.426896e-15  1.701640e-15 -3.662252e-16 -1.217809e-16      88.349619
```

```
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

The data represents credit card transactions that occurred over two days in September 2013 by European cardholders. V1-V28 represent features that maybe were obtained using a PCA approach. This is also a common practise due to privacy reasons. Imagine having private data, at least in theory, the data must be hashed.

Feature **Time** contains the seconds elapsed between each transaction and the first transaction in the dataset.

The feature **Amount** is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning.

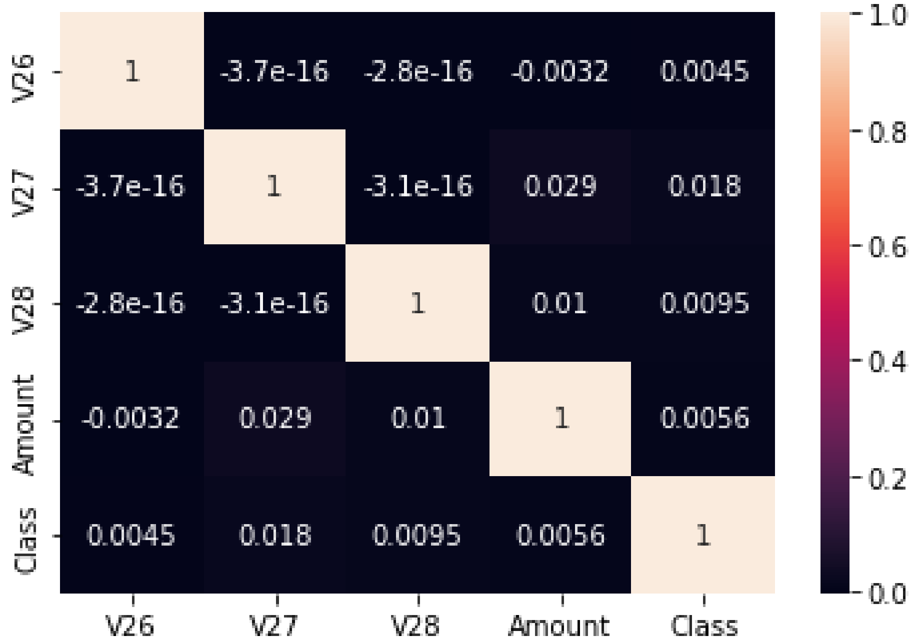
Feature **Class** is the response variable and it takes value 1 in case of fraud and 0 otherwise.

[64]:
```python
# let's check correlation between features
```

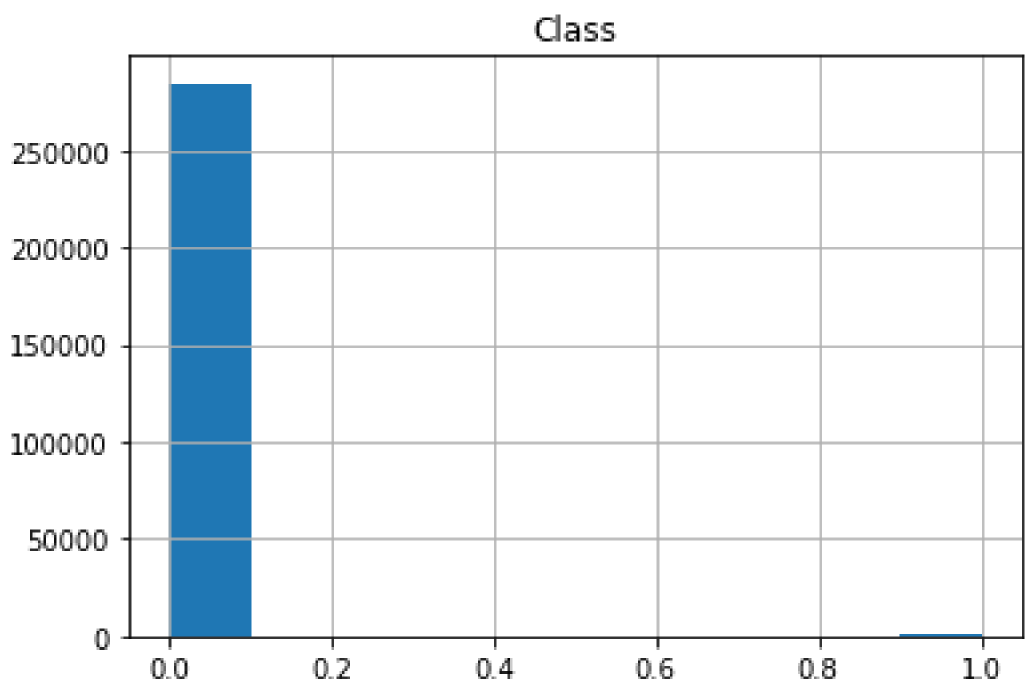
[65]:
```python
import seaborn as sns

Var_Corr = df[df.columns[-5:]].corr()
# plot the heatmap and annotation on it
sns.heatmap(Var_Corr, xticklabels=Var_Corr.columns, yticklabels=Var_Corr.
 ↪columns, annot=True)
# Not a huge help because we have a lot of features
```

[65]: <AxesSubplot:>



[66]:
```
# let's check class imbalance
values = df[["Class"]].hist()
```



[67]:
```python
import numpy as np
values = np.histogram(df[["Class"]], bins=2)
max_ratio = values[0][0] / values[0][1]
print(f"ratio {max_ratio}:1")
print(f"majority class has {(values[0][0] / sum(values[0]))*100} %")
```

```
ratio 577.8760162601626:1
majority class has 99.82725143693798 %
```

Given the above plot we have a huge imbalance dataset. With a ratio of 578 : 1, with the majority class having 99.83 this means that if we opt by a random model we would have a accuracy of 99.83. So this our new baseline :)

3 Data Preprocessing

Since our dataset does not have missing data we do not need to fill.

4 Data engineering

[68]:
```python
df.describe()
```

[68]:
```
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  3.918649e-15  5.682686e-16 -8.761736e-15  2.811118e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean  -1.552103e-15  2.040130e-15 -1.698953e-15 -1.893285e-16 -3.147640e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       …           V21           V22           V23           V24  \
count  …  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   …  1.473120e-16  8.042109e-16  5.282512e-16  4.456271e-15
std    …  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    … -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    … -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
```

```
50%    … -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    …  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    …  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   1.426896e-15  1.701640e-15 -3.662252e-16 -1.217809e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

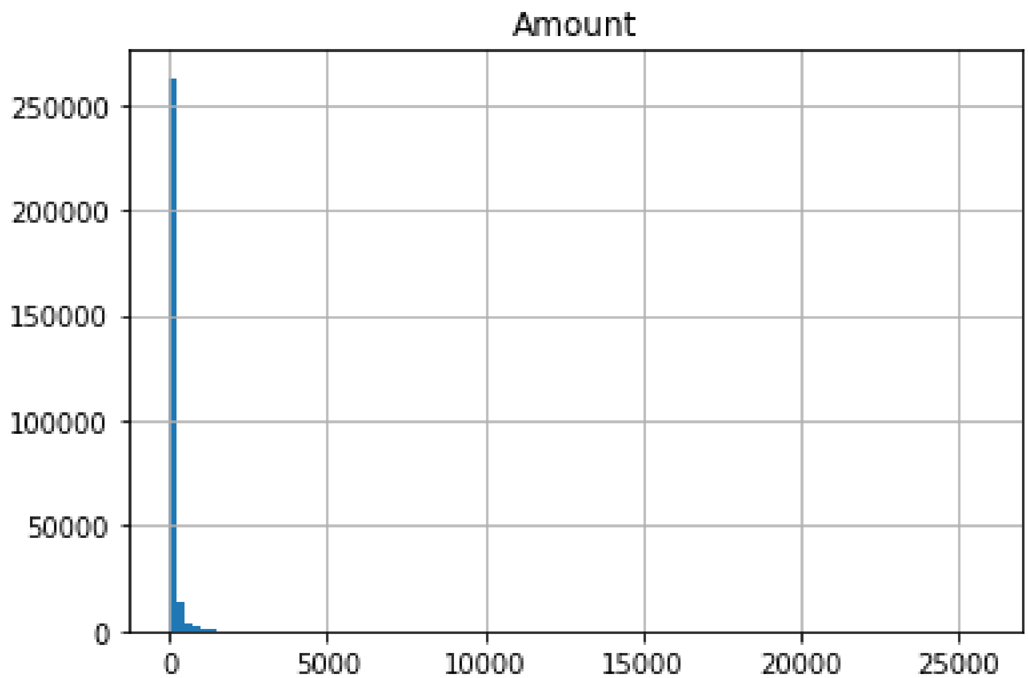
[69]:
```
# Let's plot Time to understand better the distribution

# since time is dependent on the previous row we need to remove that feature
df.drop("Time", axis=1, inplace=True)
```

[70]:
```
import matplotlib.pyplot as plt
```

[71]:
```
df[["Amount"]].hist(bins=100)
```

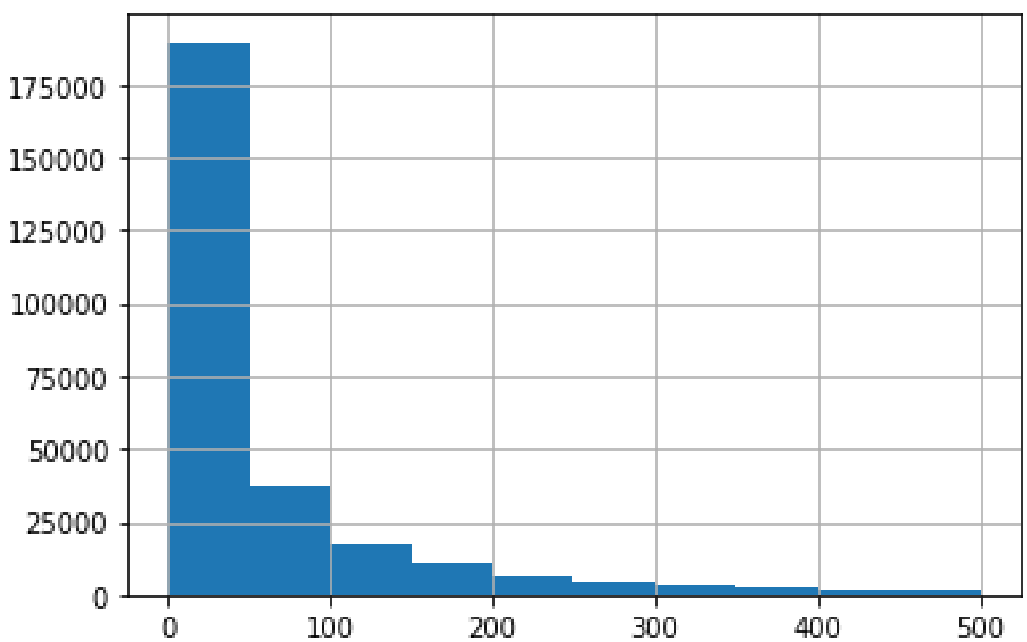
[71]:
```
array([[<AxesSubplot:title={'center':'Amount'}>]], dtype=object)
```



From the above plot we can conclude that we have a very skewd dataset.

[72]:
```
df[df["Amount"] < 500]["Amount"].hist()
```

[72]: <AxesSubplot:>



Even by plot just transaction below 500 we confirm our thesis of having a very skew dataset

[73]:
```
df["AmountBand"] = pd.cut(df["Amount"], 200)
```

[74]:
```
df["AmountBand"].value_counts()
```

[74]:
```
(-25.691, 128.456]        238673
(128.456, 256.912]         24117
(256.912, 385.367]          8707
(385.367, 513.823]          4495
(513.823, 642.279]          2481
                           …
(13487.859, 13616.315]         0
(13359.403, 13487.859]         0
(13230.947, 13359.403]         0
(13102.492, 13230.947]         0
(16827.71, 16956.166]          0
Name: AmountBand, Length: 200, dtype: int64
```

[75]:
```
from sklearn.preprocessing import LabelEncoder
```

[76]:
```
le = LabelEncoder()
df["AmountBand"] = le.fit_transform(df["AmountBand"])
```

[77]:
```
# S
max(df.AmountBand)
# Since we have 67 bands, we should normalize the dataset, because dependning␣
 ↪on the dataset, the real value of the data may affect
```

[77]:
```
67
```

5 Split dataset

[78]:
```
from sklearn.model_selection import train_test_split
```

[79]:
```
y = df["Class"]
```

[80]:
```
X = df.drop(["Amount", "Class"], axis=1)
```

[81]:
```
X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, random_state=42)
```

[82]:
```
y_train.value_counts()
```

[82]:
```
0    190477
1       343
Name: Class, dtype: int64
```

6 Apply Random Forest

[23]:
```python
from sklearn.ensemble import RandomForestClassifier
```

[24]:
```python
RF_clf = RandomForestClassifier()
```

[25]:
```python
model = RF_clf.fit(X_train, y_train)
```

[26]:
```python
preds = model.predict(X_test)
```

[27]:
```python
# Let's check results
```

[28]:
```python
from sklearn.metrics import classification_report
```

[29]:
```python
print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     93838
           1       0.95      0.79      0.86       149

    accuracy                           1.00     93987
   macro avg       0.98      0.90      0.93     93987
weighted avg       1.00      1.00      1.00     93987
```

Here, since we are in a fraud situation, what we are really converned is about recall, because we need to maximize the fraud cases that we find

7 Let's apply several tecniques to improve the imbalance situation

[83]:
```python
from imblearn.under_sampling import RandomUnderSampler
```

[84]:
```python
random_under_sampler = RandomUnderSampler()
```

[89]:
```python
y_train.value_counts()
```

[89]:
```
0    190477
1       343
Name: Class, dtype: int64
```

[90]:
```python
X_under, y_under = random_under_sampler.fit_resample(X_train, y_train)
```

[91]:
```
y_under.value_counts() # confirm that sampling was made
```

[91]:
```
0    343
1    343
Name: Class, dtype: int64
```

[92]:
```
model = RF_clf.fit(X_under, y_under)
preds = model.predict(X_test)
print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     93838
           1       0.04      0.93      0.07       149

    accuracy                           0.96     93987
   macro avg       0.52      0.95      0.53     93987
weighted avg       1.00      0.96      0.98     93987
```

8 Let's try upper sampling

[93]:
```
from imblearn.over_sampling import RandomOverSampler
```

[94]:
```
random_over_sampler = RandomOverSampler()
```

[95]:
```
X_over, y_over = random_over_sampler.fit_resample(X_train, y_train)
```

[96]:
```
y_over.value_counts() # confirm that sampling was made
```

[96]:
```
0    190477
1    190477
Name: Class, dtype: int64
```

[97]:
```
model = RF_clf.fit(X_over, y_over)
preds = model.predict(X_test)
print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     93838
           1       0.93      0.81      0.87       149

    accuracy                           1.00     93987
   macro avg       0.97      0.91      0.93     93987
weighted avg       1.00      1.00      1.00     93987
```

9 Let's try SMOTE

[98]:
```
from imblearn.over_sampling import SMOTE
```

[99]:
```
SMOTE_sampler = SMOTE(random_state=42)
```

[100]:
```
X_smote, y_smote = SMOTE_sampler.fit_resample(X_train, y_train)
```

[101]:
```
y_smote.value_counts() # confirm that sampling was made
```

[101]:
```
0    190477
1    190477
Name: Class, dtype: int64
```

[102]:
```
model = RF_clf.fit(X_smote, y_smote)
preds = model.predict(X_test)
print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     93838
           1       0.83      0.87      0.85       149

    accuracy                           1.00     93987
   macro avg       0.91      0.93      0.92     93987
weighted avg       1.00      1.00      1.00     93987
```

[106]:
```
# It is possible to try different sampling ratios
SMOTE_sampler = SMOTE(random_state=42,
                      sampling_strategy=0.5)
```

[107]:
```
X_smote, y_smote = SMOTE_sampler.fit_resample(X_train, y_train)
```

[108]:
```
y_smote.value_counts() # confirm that sampling was made
```

[108]:
```
0    190477
1     95238
Name: Class, dtype: int64
```

[109]:
```
model = RF_clf.fit(X_smote, y_smote)
preds = model.predict(X_test)
print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     93838
           1       0.84      0.87      0.85       149
```

```
    accuracy                           1.00     93987
   macro avg       0.92      0.93      0.93     93987
weighted avg       1.00      1.00      1.00     93987
```

[ ]: