Youssef EL OUARDIGHI

BAHCESEHIR UNIVERSITY

DM5101

Card Fraud Detection Project

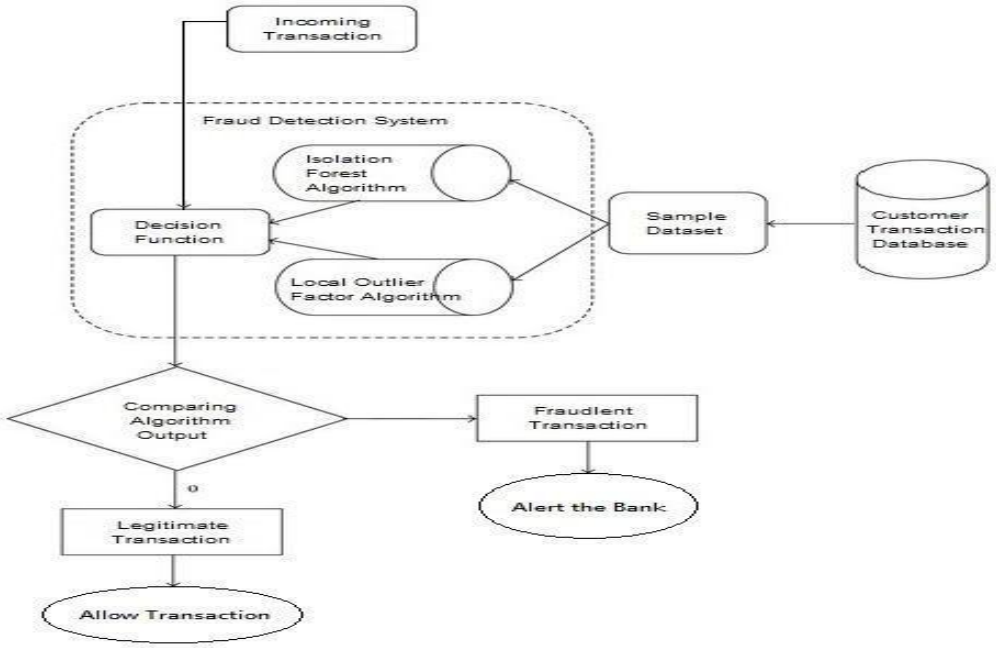
Table of content :

Abstract :

For banks, the main goal is to retain its customers. However banking fraud is a huge threat to these banks in terms of financial losses, credibility and trust, this is a concerning issue to both customers and banks alike. Card fraud detection using machine learning is a necessity for these banks to monitor and prevent fraud from happening. This project intends to illustrate the modeling of a data set using machine learning with Credit Card Fraud Detection. The Credit Card Fraud Detection Problem includes modeling past credit card transactions with the data of the ones that turned out to be fraud. This model is then used to recognize whether a new transaction is fraudulent or not. Our objective here is to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications. In this process, we have focused on analyzing and preprocessing data set as well as the deployment of multiple anomaly detection algorithms such as Logistic Regression and Random forest on the PCA transformed Credit Card Transaction data.

Introduction :

In credit card transactions, 'fraud' refers to the unlawful and unwelcome use of a card account by someone who is not the cardholder. To stop this misuse, necessary preventative steps should be adopted, and the behavior of such fraudulent acts can be analyzed to decrease it and defend against future occurrences. In other words, credit card fraud occurs when a person uses another person's credit card for personal gain while the owner and card issuing authorities are unaware of the transaction. Fraud detection is tracking the behaviors of large groups of people in order to predict, detect, or avert unacceptable behavior such as fraud, intrusion, or defaulting. This is a very important problem that requires the attention of new technologies like machine learning and data science, where the answer can be automated. This issue is particularly difficult to solve from the standpoint of education because it is characterized by many elements such as class imbalance. The number of legitimate transactions considerably outnumbers the number of fraudulent transactions. Furthermore, transaction patterns frequently modify their statistical features over time. However, these aren't the only difficulties that come with implementing a real-world fraud detection system. In real-world scenarios, automatic tools scan a vast stream of payment requests to identify which transactions to authorize. To analyze all allowed transactions and report suspect ones, machine learning techniques are used. Professionals evaluate these reports and call cardholders to confirm whether the transaction was legitimate or fraudulent. The investigators submit feedback to the automated system, which is then utilized to train and update the algorithm in order to enhance fraud detection over time.



Methods for detecting fraud are always being improved in order to protect criminals from altering their fraudulent schemes. These deceptions are categorized as follows:

- Credit Card Frauds: Online and Offline
- Card Theft
- Account Bankruptcy
- Device Intrusion
- Application Fraud
- Counterfeit Card
- Telecommunication Fraud

The following are some of the existing methods for detecting such fraud:

- Artificial Neural Network
- Fuzzy Logic
- Genetic Algorithm
- Logistic Regression
- Decision tree
- Support Vector Machines
- Bayesian Networks
- Hidden Markov Model
- K-Nearest Neighbour

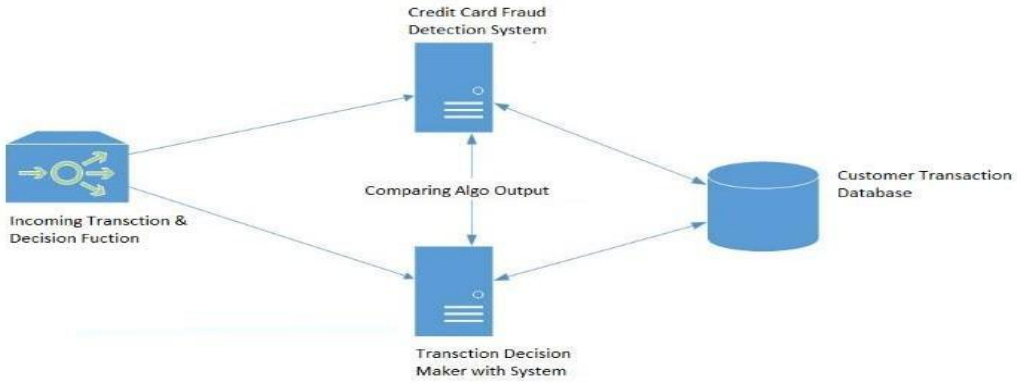
For this project I'll use different algorithms and compare the results to see which one is the most accurate. The algorithms I chose for this project are :

- Logistic regression
- Random forest
- XGB classifier

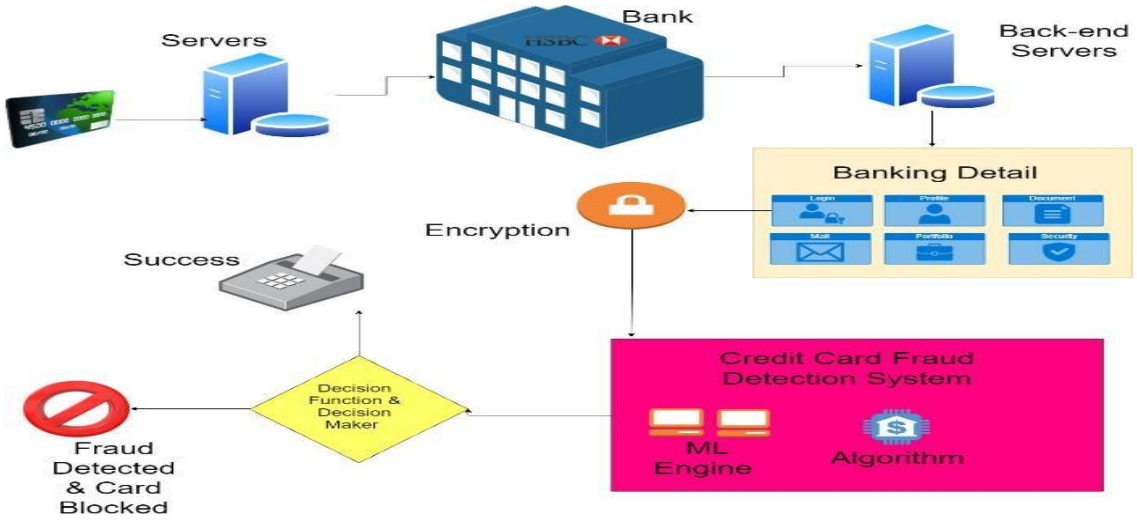
Given the imbalance of the data I measured the accuracy with F1 score, the confusion matrix isn't relevant in this case only when I did an undersampling method.

Methodology :

The approach proposed in this research employs the most up-to-date machine learning algorithms to detect aberrant activities known as outliers. The following is a representation of the fundamental rough architecture diagram:

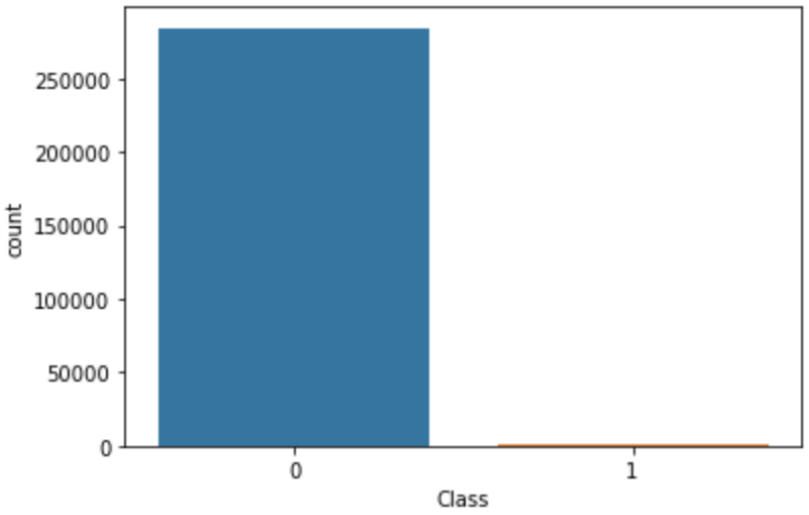


The whole architecture diagram can be rendered as follows when viewed on a bigger scale with real-life elements:

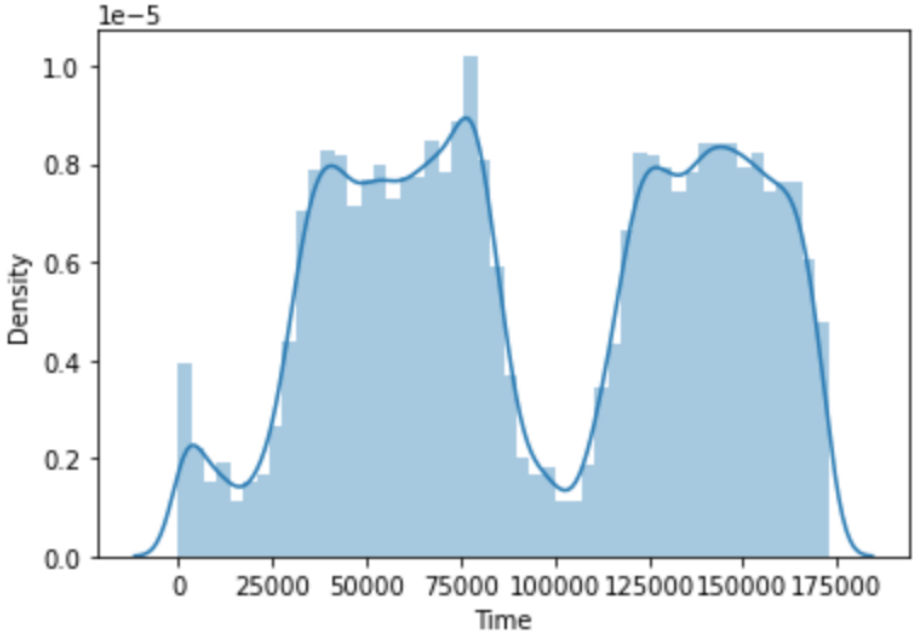


Data information :

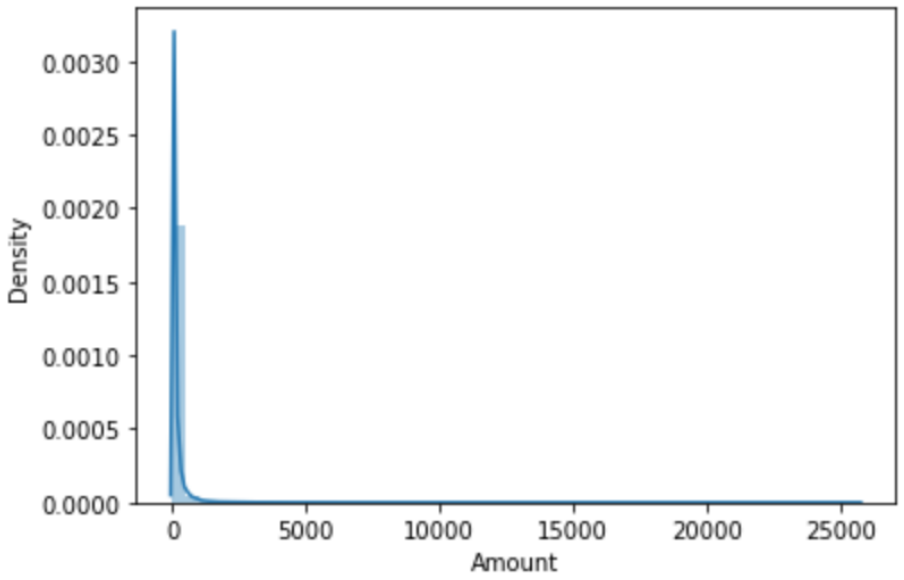
First and foremost, we got our data from Kaggle, a data analysis service that offers datasets. There are 31 columns in this dataset, with 28 of them labeled v1-v28 to preserve sensitive information. Time, Amount, and Class are represented by the other columns. The time difference between the first and subsequent transactions is shown in this graph. The amount of money exchanged is referred to as the amount. A genuine transaction is represented by class 0, while a fraudulent transaction is represented by class 1.



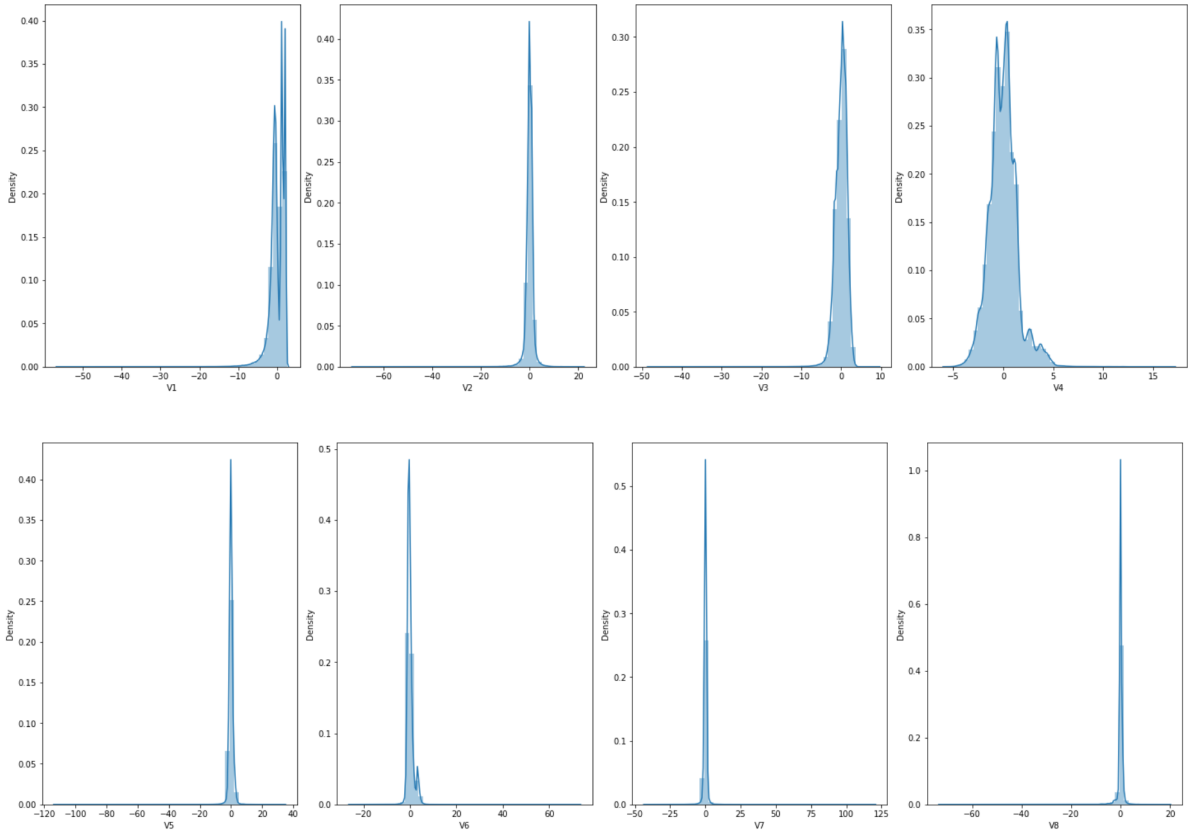
The number of fraudulent transactions is substantially fewer than the number of valid transactions, as shown in this graph. (284 807 transactions)



This graph depicts the times when transactions were completed in less than two days. The lowest number of transactions were made at night, while the biggest number were made during the day.



This graph depicts the total amount of money exchanged. The bulk of transactions are tiny, and just a few come close to exceeding the maximum transaction value.
As for the other features we don't have much information about them because of the pca protocol for confidentiality purposes.



But before having all this information, I needed to import the necessary libraries in order to have this information on the dataset but also to import the file containing the data on the notebook.

Importing the dependencies

```
# Importing the libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn import metrics
# To ignore warnings
import warnings
warnings.filterwarnings("ignore")
```

Loading the data

```
# Loading the data
cdf = pd.read_csv('creditcard.csv')
```

Now that I've got what I need I can start exploring the dataset.

Data exploratory :

To start, I displayed the first 5 rows of the dataset as well as the last 5.

```
cdf.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V
0	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	0.1285
1	0.0	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	0.1671
2	1.0	-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	-0.3276
3	1.0	-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	0.6473
4	2.0	-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	-0.2060

5 rows × 31 columns

```
# 5 last rows
cdf.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797

Then I wanted to have some statistical information on this data like the maximum value.

```
cdf.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	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	-1.552103e-15	2.040130e-15	-1.698953e-15	-1.893285e-16	-3.147640e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01

We notice that the range between the values is too large, this is because the data is not balanced.

I had to see if I had any null values on my dataset to correct them afterwards, but as we can see I don't have any null values so I can continue my work.

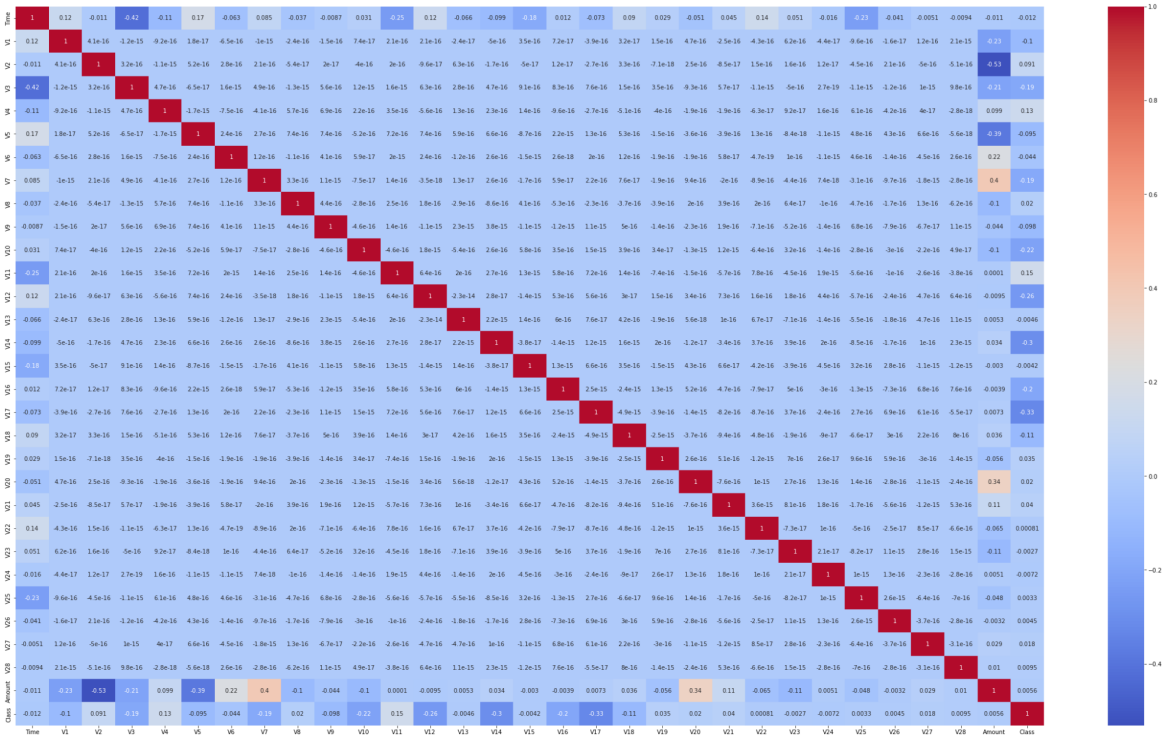
```
cdf.isnull().sum()
```

```
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
```

```
<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
```

As you can see, I don't need to clean the data because I don't have null values and don’t need to do the encoding part because I only have numerical values.

Correlation matrices are crucial to comprehending our data. We want to know if there are any characteristics that have a significant impact on whether or not a transaction is fraudulent. We create a heatmap to visualize the data in color and investigate the relationship between our predictive factors and the class variable. The following is a heatmap:



As we can see, this correlation matrix doesn't help us that much and what we need to do to fix this is to transform our data a bit in order to have better results. For this, I have chosen two methods:

- Undersampling
- Oversampling

Methods and models :

For this method, I must first use the standard scaling to have a better range of values. For that I will use standard scaler.

```
Entrée [16]: from sklearn.preprocessing import StandardScaler
             sc = StandardScaler()
             X_scaler = sc.fit_transform(X)
```

```
Entrée [17]: X_scaler[-1]
```

```
Out[17]: array([ 1.64205773, -0.27233093, -0.11489898,  0.46386564, -0.35757   ,
                -0.00908946, -0.48760183,  1.27476937, -0.3471764 ,  0.44253246,
                -0.84072963, -1.01934641, -0.0315383 , -0.18898634, -0.08795849,
                 0.04515766, -0.34535763, -0.77752147,  0.1997554 , -0.31462479,
                 0.49673933,  0.35541083,  0.8861488 ,  0.6033653 ,  0.01452561,
                -0.90863123, -1.69685342, -0.00598394,  0.04134999,  0.51435531])
```

as you can see the values now go from 2 to -2 whereas before the range of values was bigger. After scaling my data, I can now split my data into training and testing using the known ratio of 0.2 for testing and the rest for training.
Then, I wanted to have an idea on the results that I could have by using the data as it is without modification.

```
: from sklearn.linear_model import LogisticRegression
  modl = LogisticRegression()
  # training
  modl.fit(X_train, y_train)
  # testing
  y_pred1 = modl.predict(X_test)
  print(classification_report(y_test, y_pred1))
  print("F1 Score : ", f1_score(y_test, y_pred1))
```

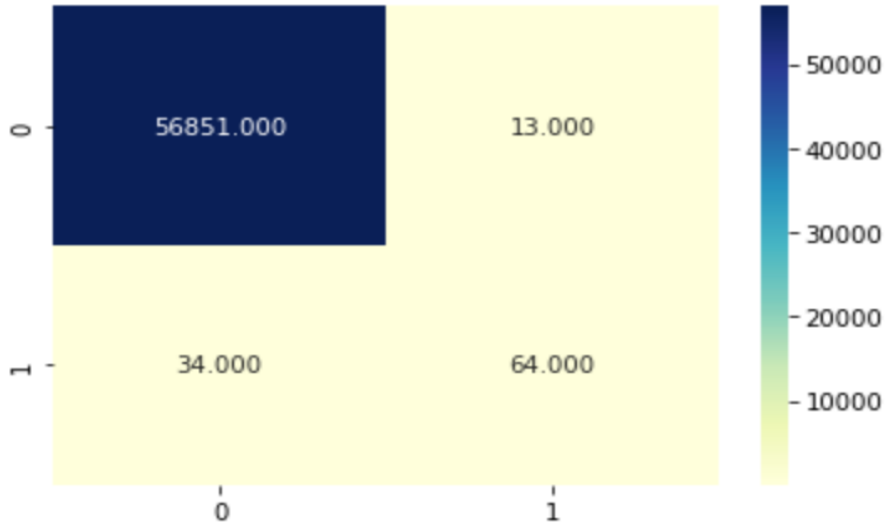
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.83      0.65      0.73        98

    accuracy                           1.00     56962
   macro avg       0.92      0.83      0.87     56962
weighted avg       1.00      1.00      1.00     56962

F1 Score :  0.7314285714285713
```

In a confusion matrix, Trained Data and Testing Data are depicted as follows:
TP: True Positive, which denotes real data from customers who have been subjected to fraud and have been accurately anticipated.
TN: True Negative represents data that was not expected and does not correspond to the data that was manipulated.
FP: False Positive is expected, but there is no chance that the data will be tampered with.
FN: False Negative is not anticipated, but there is a chance that the data has been tampered with.



Then, I did the same thing for the other two models I chose.

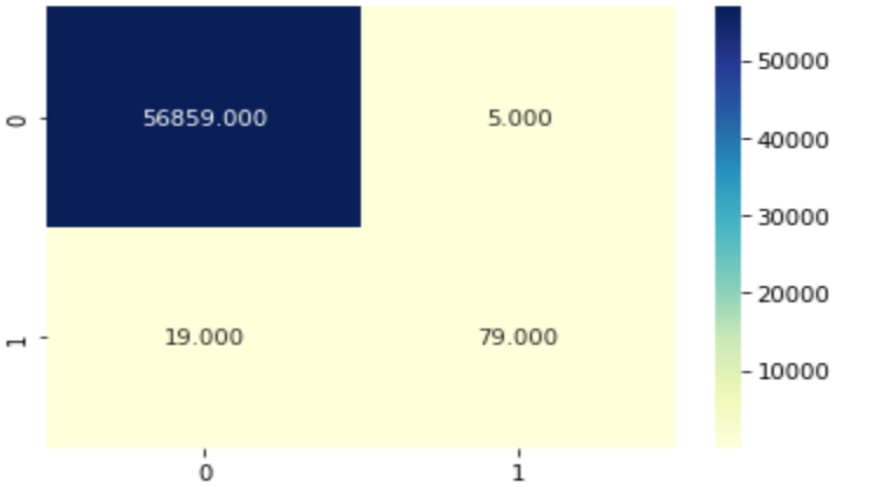
```
Entrée [65]: from sklearn.ensemble import RandomForestClassifier
             modl = RandomForestClassifier()
             # training
             modl.fit(X_train, y_train)
             # testing
             y_pred2 = modl.predict(X_test)
             print(classification_report(y_test, y_pred2))
             print("F1 Score : ", f1_score(y_test, y_pred2))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.94      0.81      0.87        98

    accuracy                           1.00     56962
   macro avg       0.97      0.90      0.93     56962
weighted avg       1.00      1.00      1.00     56962

F1 Score :  0.8681318681318683
```



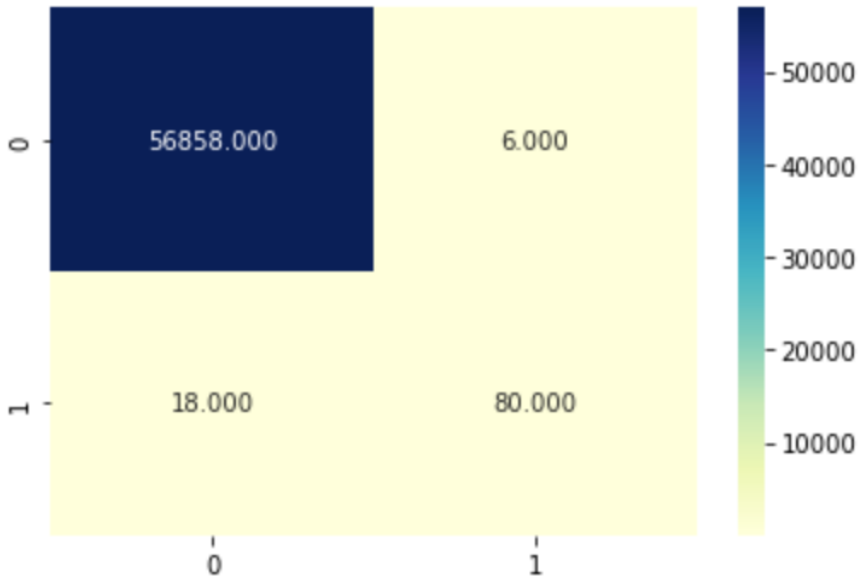
```
Entrée [66]: from xgboost import XGBClassifier
             modl = XGBClassifier(n_jobs=-1)
             # training
             modl.fit(X_train, y_train)
             # testing
             y_pred3 = modl.predict(X_test)
             print(classification_report(y_test, y_pred3))
             print("F1 Score : ", f1_score(y_test, y_pred3))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.93      0.82      0.87        98

    accuracy                           1.00     56962
   macro avg       0.96      0.91      0.93     56962
weighted avg       1.00      1.00      1.00     56962

F1 Score :  0.8695652173913043
```



As expected, the results obtained were not at all satisfactory, so I decided to use two methods because the structure of the dataset forces me to do so. The methods we will use are oversampling and undersampling.

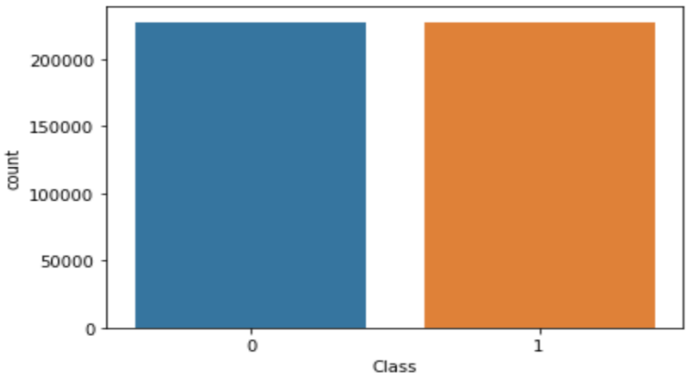
Oversampling :

To do the oversampling we will use the smote technique. SMOTE is an acronym for Synthetic Minority Over-sampling Technique. SMOTE, unlike Random UnderSampling, develops new synthetic points in order to achieve a balanced distribution of classes. This is yet another option for resolving the "class imbalance issues."

Entrée [67]:
```
# Balance the class with equal distribution
from imblearn.over_sampling import SMOTE
over_sample = SMOTE()
x_smote, y_smote = over_sample.fit_resample(X_train, y_train)
```

Entrée [68]:
```
sns.countplot(y_smote)
```

Out[68]: <AxesSubplot:xlabel='Class', ylabel='count'>

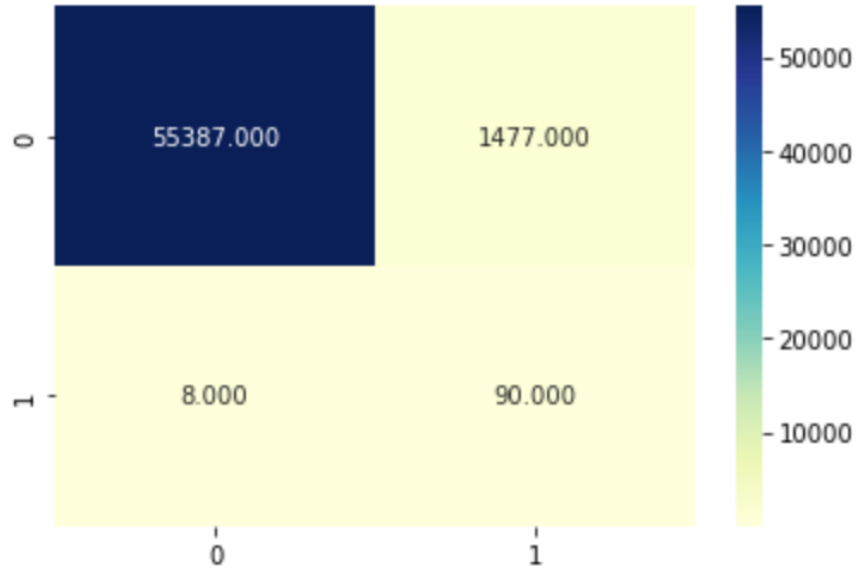


Now we can see that both classes i.e. legit and fraud transactions are the same, we can implement our models and see the results obtained.

Logistic Regression :

	precision	recall	f1-score	support
0	1.00	0.97	0.99	56864
1	0.06	0.92	0.11	98
accuracy			0.97	56962
macro avg	0.53	0.95	0.55	56962
weighted avg	1.00	0.97	0.99	56962

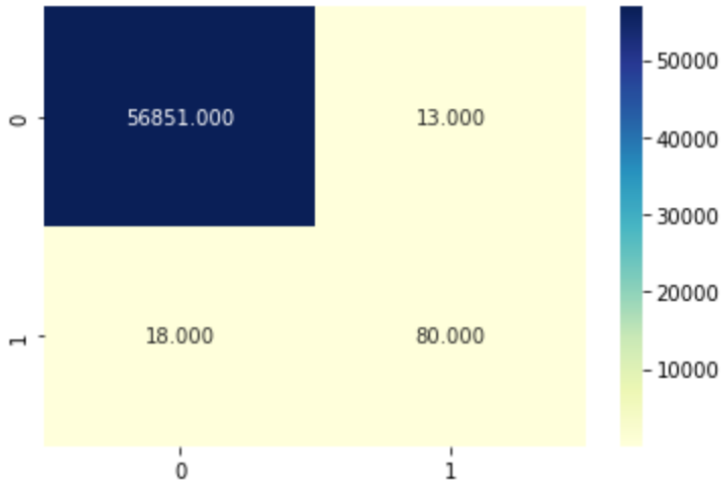
F1 Score : 0.10810810810810811



Random forest :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.86	0.82	0.84	98
accuracy			1.00	56962
macro avg	0.93	0.91	0.92	56962
weighted avg	1.00	1.00	1.00	56962

F1 Score : 0.837696335078534



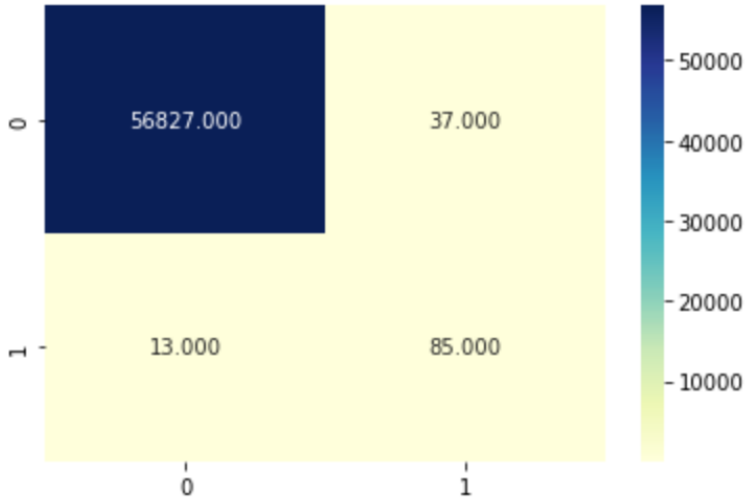
XGBclassifier:

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.70      0.87      0.77        98

    accuracy                           1.00     56962
   macro avg       0.85      0.93      0.89     56962
weighted avg       1.00      1.00      1.00     56962

F1 Score :  0.7727272727272727
```



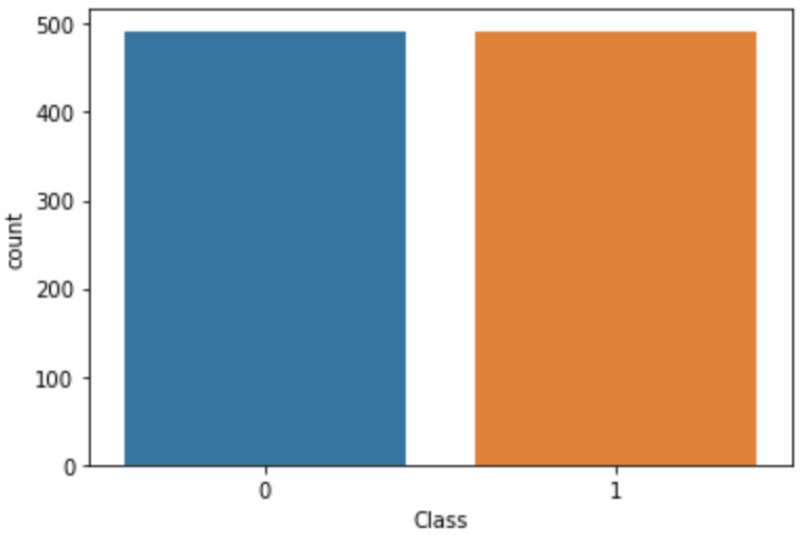
As we can see, the results obtained are not promising. Compared to the ones before, they are less so I will try the other method and see what I will get as results.

Undersampling :

For this method, I will first randomly select the data from the dataset to have a coherent data, then I will create a new dataset and do my study for this method.

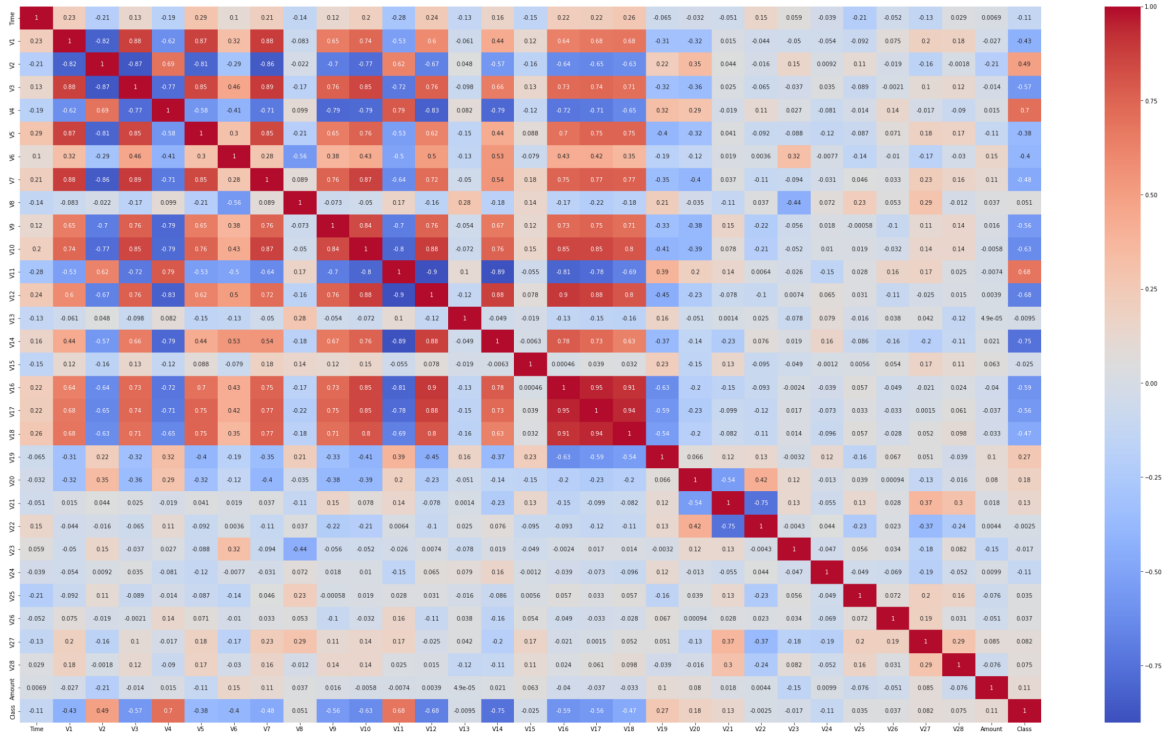
```
normal_sample = normal.sample(n=492)
```

```
# Concatenate the dataframes
new_ds = pd.concat([normal_sample, fraud], axis=0)
```

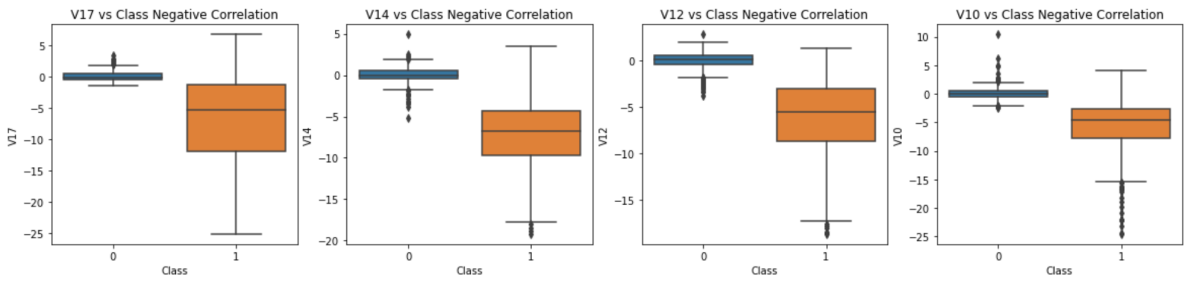


Now that our data is balanced, we can see on the correlation matrix that :

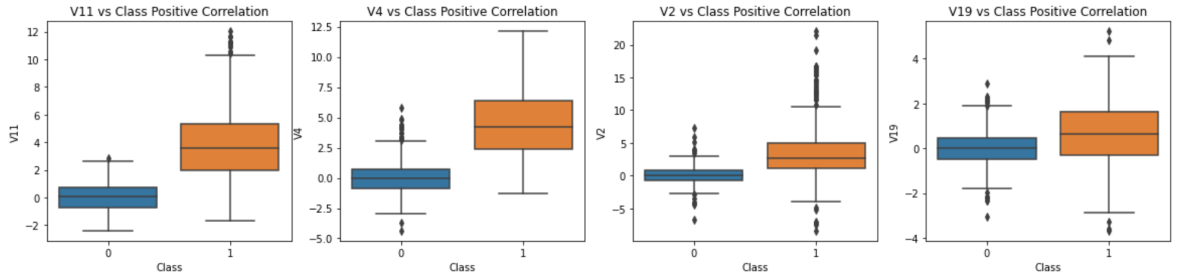
- **Negative Correlations:** V17, V14, V12 and V10 are negatively correlated. Notice how the lower these values are, the more likely the end result will be a fraud transaction.
- **Positive Correlations:** V2, V4, V11, and V19 are positively correlated. Notice how the higher these values are, the more likely the end result will be a fraud transaction.



We will use boxplots to have a better understanding of the distribution of these features in fraudulent and non fraudulent transactions.



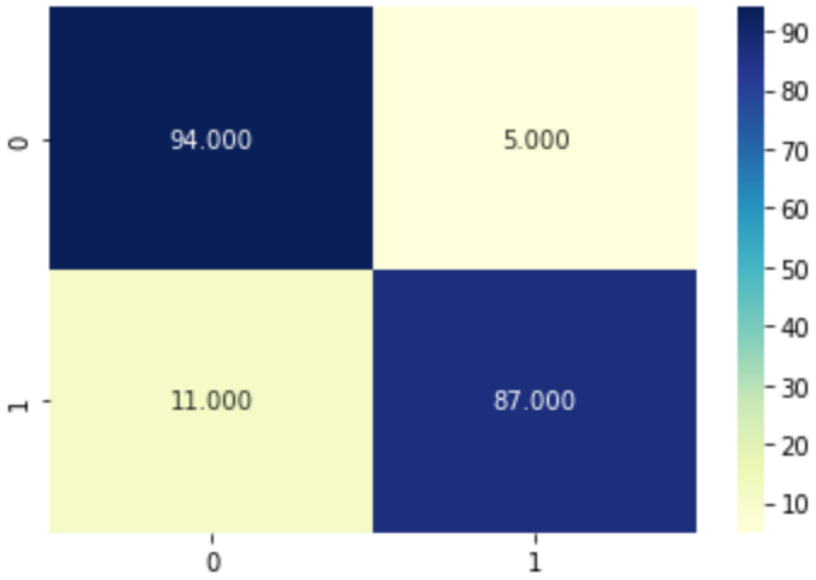
As we can see here the lower the value of the feature the more likely it will be a fraud.



On the contrary here the higher the value, the more likely it will be a legit transaction.
Now we can implement our models and see the results that I got.
Logistic Regression :

	precision	recall	f1-score	support
0	0.90	0.95	0.92	99
1	0.95	0.89	0.92	98
accuracy			0.92	197
macro avg	0.92	0.92	0.92	197
weighted avg	0.92	0.92	0.92	197

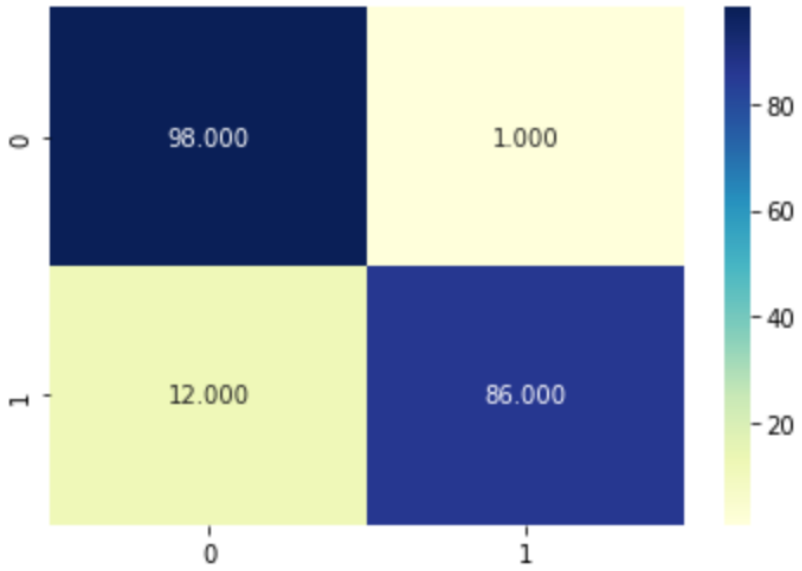
F1 Score : 0.9157894736842105



Random Forest :

	precision	recall	f1-score	support
0	0.89	0.99	0.94	99
1	0.99	0.88	0.93	98
accuracy			0.93	197
macro avg	0.94	0.93	0.93	197
weighted avg	0.94	0.93	0.93	197

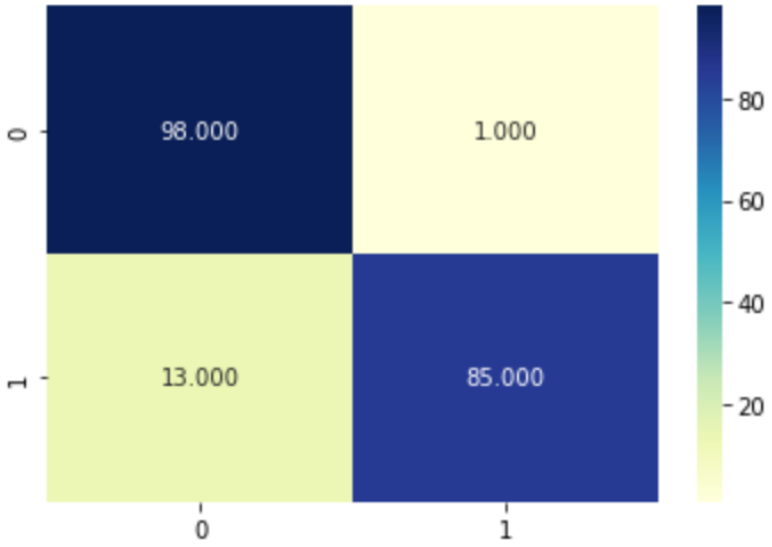
F1 Score : 0.9297297297297297



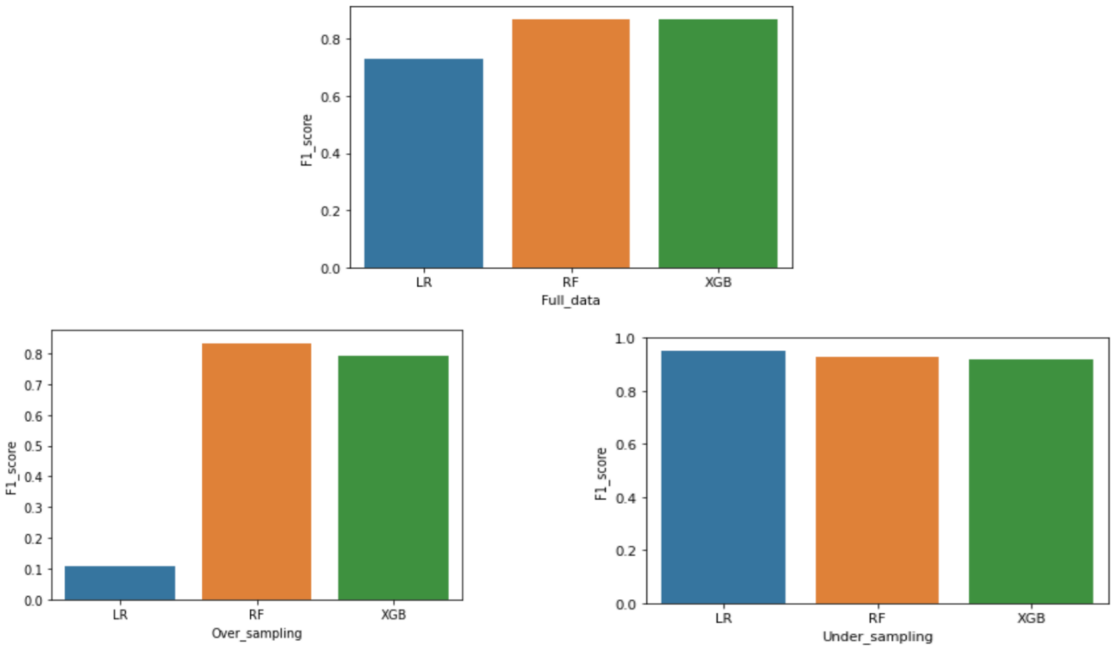
XGBclassifier :

	precision	recall	f1-score	support
0	0.88	0.99	0.93	99
1	0.99	0.87	0.92	98
accuracy			0.93	197
macro avg	0.94	0.93	0.93	197
weighted avg	0.94	0.93	0.93	197

F1 Score : 0.9239130434782609



Results :



as you can see from the scores and the graphs i got better results with the undersampling method with over .90 score but it still have to be better results because it's not a simple problem i need to have over .99 score so for that i will do more research and also with this problem the data is changing everyday, so it is difficult to reach this accuracy score.

Conclusion :

Credit card fraud is unquestionably a form of criminal deception. This article evaluated recent results in this field and outlined the most common types of fraud, as well as how to detect them. This paper also explains in detail how machine learning can be used to improve fraud detection outcomes, including the method, pseudocode, explanations, and experimentation results. Because the complete dataset is made up of only two days' worth of transaction records, it's only a small portion of the data that could be made public if this research were to be used commercially. Because the program is based on machine learning methods, it will only get more efficient over time as more data is sent into it. While we didn't achieve our target of 100 percent accuracy in fraud detection, we did create a system that can get very close to it given enough time and data. As with any undertaking of this nature, there is potential for improvement. Because of the nature of this project, more algorithms can be added to this model to improve it even further. The project gains a lot of modularity and versatility as a result of this. The dataset contains further room for development. As a result, more data will undoubtedly improve the model's accuracy in detecting frauds while lowering the amount of false positives. This, however, necessitates official backing from the banks themselves.