Projet2 - Credit fraud detection using Machine learning models

Hamza Boulaala, Fes Maroc,
E-mail : hamzaboulaalaop@gmail.com

December 2020

Contents

List of Figures

1 Introduction

Credit Card Fraud has been an issue that plagued the world, especially since online currency transactions became an essential part of our lives. The goal of this project is to build three models, using Machine Learning techniques, that can detect Credit Card Fraud and to choose the best one. These models can help predict fraud based on features that were extract from raw data using PCA (Principal component Analysis), in this case we have 28 features, provided by Finamaze.

2 Getting to Know the Data

2.1 Inspecting the Data

The data provided by Finamaze has 28 anonymous variables, Time, Amount and Class.
To do an exploratory data analysis, we need to know more about our Data.

```
In [239]: df=pd.read_csv('creditcard.csv',sep=',')

In [240]: df.head(10)
Out[240]:
   Time        V1        V2        V3  ...       V27       V28  Amount  Class
0   0.0 -1.359807 -0.072781  2.536347  ...  0.133558 -0.021053  149.62      0
1   0.0  1.191857  0.266151  0.166480  ... -0.008983  0.014724    2.69      0
2   1.0 -1.358354 -1.340163  1.773209  ... -0.055353 -0.059752  378.66      0
3   1.0 -0.966272 -0.185226  1.792993  ...  0.062723  0.061458  123.50      0
4   2.0 -1.158233  0.877737  1.548718  ...  0.219422  0.215153   69.99      0
5   2.0 -0.425966  0.960523  1.141109  ...  0.253844  0.081080    3.67      0
6   4.0  1.229658  0.141004  0.045371  ...  0.034507  0.005168    4.99      0
7   7.0 -0.644269  1.417964  1.074380  ... -1.206921 -1.085339   40.80      0
8   7.0 -0.894286  0.286157 -0.113192  ...  0.011747  0.142404   93.20      0
9   9.0 -0.338262  1.119593  1.044367  ...  0.246219  0.083076    3.68      0

[10 rows x 31 columns]
```

Figure 1: the first 10 lines of the Data-frame

Time : represents the time passed between the first transaction and the current one, by seconds.
Amount : represents the amount of money in each transaction.
Class : unlike the two previously mentioned variables, this one is a Categorical variable. It only takes two values :

- If it is a fraudulent transaction then : Class = 1.
- If it is not fraudulent then : Class = 0.

The other 28 variables (V1,. . . , V28) are all quantitative variables.
We can see from this output :

```
In [242]: 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
```

Figure 2: Detailed information about all the variables

- we have 31 variables.
- The type of all the variables is *float* except Class is an *integer*, it is a categorical variable with only two values.
- We have 284807 observations (rows).

2.2 Missing values

```
In [244]: df.isnull().values.any()
Out[244]: False

In [245]: df.Class.value_counts()
Out[245]:
0    284315
1       492
Name: Class, dtype: int64
```

Figure 3: Missing Data and value count

Checking for missing Data in the Data-set is essential because if there are any, then we have to either delete the observations that have them, or replace all of the Na with the mean of the variable. From this output, luckily we can conclude that there are no Na or NaN in the Data. Which is good, we won't have to make any change that may alter the data. Also, the value-counts function shows us something interesting. The number of fraudulent cases is very small (492) compared to non fraudulent cases (284315). To put in better perspective we can see the percentage in the output below :

The percentage of fraud cases in the data is approximately 0.1727%, whereas 99.87% of the data

```
In [248]:
     ...: fraud_percentage = (fraud_cases/len(df))*100
     ...: nofraud_percentage = (nofraud_cases/len(df))*100
     ...:
     ...: print('the percentage of fraud cases in the data is =', fraud_percentage)
     ...: print('the percentage of non fraud transactions in the data is = ', nofraud_percentage)
the percentage of fraud cases in the data is = 0.1727485630620034
the percentage of non fraud transactions in the data is =  99.82725143693798
```

Figure 4: percentage of each Class in the Data

are not. So there is a clear imbalance in this set. If we apply Machine learning algorithms directly with this obvious difference, it won't give us good prediction results. One of the remedies that we can use is sampling. Meaning we are gonna extract a random sample from the non fraud cases , in a proportional size to the fraud data. That way when training the models, it will train equally for both cases. This will be in another section of the report.

2.3 Data visualisation

Data visualisation is the best way to inspect the data and derive good insight.

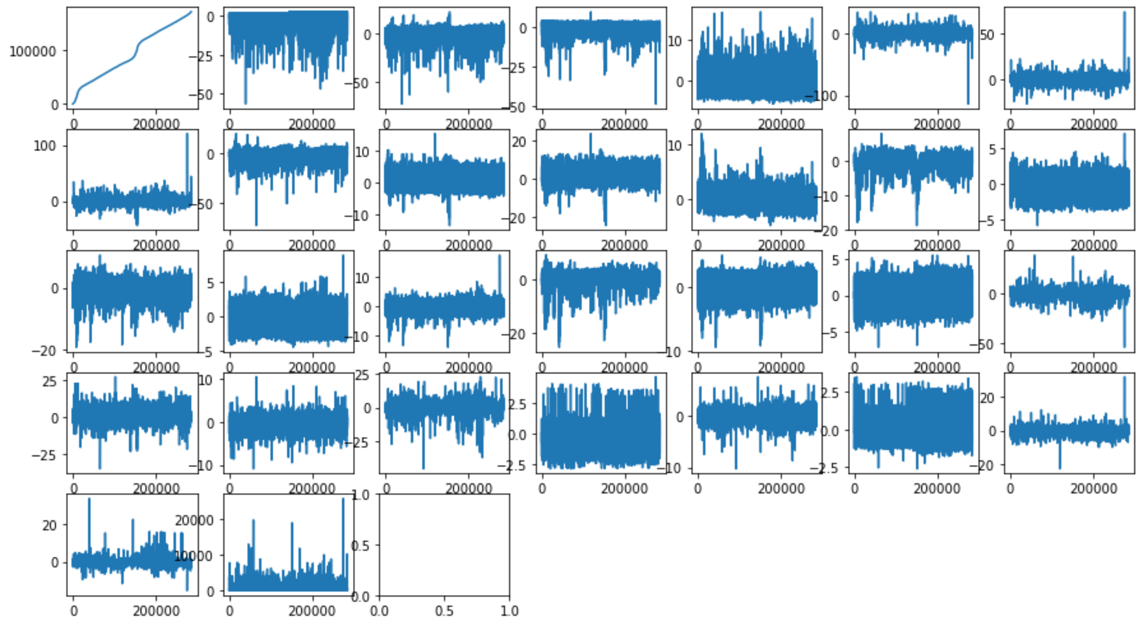

Figure 5: Plots of the variables

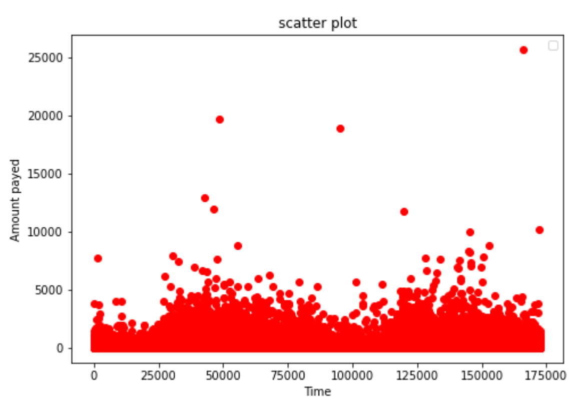

Figure 6: Amount payed sorted by Time

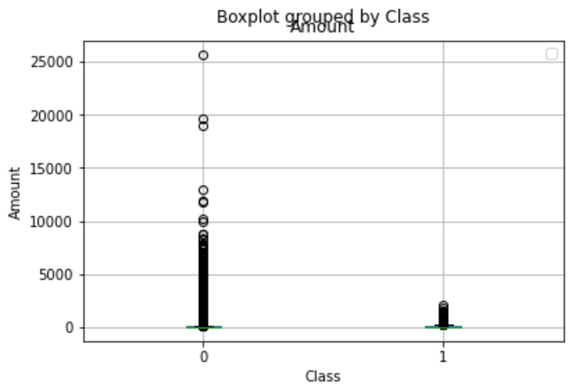

Figure 7: Boxplot of Class by Amount

2.4 Exploratory Statistics

To get a more in depth look at our data, we use exploratory analysis to get results such as the mean, standard deviation, min-max, and the quantiles(25%, 50%,75%). The information about the features V1,. . . , is not of the highest importance since they are unknown, unlike Time and Amount of payment.

```
In [3]: df_filtred=df.drop(df[["Class"]], axis="columns")

In [4]: df_filtred.describe()
Out[4]:
                Time            V1  ...           V28         Amount
count  284807.000000  2.848070e+05  ...  2.848070e+05  284807.000000
mean    94813.859575  3.919560e-15  ... -1.206049e-16      88.349619
std     47488.145955  1.958696e+00  ...  3.300833e-01     250.120109
min         0.000000 -5.640751e+01  ... -1.543008e+01       0.000000
25%     54201.500000 -9.203734e-01  ... -5.295979e-02       5.600000
50%     84692.000000  1.810880e-02  ...  1.124383e-02      22.000000
75%    139320.500000  1.315642e+00  ...  7.827995e-02      77.165000
max    172792.000000  2.454930e+00  ...  3.384781e+01   25691.160000

[8 rows x 30 columns]
```

Figure 8: Descriptive Statistics

2.5 Correlation

We can get an idea about the correlation between the different variables of our data using the correlation matrix.

```
In [20]: mat_corr
Out[20]:
            Time            V1            V2  ...           V28    Amount     Class
Time    1.000000  1.173963e-01 -1.059333e-02  ... -9.412688e-03 -0.010596 -0.012323
V1      0.117396  1.000000e+00  4.697350e-17  ...  9.820892e-16 -0.227709 -0.101347
V2     -0.010593  4.697350e-17  1.000000e+00  ... -3.676415e-16 -0.531409  0.091289
V3     -0.419618 -1.424390e-15  2.512175e-16  ...  7.726948e-16 -0.210880 -0.192961
V4     -0.105260  1.755316e-17 -1.126388e-16  ... -5.863664e-17  0.098732  0.133447
V5      0.173072  6.391162e-17 -2.039868e-16  ... -3.299167e-16 -0.386356 -0.094974
V6     -0.063016  2.398071e-16  5.024680e-16  ...  4.813155e-16  0.215981 -0.043643
V7      0.084714  1.991550e-15  3.966486e-16  ... -6.836764e-17  0.397311 -0.187257
V8     -0.036949 -9.490675e-17 -4.413984e-17  ... -4.484325e-16 -0.103079  0.019875
V9     -0.008660  2.169581e-16 -5.728718e-17  ...  9.146779e-16 -0.044246 -0.097733
V10     0.030617  7.433820e-17 -4.782388e-16  ... -1.515934e-16 -0.101502 -0.216883
V11    -0.247689  2.438580e-16  9.468995e-16  ... -3.091914e-16  0.000104  0.154876
V12     0.124348  2.422086e-16 -6.588252e-16  ...  7.327446e-16 -0.009542 -0.260593
V13    -0.065902 -2.115458e-16  3.854521e-16  ...  1.049541e-15  0.005293 -0.004570
V14    -0.098757  9.352582e-16 -2.541036e-16  ...  2.503271e-15  0.033751 -0.302544
V15    -0.183453 -3.252451e-16  2.831060e-16  ... -1.063286e-15 -0.002986 -0.004223
V16     0.011903  6.308789e-16  4.934097e-17  ...  8.637186e-16 -0.003910 -0.196539
V17    -0.073297 -5.011524e-16 -9.883008e-16  ... -2.182692e-16  0.007309 -0.326481
V18     0.090438  2.870125e-16  2.636654e-16  ...  8.844995e-16  0.035650 -0.111485
V19     0.028975  1.818128e-16  9.528280e-17  ... -1.375843e-15 -0.056151  0.034783
V20    -0.050866  1.036959e-16 -9.309954e-16  ... -1.133579e-16  0.339403  0.020090
V21     0.044736 -1.755072e-16  8.444409e-17  ...  5.132234e-16  0.105999  0.040413
V22     0.144059  7.477367e-17  2.500830e-16  ... -3.021376e-16 -0.064801  0.000805
V23     0.051142  9.808705e-16  1.059562e-16  ...  9.029821e-16 -0.112633 -0.002685
V24    -0.016182  7.354269e-17 -8.142354e-18  ... -2.259275e-16  0.005146 -0.007221
V25    -0.233083 -9.805358e-16 -4.261894e-17  ...  3.399375e-16 -0.047837  0.003308
V26    -0.041407 -8.621897e-17  2.601622e-16  ... -3.751403e-16 -0.003208  0.004455
V27    -0.005135  3.208233e-17 -4.478472e-16  ... -3.770124e-16  0.028825  0.017580
V28    -0.009413  9.820892e-16 -3.676415e-16  ...  1.000000e+00  0.010258  0.009536
Amount -0.010596 -2.277087e-01 -5.314089e-01  ...  1.025822e-02  1.000000  0.005632
Class  -0.012323 -1.013473e-01  9.128865e-02  ...  9.536041e-03  0.005632  1.000000

[31 rows x 31 columns]
```

Figure 9: Correlation matrix

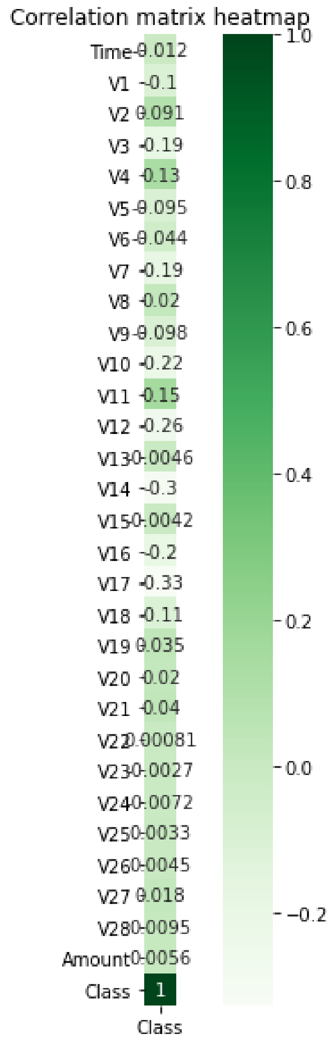

Figure 10: Correlation matrix using heatmap

This heat-map gives us a better visualisation of the correlation between our main variable Class and the other features.

3 Data Preparation

In this section we will work on preparing the data to deploy our machine learning models. As we noticed before, there is a clear imbalance between the number of cases of fraud (class=1) and the non fraudulent cases (Class=0). In order to fix this problem, I resorted to sampling the non fraudulent cases (since it has more cases) to match the fraudulent cases that way the model will be balanced and not biased to a certain result. In short, in this section i created a new data-frame

```
In [23]: fraud_cases = df.Class.value_counts()[1]
    ...: nofraud_cases = df.Class.value_counts()[0]
    ...: print('the number of fraud cases in the data is =', fraud_cases)
    ...: print('the number of non fraud transactions in the data is = ', nofraud_cases)
the number of fraud cases in the data is = 492
the number of non fraud transactions in the data is =  284315
```

Figure 11: Number of cases

```
#seperating the data
df_fraud = df.loc[df['Class'] != 0]
df_nofraud = df.loc[df['Class'] != 1]


#sampling new data to balance the ratio of fraud and no fraud data
New_nofraud = df_nofraud.sample(fraud_cases)
New_nofraud

#merging data to create new dataframe
New_df = pd.concat([df_fraud, New_nofraud])
New_df = New_df.sample(frac=1).reset_index(drop=True) #shuffling
Class_vect = New_df[['Class']]
# boxplot of amount by class
plt.figure(figsize=(7.5, 9))
New_df.boxplot( column = 'Amount', by = 'Class')
plt.xlabel('Class'); plt.ylabel('Amount')
plt.legend()
plt.show()
```

Figure 12: Code for sampling and merging the data

that contains approximately an equal number of cases for both classes.

To visualise the difference, I used the box-plot and new correlation matrix.

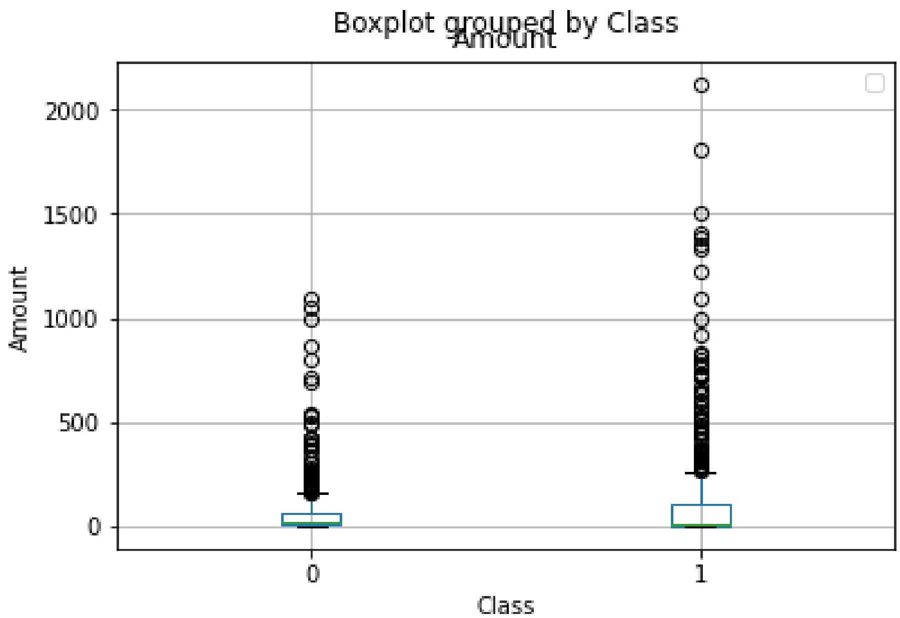

Figure 13: New data boxplot

We can notice by this graph that there are some points that can be considered as outliers but we're gonna keep them for now because they also represent cases of fraud where large amounts of money is taken, meaning our models have to be able to detect it. If it gives us poor accuracy for the model or interferes with the results then we can removed easily.
Also we can check the new correlation between each variable and the main **Class** variable.

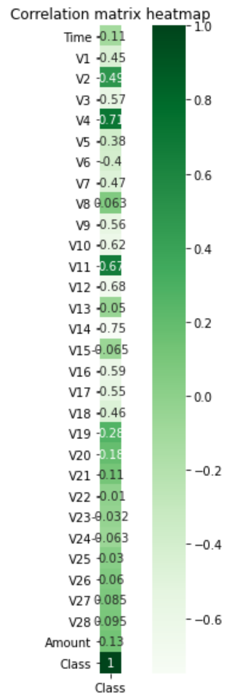

Figure 14: New data Correlation vector

```
In [35]:
    ...: corr1 = Class_corr[Class_corr.Class< -0.5]
    ...: corr2 = Class_corr[Class_corr.Class > 0.5]
    ...: corr1
    ...: corr2
Out[35]:
          Class
V4     0.714055
V11    0.670292
Class  1.000000

In [36]: corr1
Out[36]:
        Class
V3  -0.568132
V9  -0.562066
V10 -0.622054
V12 -0.678143
V14 -0.748858
V16 -0.592750
V17 -0.553028
```

Figure 15: Highest correlated variables

The correlation between most variables and the 'Class' variable has increased compared to the heat-map from before. that's interesting! We can check that in the figure below.

As it shows, the positively correlated variables with **Class** are V4 and V11. On the other hand, the negatively correlated variables are V3, V9, V10, V12, V14, V16, V17.

4 Deploying the machine learning models

4.1 Splitting the data

Before training our models, we have to split the data into training part and testing. Here I chose 80% for training and 20% for testing.

```
# Splitting data to train and to test
New_df = New_df.drop('Class', axis =1)
X = New_df
Y = Class_vect
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=1)
```

Figure 16: Splitting the data

4.2 First method : Confusion Matrix

In this first method, I used the *predict* function after training the model, and used the confusion matrix to showcase the prediction results. We're gonna use this on the three models.

4.2.1 Logistic regression model

Since the variable we are working on is categorical, then it is only reasonable to start with Logistic regression.
Input :

```
#Logistic regression
model_LR = LogisticRegression()
model_LR.fit(X_train, Y_train.values.ravel())

pred_LR= model_LR.predict(X_test)

#visualising the predicted data vs the real data
sns.heatmap(confusion_matrix(Y_test, pred_LR), annot=True, cbar=None, cmap="Blues", fmt = 'g')
plt.title("confusion matrix : results evaluation")
plt.tight_layout()
plt.ylabel('Real data')
plt.xlabel('model predicted result')
plt.show()
```

Figure 17: Logistic regression code

Output :

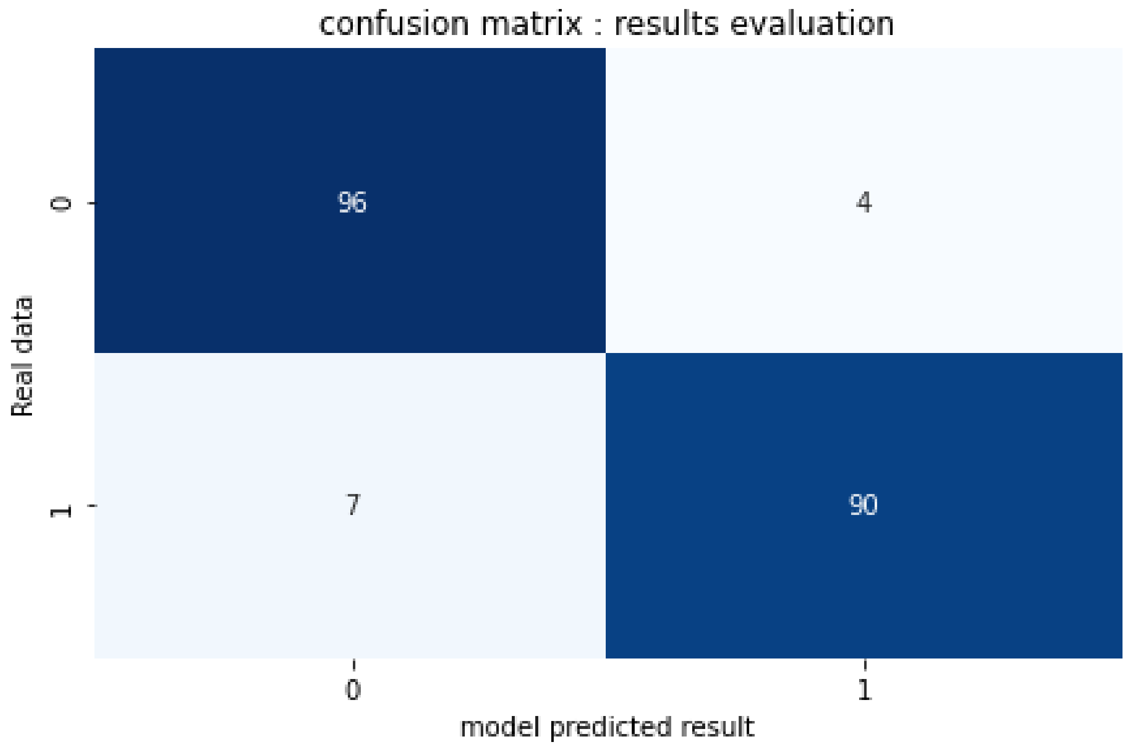

Figure 18: Logistic regression confusion matrix

As we can see here, the results show us that :

- 96 of the cases that were predicted non fraudulent and 90 of the predicted fraudulent cases were right.
- However, there were some false positive and false negative results. 7 of the predicted non fraud cases were actually fraud, and 4 of the predicted fraud were not.

4.2.2 K-nearest-neighbour

Same as we did before, train the model, predict using test data, illustrate the result using confusion matrix.
Input :
Output :
The confusion matrix show us that :

- 70 of the cases that were predicted non fraudulent and 76 of the predicted fraudulent cases were right.

```python
# K-Nearest Neighbor
model_KNN = KNeighborsClassifier()

model_KNN.fit(X_train , Y_train.values.ravel())
pred_KNN = model_KNN.predict(X_test)
#Result visualisation KNN model :
sns.heatmap(confusion_matrix(Y_test, pred_KNN), annot=True, cbar=None, cmap="Greens", fmt = 'g')
plt.title("confusion matrix : results evaluation")
plt.tight_layout()
plt.ylabel('Real data')
plt.xlabel('model predicted result')
plt.show()
```

Figure 19: KNN code

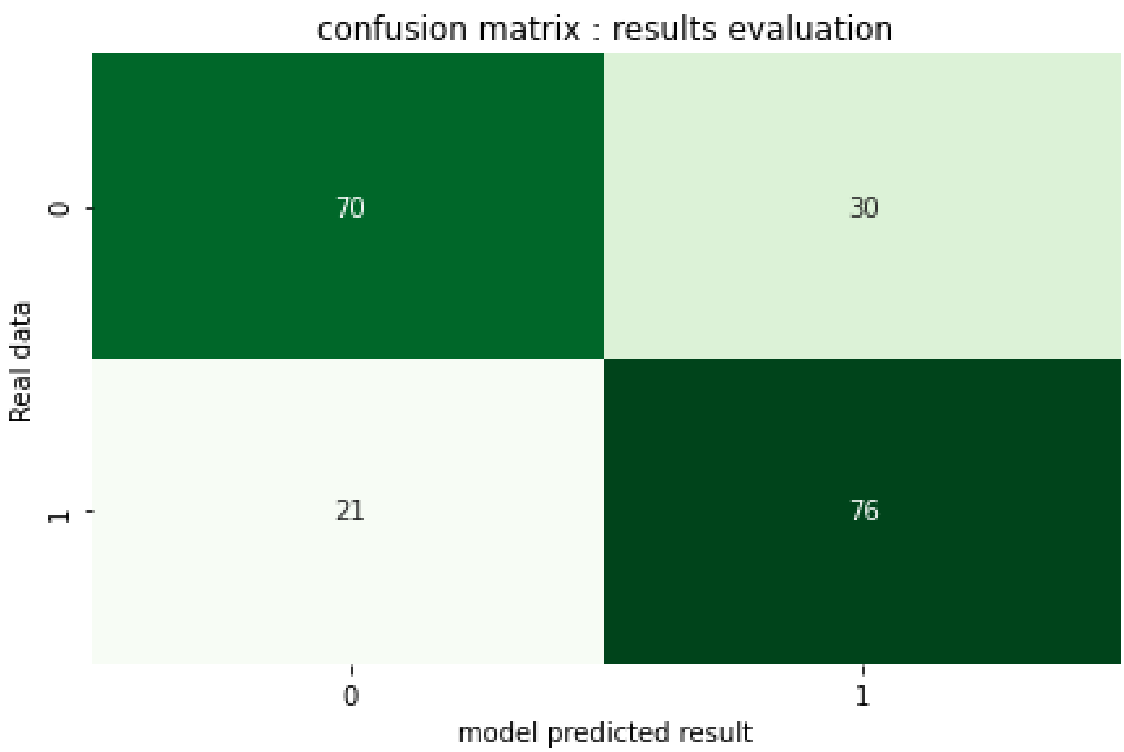

Figure 20: KNN confusion matrix

- However, there were some false positive and false negative results. 21 of the predicted non fraud cases were actually fraud, and 30 of the predicted fraud were not.

We can already tell this model is inaccurate compared to the Logistic regression one.

4.2.3 Random Forest

We repeat the same test structure as before. **Input :**

```
#Random forest Classifier :
model_RF = RandomForestClassifier()

model_RF.fit(X_train, Y_train.values.ravel())
pred_RF = model_RF.predict(X_test)

#Result visualisation Random forest model :
sns.heatmap(confusion_matrix(Y_test, pred_RF), annot=True, cbar=None, cmap="Greys", fmt = 'g')
plt.title("confusion matrix : results evaluation")
plt.tight_layout()
plt.ylabel('Real data')
plt.xlabel('model predicted result')
plt.show()
```

Figure 21: Random Forest Code

Output : As we can see here, the confusion matrix show us that :

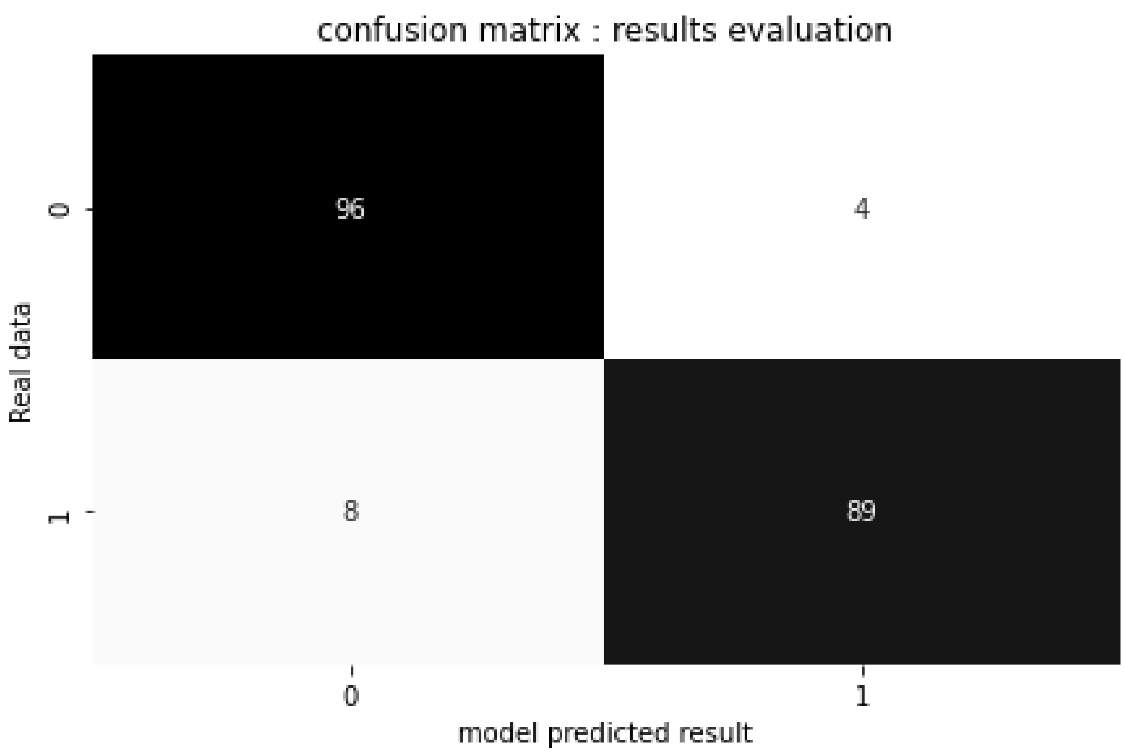

Figure 22: Random Forest Confusion matrix

- 96 of the cases that were predicted non fraudulent and 89 of the predicted fraudulent cases were right.
- However, there were some false positive and false negative results. 8 of the predicted non fraud cases were actually fraud, and 4 of the predicted fraud were not.

This result is precise and resembles the one we found for the Logistic regression model.

4.3 Second method : Cross validation

Using the Cross validation Score function from **sklearn** library, we can choose how many accuracy scores we get for each model. I chose 5 which is the default number to give us more perception, and the calculate the mean of those scores for each model.
Input :

```
#Cross validation scores #generates 5 cross validation scores by default
cross_val_LR= cross_val_score(model_LR, X_train, Y_train)
cross_val_KNN = cross_val_score(model_KNN, X_train, Y_train)
cross_val_RF= cross_val_score(model_RF, X_train, Y_train)

Result = [cross_val_LR.mean(), cross_val_KNN.mean(), cross_val_RF.mean()]
```

Figure 23: Code for cross validation

Output :

```
In [51]: Result
Out[51]: [0.9186325888897848, 0.6327662662259131, 0.9415302749334838]
```

Figure 24: Cross validation mean Scores

We can see that :

- The Logistic regression model has an accuracy of 91.18%
- The KNN model has an accuracy of 62.27%
- The Random Forest model has an accuracy of 94.15%

Random forest is clearly more accurate than the other two models.

```
metrics.accuracy_score(Y_test, pred_LR) #Logistic Regression accuracy score
metrics.accuracy_score(Y_test, pred_KNN) #K-Nearest-Neighbor accuracy score
metrics.accuracy_score(Y_test, pred_RF) #Random Forest accuracy score
```

Figure 25: Accuracy metrics Code

4.4 Third method : Accuracy metrics

We also test the accuracy of our model using the test data and predicted data for all three models.
Input :
Output : Result

```
In [53]: metrics.accuracy_score(Y_test, pred_LR) #Logistic Regression accuracy score
Out[53]: 0.9441624365482234

In [54]: metrics.accuracy_score(Y_test, pred_KNN) #K-Nearest-Neighbor accuracy score
Out[54]: 0.7411167512690355

In [55]: metrics.accuracy_score(Y_test, pred_RF) #Random Forest accuracy score
Out[55]: 0.9390862944162437
```

Figure 26: Accuracy metrics results

- The Logistic regression model has an accuracy of 94.41%
- The KNN model has an accuracy of 74.11%
- The Random Forest model has an accuracy of 93.90%

The Logistic regression and the Random forest models both have a high accuracy and approximate from each other.

5 Conclusion

We can conclude from the three methods we used to determine the best Machine Learning model out of Logistic regression, K-nearest-neighbor and Random Forest, is the Random forest model. Because it had the best result in confusion matrix in terms of prediction, the best accuracy score for Cross validation mean score and for the metrics accuracy score, even if it was close to the logistic regression model. All in all, the best model out of the three is the Random forest model.

6 Bibliography

- Hands on machine learning : Book.
- Libraries documentation : Pandas, matplotlib, seasborn, Sklearn (selection, metrics,...).
- Fraud detection Kaggle notebooks.