THE PRIVATE HIGHER SCHOOL OF ENGINEERING AND TECHNOLOGY

Data Science
Insurance Project

Department of Computer Science
The Private Higher School of Engineering &Technology

Elaborated by:
Gnichi Meriam
Haguiga Khaled
Haguiga Walid
Jaber Khalil
Lahbib Fedi
Raddaoui Ramzi

Supervised by:
Ncib Lotfi
Mlaouh Ayoub

May, 2021

Contents

1 Ceneral Introduction **3**

2 Project Content: **4**
2.1 Business Objectives: . . . 4
2.2 Data Science Objectives: . . . 4

3 Data Acquisition & Understanding **5**
3.1 Internal Data . . . 5
3.2 External data . . . 14
3.2.1 Cars Dataset: . . . 14
3.2.2 Planned Sinister: . . . 15

4 DATA PREPARATION **16**
4.1 Multi-insurance Fraud detection . . . 16
4.1.1 Clustering . . . 17
4.1.2 Classification . . . 18
4.2 Sinister Fraud Detection . . . 18
4.3 Bonus malus class prediction . . . 19
4.3.1 Police data set . . . 19
4.3.2 Vehicule data set . . . 19
4.3.3 Merge data . . . 20
4.4 Wrecked car detector . . . 21

5 Data Visualisation **23**

6 Data Modeling **26**
6.1 Multi-insurance Fraud detection . . . 26
6.1.1 BoostingClassifier . . . 26
6.1.2 Gradient Boosting Classifier . . . 27
6.1.3 Neural network classification . . . 27
6.2 Bonus Malus class prediction . . . 28
6.3 Sinister Fraud Detection . . . 30
6.4 Wrecked Car Detector . . . 32

7 Deployment **37**
7.1 Introduction . . . 37
7.2 Tools &Technology . . . 37
7.2.1 Microsoft Power BI . . . 37
7.2.1.1 Introduction . . . 37
7.2.1.2 Screenshots . . . 38
7.2.2 Django . . . 39
7.2.2.1 Introduction . . . 39
7.2.2.2 Screenshots . . . 39

8 Conclusion and perspectives **43**

List of Figures

3.1 Classe Bonus-Malus dataset Overview . . . 5
3.2 Classe Bonus-Malus dataset Variables . . . 5
3.3 Pearson's Correlations . . . 6
3.4 Missing values . . . 6
3.5 Bonus-Malus dataset sample . . . 7
3.6 Police dataset Overview . . . 7
3.7 Police dataset Variables . . . 7
3.8 Pearson's Correlations . . . 8
3.9 Missing values in Police data-set . . . 8
3.10 Police data-set sample . . . 8
3.11 Vehicule dataset Overview . . . 9
3.12 Vehicule dataset Variables . . . 9
3.13 Missing values in Vehicule data-set . . . 9
3.14 Vehicule data-set sample . . . 10
3.15 Sinistre dataset Overview . . . 10
3.16 Sinistre dataset Variables . . . 10
3.17 Pearson's Correlations . . . 11
3.18 Missing values . . . 11
3.19 Sinistre dataset sample . . . 11
3.20 Usage Vehicule data-set sample . . . 12
3.21 Marque Vehicule data-set sample . . . 12
3.22 Souscripteur data-set sample . . . 13
3.23 Cars Dataset . . . 14
3.24 Sample of wrecked cars . . . 14
3.25 Sample of repairable cars . . . 14
3.26 Planned Sinister Variables . . . 15
3.27 Generation non-Fraud Sinisters . . . 15
3.28 Generation Fraud Sinisters . . . 15

4.1 BonusMalus dates Encoding . . . 16
4.2 Bonus-Malus variables OrdinalEncoding . . . 16
4.3 Bonus-Malus variables OneHotEncoding . . . 17
4.4 Bonus-Malus dataFrame scaling with StandardScaler . . . 17
4.5 Fraud Detection Using Data Analytics . . . 17
4.6 **Over sampling** . . . 18
4.7 **Undersampling** . . . 18
4.8 **SMOTE** . . . 18
4.9 **Clustering** . . . 18
4.10 Sinister variables OneHotEncoding . . . 18
4.11 Sinister dates Encoding . . . 19
4.12 Separating days from months in dateEcheancePolice and adding it to the dataFrame . . . 19
4.13 Dropping unnecessary Police variables . . . 19
4.14 Replacing missing values in Column "energie" . . . 19
4.15 Merging tables Police and Vehicule . . . 20
4.16 Encoding time features . . . 20
4.17 OneHotEncoder for "typeImmatriculation" and "energie" . . . 21
4.18 ImageDataGenerator instance . . . 22
4.19 Data augmentation result . . . 22

5.1 Pie charts of sinisters By Energie ,Fiscal power and Vehicle brand . . . 23

5.2 Sinisters By Day and Year . . . 23
5.3 Line chart for difference between the Occurence and the Opening of the sinister . . . 24
5.4 Pie charts of Policies by companies , month and year . . . 24
5.5 Clustered column chart of bonus malus number by UsageVehicle and classBonusMalus . . . 25

6.1 **Over sampling** . . . 26
6.2 Confusion matrix for Undersampling . . . 26
6.3 *SMOTE* . . . 27
6.4 Confusion matrix for ClusterCentroids . . . 27
6.5 Confusion matrix for Gradient Boosting Classifier model . . . 27
6.6 **over sampling** . . . 28
6.7 Confusion matrix for Undersampling model . . . 28
6.8 **Confusion matrix for SMOTE model** . . . 28
6.9 Confusion matrix for ClusterCentroids model . . . 28
6.10 implementing of Decision tree algorithm . . . 28
6.11 Decision tree Classifier . . . 29
6.12 mean of all scores . . . 29
6.13 implementing of bagging decision tree algorithm . . . 29
6.14 implementing of GradientBoosting algorithm . . . 30
6.15 the performance of the used algorithms . . . 30
6.16 implementing AdaBoost algorithm . . . 30
6.17 Confusion matrix AdaBoost . . . 31
6.18 implementing Support Vector Classification SVC algorithm . . . 31
6.19 Confusion matrix SVC . . . 32
6.20 Tuning hyper parameters with GridSearchCV . . . 32
6.21 Final Accuracy . . . 32
6.22 Building wrecked car detector model . . . 33
6.23 Wrecked car detector model summary . . . 33
6.24 Wrecked car detector model compiling . . . 34
6.25 Callback instance . . . 34
6.26 Fitting the model . . . 34
6.27 Loss and accuracy plots . . . 35
6.28 Visualization helpful functions . . . 35
6.29 Wrecked car . . . 36
6.30 Slightly damaged car . . . 36
6.31 Brand new car . . . 36
6.32 Exporting the model . . . 36

7.1 Power BI logo . . . 37
7.2 Dashboard overview . . . 38
7.3 Visuals Configurations . . . 38
7.4 Django logo . . . 39
7.5 Wrecked car Detector interface . . . 39
7.6 Bonus malus calculator interface . . . 40
7.7 Multi-insurance fraud interface . . . 40
7.8 Claims fraud interface . . . 41
7.9 Dashboard interface . . . 41
7.10 Dashboard interface . . . 42

Chapter 1

Ceneral Introduction

Data has always played a central role in the insurance industry, and today, insurance carriers have access to more of it than ever before. We have created more data in the past two years than the human race has ever created. Insurers like organisations in most industries—are overwhelmed by the explosion in data from a host of sources, including telematics, online and social media activity, voice analytics, connected sensors and wearable devices. They need machines to process this information and unearth analytical insights. But most insurers are struggling to maximise the benefits of machine learning.

This situation is seeing a gradual but steady change, driven by an environment characterised by increased competition, elastic marketplaces, complex claims and fraud behaviour, higher customer expectations and tighter regulation. Insurers are being forced to explore ways to use predictive modelling and machine learning to maintain their competitive edge, boost business operations and enhance customer satisfaction.

They are also examining how they can take advantage of recent advances in artificial intelligence (AI) and machine learning to solve business challenges across the insurance value chain. These include underwriting and loss prevention, product pricing, claims handling, fraud detection, sales and customer experience.

In this context, we were asked by the CGA[1]
to use the data source provided in order to help it to ameliorate her system .

[1]General Committee of Insurances Compagnies "CGA"

Chapter 2

Project Content:

2.1 Business Objectives:

The main objective of this project is to :

-Centralizing all information from all insurance companies in a data warehouse such as insurance policy, policyholders, etc. The objective is thus to allow researchers to have easier access to this data as part of their research and development work.

-Bonus-Malus class auto updating: ased on the customer's sinister history, we will determine his bonus-malus class using multi-class classification.

-Implementing admin dashboard:

2.2 Data Science Objectives:

The business and the technical objectives of this projects go hand on hand as the former cannot be accomplished without the latter. Accordingly, we will be using the following different techniques:

-Featuring engineering: This is the first step. It consists of cleaning the data and maneuvering it in order to giving it synthesized meaning and purpose. This is a primordial measure for the following steps.

-Applying big data procedures:

* Web scrapping: Collecting information from the web regarding our data using.

* Data storing: Using sql sever integration services to store our data (internal and external)

- Machine Learning: Applying different machine learning models (supervised and unsupervised) to classify our data and extracting distinguished profiles and behaviors. This step will help us not only score the clients, the vehicles and/or the agency but also detect fraud and contract termination cases (if they exist).

Chapter 3

Data Acquisition & Understanding

3.1 Internal Data

1. Bonus-Malus class: Represents the Bonus-Malus for each client which is an indicator calculated by insurance companies depending on the number of cases of claims they had.

Overview | Warnings 9 | Reproduction

Dataset statistics

Number of variables	11
Number of observations	1307361
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	109.7 MiB
Average record size in memory	88.0 B

Variable types

Numeric	9
Categorical	2

Figure 3.1: Classe Bonus-Malus dataset Overview

```
print(classe.columns)
```

```
Index(['CodeBonusMalus_id', 'CodePolice_id', 'CodeVehicule_id',
       'CodeAssure_id', 'CodeSouscripteur_id', 'codeUsage', 'ETAT_VEHICULE',
       'DATE_AFFECTATION', 'classeBonusMalus', 'classeBonusMalusCompagnie',
       'coefBonusMalus'],
      dtype='object')
```

Figure 3.2: Classe Bonus-Malus dataset Variables

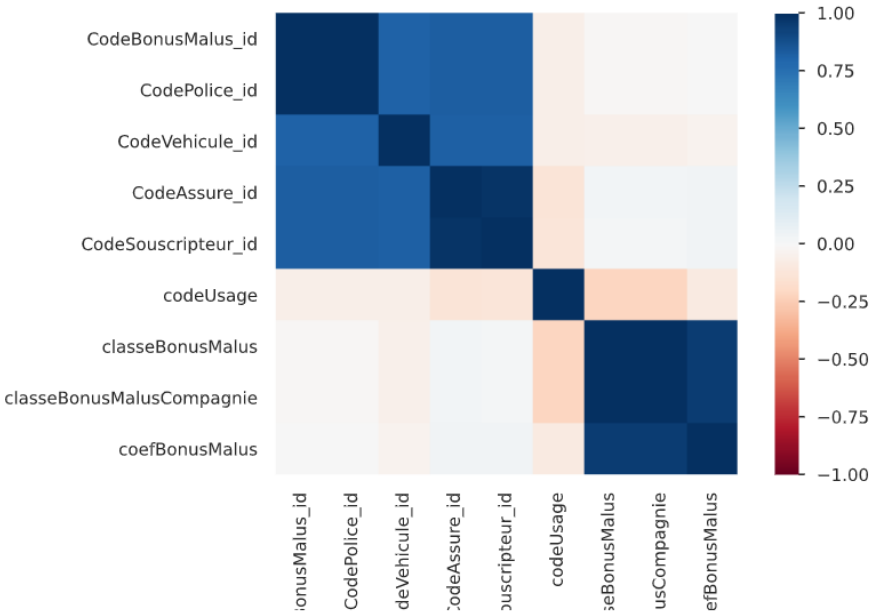


Figure 3.3: Pearson's Correlations

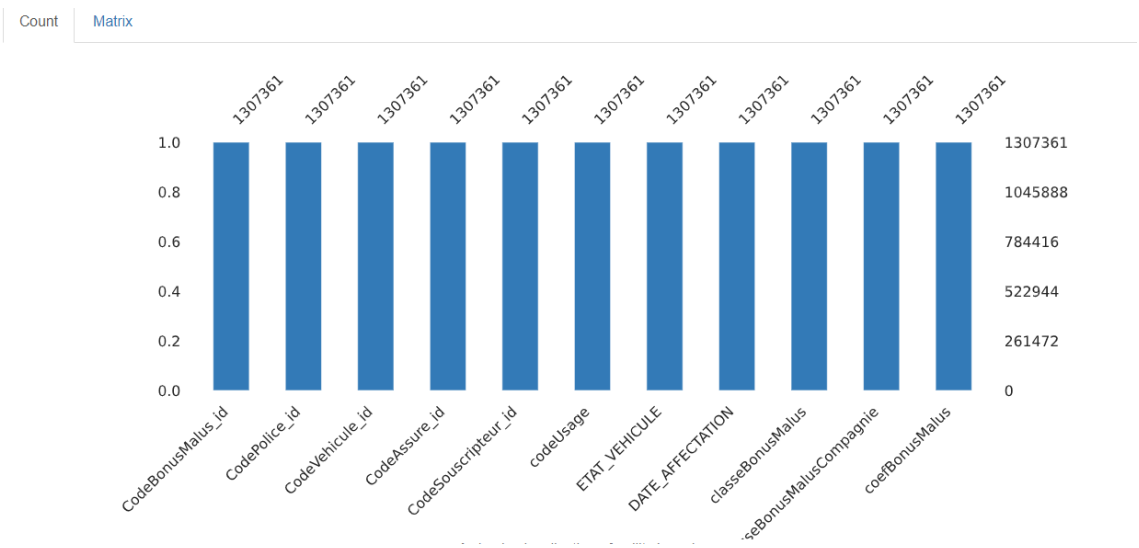


Figure 3.4: Missing values

	CodeBonusMalus_id	CodePolice_id	CodeVehicule_id	CodeAssure_id	CodeSouscripteur_id	codeUsage	ETAT_VEHICULE	DATE_AFFECTATION
0	117346	116167	111858	101772	101691	1	V	2017-07-14 00:00:00.000000
1	117347	116168	111859	99417	99336	10	V	2017-07-14 00:00:00.000000
2	117348	116169	111860	101773	101692	1	V	2017-07-14 00:00:00.000000
3	117349	116170	111861	101774	101693	1	V	2017-07-14 00:00:00.000000
4	117350	116171	111862	101775	101694	1	V	2017-07-14 00:00:00.000000
5	117351	116172	111863	101776	101695	1	V	2017-07-14 00:00:00.000000

classeBonusMalus	classeBonusMalusCompagnie	coefBonusMalus
1	1	70
1	1	80
1	1	70
1	1	70
2	2	80
1	1	70
3	3	90
1	1	70
2	2	80
2	2	80

Figure 3.5: Bonus-Malus dataset sample

2. Police : It contains the contracts between the clients and the insurance agence

Overview | Warnings 7 | Reproduction

Dataset statistics

Number of variables	12
Number of observations	1316071
Missing cells	141672
Missing cells (%)	0.9%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	120.5 MiB
Average record size in memory	96.0 B

Variable types

Numeric	3
Unsupported	1
Categorical	8

Figure 3.6: Police dataset Overview

```
police.columns
```

```
Index(['CodePolice_id', 'codeCompagnie_id', 'codeAgence', 'typeIntermediaire',
       'numPolice', 'naturePolice', 'typePolice', 'Etat_Police',
       'dateEffetPolice', 'DATE_RESILIATION', 'Date_Suspenssion',
       'dateEcheancePolice'],
      dtype='object')
```

Figure 3.7: Police dataset Variables

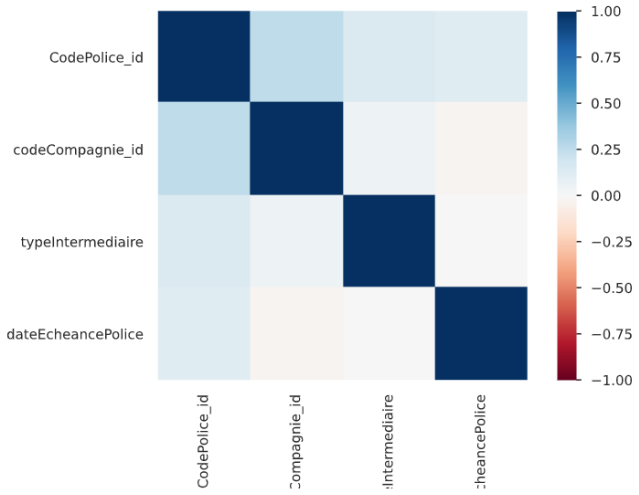


Figure 3.8: Pearson's Correlations

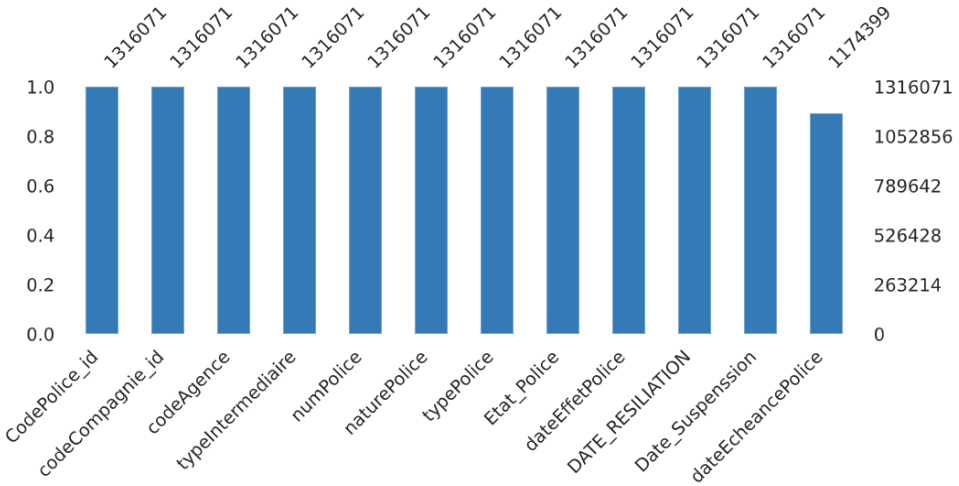


Figure 3.9: Missing values in Police data-set

	CodePolice_id	codeCompagnie_id	codeAgence	typeIntermediaire	numPolice	naturePolice	typePolice	Etat_Police	dateEffetPolice
0	269351	1	663	1	20095.0017122/9	R	I	V	2009-03-13
1	269352	1	689	1	20095.0025227/6	R	I	V	2009-03-13
2	269353	1	690	1	20095.0026073/3	R	I	V	2009-03-13
3	269354	1	724	1	20095.0045108/4	R	I	V	2009-03-13

DATE_RESILIATION	Date_Suspenssion	dateEcheancePolice
1900-01-01	1900-01-01	313.0
1900-01-01	1900-01-01	313.0
1900-01-01	1900-01-01	313.0
1900-01-01	1900-01-01	313.0
1900-01-01	1900-01-01	313.0
1900-01-01	1900-01-01	313.0
1900-01-01	1900-01-01	313.0
1900-01-01	1900-01-01	313.0
1900-01-01	1900-01-01	313.0
1900-01-01	1900-01-01	313.0

Figure 3.10: Police data-set sample

3. Vehicule

Overview | Warnings 24 | Reproduction

Dataset statistics

Number of variables	15
Number of observations	1116160
Missing cells	9909312
Missing cells (%)	59.2%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	127.7 MiB
Average record size in memory	120.0 B

Variable types

Numeric	3
Categorical	9
Unsupported	3

Figure 3.11: Vehicule dataset Overview

```
Vehicule.columns
```

```
Index(['CodeVehicule_id', 'puissanceFiscal', 'typeImmatriculation', 'energie',
       'dateInsertion', 'codeMarque', 'numChassis', 'numImmatriculation',
       'dateDerniereVisite', 'dateAjout', 'etatVehicule', 'dateMiseEpave',
       'DATE_RETRAIT', 'dateMiseCirculation', 'dateMiseAjourVehicule'],
      dtype='object')
```

Figure 3.12: Vehicule dataset Variables

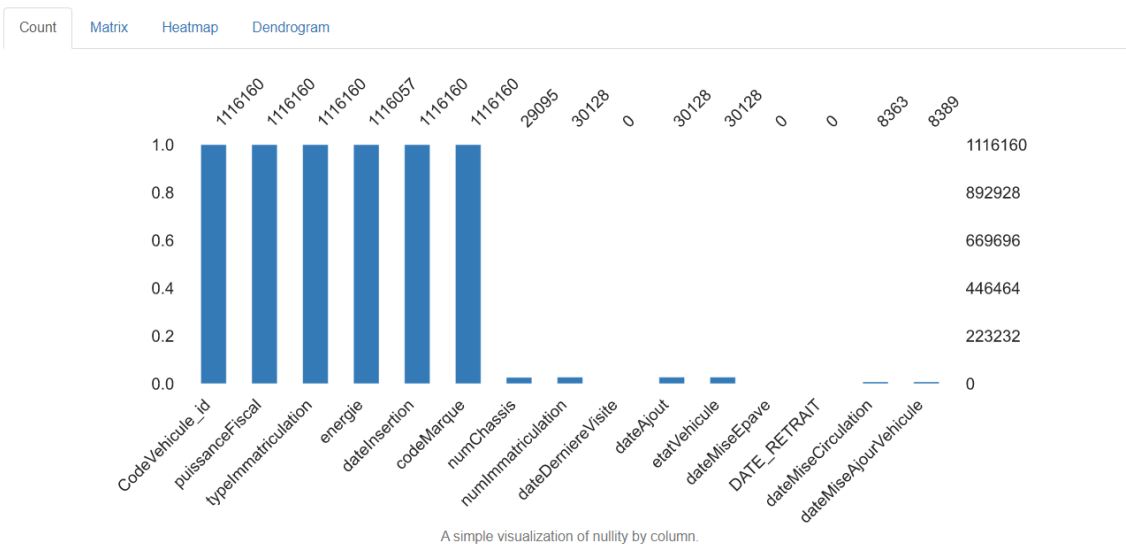


Figure 3.13: Missing values in Vehicule data-set

First rows

	CodeVehicule_id	puissanceFiscal	typeImmatriculation	energie	dateInsertion	codeMarque	numChassis	numImmatriculation	dateDerniereVisit
0	696264	5	TU	ESSENCE	2017-10-05	152	NaN	NaN	NaN
1	696597	6	TU	GASOIL	2017-10-05	808	NaN	NaN	NaN
2	696930	6	TU	ESSENCE	2017-10-05	29	NaN	NaN	NaN
3	697263	7	TU	GASOIL	2017-10-05	545	NaN	NaN	NaN
4	697597	8	TU	GASOIL	2017-10-05	376	NaN	NaN	NaN

dateAjout	etatVehicule	dateMiseEpave	DATE_RETRAIT	dateMiseCirculation	dateMiseAjourVehicule
NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN

Figure 3.14: Vehicule data-set sample

4. Sinistre : This dataset contains different sinister claims. The following figures shows that this dataset contains 832366 row representing a sinister which includes information about involved sides of the sinister and about it's treatment procedure. As shown in the figures 3.15 and 3.18, 3.9% of our data are duplicated while 17 % of the rows are missing from some variables.

Overview | Warnings 32 | Reproduction

Dataset statistics

Number of variables	17
Number of observations	832366
Missing cells	2502323
Missing cells (%)	17.7%
Duplicate rows	32291
Duplicate rows (%)	3.9%
Total size in memory	102.4 MiB
Average record size in memory	129.0 B

Variable types

Numeric	6
Categorical	9
Boolean	1
Unsupported	1

Figure 3.15: Sinistre dataset Overview

```
Sinistre.columns
```

```
Index(['CodeSinistre_id', 'CodePolice_id', 'CodeVehicule_id', 'CodeAssure_id',
       'dateOuvertureDuSinistre', 'lieuDuSinistre',
       'pourcentadeDeResponsabilite', 'natureDuSinistre', 'Calculer_Sinistre',
       'numeroDuSinistre', 'dateDeSurvenanceDuSinistre', 'identificationTiers',
       'numeroDePoliceCompagnieAdverse', 'codeCompagnieAdverse_id',
       'pourcentageCompagnieAdverse', 'numeroImmatriculationVehiculeAdverse',
       'mouvementDusinistre'],
      dtype='object')
```

Figure 3.16: Sinistre dataset Variables

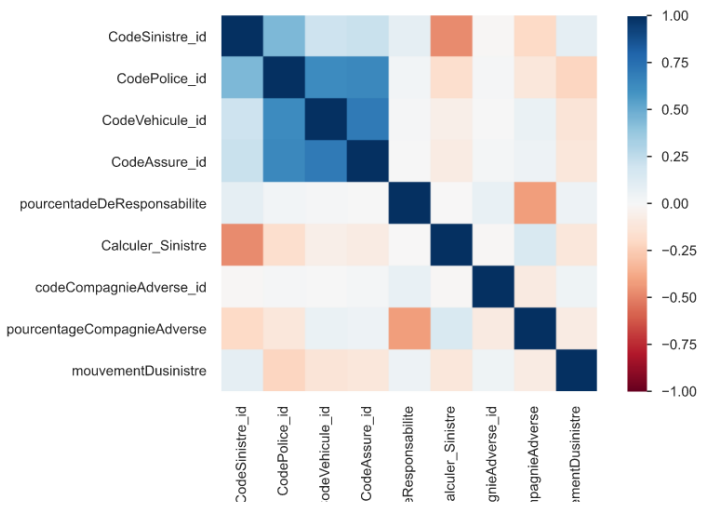


Figure 3.17: Pearson's Correlations

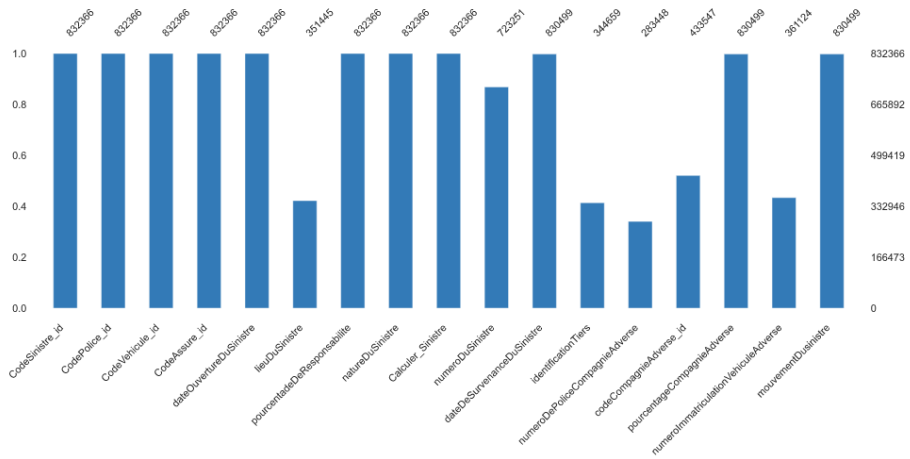


Figure 3.18: Missing values

	CodeSinistre_id	CodePolice_id	CodeVehicule_id	CodeAssure_id	dateOuvertureDuSinistre	lieuDuSinistre	pourcentadeDeResponsabilite	natureDuSin
0	58043	404756	348705	336250	2017-11-21	Tunis	50	M
1	58044	1420995	161572	1063012	2017-11-21	Tunis	100	M
2	58045	1397827	1082113	113223	2017-11-21	Monastir	100	M
3	58046	381019	327914	318249	2017-11-21	Monastir	100	M

Calculer_Sinistre	numeroDuSinistre	dateDeSurvenanceDuSinistre	identificationTiers	numeroDePoliceCompagnieAdverse	codeCompagnieAdverse_id
True	20170101765U	2017-11-19	U	NaN	6.0
True	20170101778H	2017-11-19	NaN	NaN	NaN
True	20170126546E	2017-11-19	NaN	NaN	NaN
True	20170101815Y	2017-11-19	NaN	NaN	NaN

pourcentageCompagnieAdverse	numeroImmatriculationVehiculeAdverse	mouvementDusinistre
0.0	1101TU144	3.0
0.0	NaN	3.0
0.0	NaN	3.0
0.0	NaN	3.0

Figure 3.19: Sinistre dataset sample

5. Usage Vehicule :Represents the usage of vehicles (for example: taxis, public transport etc)

	Code	Marque
0	2888	A SILVA MATOS
1	581	A.C.L.
2	873	A.K
3	886	ACC-ADOC
4	1314	ACCESS
...	...	...
1765	819	ZLM
1766	778	ZMAJ
1767	1751	ZNEN
1768	1782	ZONDA
1769	1764	ZOOMLION

1770 rows × 2 columns

Figure 3.20: Usage Vehicule data-set sample

6. Marque Vehicule :It contains the brands of the clients' vehicles

	Code	Marque
0	2888	A SILVA MATOS
1	581	A.C.L.
2	873	A.K
3	886	ACC-ADOC
4	1314	ACCESS
...	...	...
1765	819	ZLM
1766	778	ZMAJ
1767	1751	ZNEN
1768	1782	ZONDA
1769	1764	ZOOMLION

1770 rows × 2 columns

Figure 3.21: Marque Vehicule data-set sample

7. Souscripteur :

	CodeSouscripteur_id
0	134601
1	134602
2	134603
3	134604
4	134605
...	...
1037440	945939
1037441	945940
1037442	945941
1037443	945942
1037444	945943

1037445 rows × 1 columns

Figure 3.22: Souscripteur data-set sample

3.2 External data

3.2.1 Cars Dataset:

A wrecked car (or not repairable car) is any car whose body is severely damaged that even after repair, it has some balance issues that the car becomes incapable of safe operation for use on roads or highways and which has no resale value, except as a source for parts or scrap.So a potential fraud can be done by declaring the client's car as completely wrecked to exaggerate the refund amount paid by the insurance companies then sell the vehicle as parts. The current process adopted by the CGA (The General Insurance Committee) is to send a repairing specialist to verify the status of the vehicle whether it's repairable or not. This process may take weeks to months depending on many factors. Therefore, this delay can cause the clients to become unhappy.Our goal is to give a reliable and instant solution to detect wrecked cars with at least 95% accuracy.
Using the assistance of a vehicle repairing specialist, we collected a balanced dataset manually and divided it into a train, validate and test sets.

```
E:.
├───test
├───train
│   ├───epave
│   └───not_epave
└───validate
    ├───epave
    └───not_epave
```

Figure 3.23: Cars Dataset

Here we can see a sample of wrecked cars and repairable cars from our dataset.

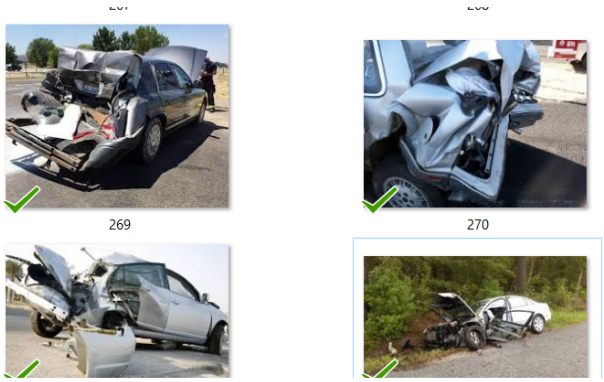


Figure 3.24: Sample of wrecked cars

Figure 3.25: Sample of repairable cars

3.2.2 Planned Sinister:

Due to the large number of our dataset and the impossibility to detect planned sinisters that includes a big group of insured, we ended up generating a labeled and balanced dataset while covering all possible cases of network sinisters. This dataset is composed of 99994 rows for training and 19998 rows for testing

```
cols = ['CodeAssure_id','Calculer_Sinistre','CodeAssure_id_adverse','IsPlannedFraud','dateOuvertureDuSinistre']
```

```
data = pd.DataFrame(columns = cols)
```

```
test = pd.DataFrame(columns = cols)
```

Figure 3.26: Planned Sinister Variables

Non Fraud Generation: We generated data for insured who had zero sinisters or plus without forgetting to include them with different percentages of responsibility

```
i = 0
lis = []
while i < 5000:
    ind = (random.randint(1,120000),random.randint(1,120000))
    if ind not in used:
        lis.append(ind)
        i+=1
```

```
for i in range(5000):
    randomdate=random.choice(dateslist)
    randomdate1=randomdate + timedelta(random.randint(90,365))
    data = data.append({'CodeAssure_id': lis[i][0], 'Calculer_Sinistre': 1, 'CodeAssure_id_adverse': lis[i][1], 'IsPlannedFraud': 0,'dateOuvertureDuSinis
    data = data.append({'CodeAssure_id': lis[i][1], 'Calculer_Sinistre': 0, 'CodeAssure_id_adverse': lis[i][0], 'IsPlannedFraud': 1,'dateOuvertureDuSinis
```

Figure 3.27: Generation non-Fraud Sinisters

Fraud Generation: For the fraud we made sure to include different number of insured involved in the network as much as possible, and to not interfere with the non fraud data thing that would make our dataset unbalanced.

```
i = 0
lis = []
while i < 5000:
    ind = (random.randint(1,120000),random.randint(1,120000))
    if ind not in used:
        lis.append(ind)
        i+=1
```

```
for i in range(5000):
    randomdate=random.choice(dateslist)
    randomdate1=randomdate + timedelta(random.randint(90,365))
    data = data.append({'CodeAssure_id': lis[i][0], 'Calculer_Sinistre': 1, 'CodeAssure_id_adverse': lis[i][1], 'IsPlannedFraud': 0,'dateOuvertureDuSinis
    data = data.append({'CodeAssure_id': lis[i][1], 'Calculer_Sinistre': 0, 'CodeAssure_id_adverse': lis[i][0], 'IsPlannedFraud': 1,'dateOuvertureDuSinis
```

Figure 3.28: Generation Fraud Sinisters

Chapter 4

DATA PREPARATION

4.1 Multi-insurance Fraud detection

At this part we are going to look at how to properly encode cyclical features for use .Many features commonly found in datasets are cyclical in nature. The best example of such a feature is of course time: months, days, weekdays, hours, minutes, seconds etc. are all cyclical in nature.We will be exploring this problem at the hand of predicting the multi assurance fraud detection.

```
dateAffectation=pd.to_datetime(df["DATE_AFFECTATION"])
dateInsertion=pd.to_datetime(df["dateInsertion"])
Date_Suspenssion=pd.to_datetime(df["Date_Suspenssion"])
DATE_RESILIATION=pd.to_datetime(df["DATE_RESILIATION"])
dateEffetPolice=pd.to_datetime(df["dateEffetPolice"])

dateAffectation_year=dateAffectation.dt.year
dateAffectation_day_sin=np.sin((2 * np.pi )/(dateAffectation.dt.day*30))
dateAffectation_day_cos= np.cos((2 * np.pi )/(dateAffectation.dt.day*30))
dateAffectation_month_sin=np.sin((2 * np.pi )/(dateAffectation.dt.month*12))
dateAffectation_month_cos = np.cos((2 * np.pi )/(dateAffectation.dt.month*12))
```

```
temp= pd.concat([dateAffectation_year,dateAffectation_day_sin,dateAffectation_day_cos,
                 dateAffectation_month_sin,dateAffectation_month_cos,dateInsertion_year,dateInsertion_day_sin,dateInsertion_day_cos,dateIns
],axis=1)
temp.columns =["dateAffectation_year","dateAffectation_day_sin","dateAffectation_day_cos",
               "dateAffectation_month_sin","dateAffectation_month_cos","dateInsertion_year",
               "dateInsertion_day_sin","dateInsertion_day_cos","dateInsertion_month_sin","dateInsertion_month_cos","Date_Suspenssion_year",
df= pd.concat([df,temp],axis=1)
df.drop(columns =["DATE_AFFECTATION","dateEffetPolice","DATE_RESILIATION","Date_Suspenssion"], axis=1, inplace=True)
df
```

Figure 4.1: BonusMalus dates Encoding

Encoding some features like we mentioned with implementing ordinal encoder and we drop the others which are useless:

```
enc_police= OrdinalEncoder()
columns_encoder_list=[]
X=df[['naturePolice','typePolice','Etat_Police','ETAT_VEHICULE']]
X_new=enc_police.fit_transform(X)
encoded_frame=pd.DataFrame(X_new)
encoded_frame.columns =['naturePolice','typePolice','Etat_Police','ETAT_VEHICULE']
df.drop(['naturePolice','typePolice','Etat_Police','ETAT_VEHICULE'], axis=1, inplace=True)
df=pd.concat([df,encoded_frame],axis=1)
df=df.dropna()
pickle.dump( OrdinalEncoder, open( "police_enc.p", "wb" ) )
df
```

Figure 4.2: Bonus-Malus variables OrdinalEncoding

One hot encoder refers to splitting the column which contains numerical categorical data to many columns depending on the number of categories present in that column, so we used to encoder for the feature "typeImmatriculaltion":

```
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder(handle_unknown='ignore', sparse=False)
x=df[['typeImmatriculation','energie']]
x = ohe.fit_transform(x)
col_names = ohe.get_feature_names(['typeImmatriculation','energie'])
encoded_frame=pd.DataFrame(x,index=df.index)
encoded_frame.columns =col_names
df.drop(columns =['typeImmatriculation','energie'], axis=1, inplace=True)
df= pd.concat([df,encoded_frame],axis=1)
pickle.dump( enc_police, open( "typeImmatriculation_energie_enc.p", "wb" ) )
df
```

```
columns_encoder_list=[ 'codeCompagnie_id','codeAgence','numPolice','numImmatriculation','dateInsertion']
```

```
columns_encoder_list=[ 'codeCompagnie_id','codeAgence','numPolice','numImmatriculation','dateInsertion']
enc = OrdinalEncoder()
X=df[columns_encoder_list]
df[columns_encoder_list]=enc.fit_transform(X)
pickle.dump(enc, open('columns_encoder_list.h', 'wb'))
```

Figure 4.3: Bonus-Malus variables OneHotEncoding

Replacing the missing values for the feature "date echeancePolice" with a nan-value

```
df[['dateEcheancePolice']].replace(np.nan, 0)
```

```
X_t=df[columns_list]
scaler = StandardScaler()
X_train = scaler.fit_transform(X_t)
```

```
X_train.shape
```

```
(19381, 49)
```

Figure 4.4: Bonus-Malus dataFrame scaling with StandardScaler

4.1.1 Clustering

We used 4 clustering techniques : KMeans ,IsolationForest,DBSCAN and Anomaly detection with auto encoders but we don't find good results .So we used the data clustering analysis to gain some valuable insights from our data by seeing what groups the data points fall into when we apply a clustering algorithm

```
pd.set_option("display.max_rows", 18557)
test=df.groupby(['CodeVehicule_id','dateEffetPolice_year','Etat_Police']).count()
test
```

```
test_2=test[test.CodePolice_id>1]
pd.set_option("display.max_rows", 19148 )
test_2
```

```
df['fraude_multi_assurance']=0
df['fraude_multi_assurance'].loc[df['CodeVehicule_id'].isin(test_2.index.get_level_values('CodeVehicule_id').values)]=1
df['fraude_multi_assurance'].value_counts()
```

```
df[df['fraude_multi_assurance']==1]
```

Figure 4.5: Fraud Detection Using Data Analytics

4.1.2 Classification

This is a problem as it is typically the minority class on which predictions are most important. One approach to addressing the problem of class imbalance is to randomly resample the training dataset. The two main approaches to randomly resampling an imbalanced dataset are to delete examples from the majority class, called undersampling, and to duplicate examples from the minority class, called oversampling. so we used them to solve this problem :

```
rOs = RandomOverSampler()
X, y = rOs.fit_resample(X, y)
```

Figure 4.6: **Over sampling**

```
rUs = RandomUnderSampler()
X, y = rUs.fit_resample(X_train, y_train)
```

Figure 4.7: **Undersampling**

This is a type of data augmentation for the minority class and is referred to as the Synthetic Minority Oversampling Technique, or SMOTE for short:

```
oversample = SMOTE()
X, y = oversample.fit_resample(X, y)
```

Figure 4.8: **SMOTE**

```
cc = ClusterCentroids(random_state=42)
X, y = cc.fit_resample(X_train, y_train)
```

Figure 4.9: **Clustering**

Dataset Splitting & Normalization

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

4.2 Sinister Fraud Detection

*We deleted the duplicated rows in this dataset which are 32291 row. *Then we corrected the mistaken and missing values in "pourcentageCompagnieAdverse" by subtracting "pourcentadeDeResponsabilite" from 100 "numeroDuSinistre", "numeroDePoliceCompagnieAdverse", "codeCompagnieAdverse", "numeroImmatriculationVehiculeAdverse", "mouvementDusinistre" contain a lot of missing values and the others we don't need them for our prediction, so we deleted them. *For the "lieuDuSinistre" coulumn, wedeleted the missing values then managed to replace bad cells with their closest match from took in a list of governorates.

For the encoding ,we used one hot encoder to transform categorical columns in which natural order does not exist

```
ohe = OneHotEncoder(handle_unknown='ignore', sparse=False)
oh_encoded_cols = ['natureDuSinistre','identificationTiers']

sinistreN = ohe.fit_transform(sinistre[oh_encoded_cols])
col_names = ohe.get_feature_names(oh_encoded_cols)
```

Figure 4.10: Sinister variables OneHotEncoding

As mentionned earlier, we convert the date features into datetime which has been encoded into cyclical ordinal attributes

```
dateOuvertureDuSinistre = pd.to_datetime(sinistre["dateOuvertureDuSinistre"])
dateDeSurvenanceDuSinistre = pd.to_datetime(sinistre["dateDeSurvenanceDuSinistre"])

dateOuvertureDuSinistre_year = dateOuvertureDuSinistre.dt.year
dateOuvertureDuSinistre_month_cos = np.cos((2*np.pi)/12*dateOuvertureDuSinistre.dt.month)
dateOuvertureDuSinistre_month_sin = np.sin((2*np.pi)/12*dateOuvertureDuSinistre.dt.month)
dateOuvertureDuSinistre_day_sin = np.cos((2*np.pi)/30*dateOuvertureDuSinistre.dt.day)
dateOuvertureDuSinistre_day_cos = np.cos((2*np.pi)/30*dateOuvertureDuSinistre.dt.day)
```

```
sinistre['dateOuvertureDuSinistre_year'] = dateOuvertureDuSinistre_year
sinistre['dateOuvertureDuSinistre_month_cos'] = dateOuvertureDuSinistre_month_cos
sinistre['dateOuvertureDuSinistre_month_sin'] = dateOuvertureDuSinistre_month_sin
sinistre['dateOuvertureDuSinistre_day_cos'] = dateOuvertureDuSinistre_day_cos
sinistre['dateOuvertureDuSinistre_day_sin'] = dateOuvertureDuSinistre_day_sin
```

Figure 4.11: Sinister dates Encoding

4.3 Bonus malus class prediction

4.3.1 Police data set

```
rows=[]
for row in police['dateEcheancePolice']:
    a=str(row)
    if len(a)==5 :
        a='0'+a[0:4]
    row=a[0:2]+'.'+a[2:4]
    rows.append(row)
```

```
police.drop(columns=['dateEcheancePolice'], axis=1, inplace=True)
police=pd.concat([police, dateEcheancePolice], axis=1)
```

Figure 4.12: Separating days from months in dateEcheancePolice and adding it to the dataFrame

4.3.2 Vehicule data set

we deleted the unnecessary coloumns and we fill the column "energie" with "gasoil" in the table"vehicule"

```
Vehicule.drop(columns=["numChassis","dateDerniereVisite","dateAjout","etatVehicule",
                       "dateMiseEpave","DATE_RETRAIT","dateMiseCirculation","dateMiseAjourVehicule" ], axis=1, inplace=True)
```

Figure 4.13: Dropping unnecessary Police variables

```
Vehicule['energie']=Vehicule['energie'].fillna('GASOIL')
```

Figure 4.14: Replacing missing values in Column "energie"

4.3.3 Merge data

We make the correspondence between Vehicule and police class

```
police_class=pd.merge(police,data_transf,on='CodePolice_id')
Vehicule_police_class=pd.merge(police_class,Vehicule,on='CodeVehicule_id')
```

Figure 4.15: Merging tables Police and Vehicule

we transform the time features to Cyclic Ordinal attributes:A common method for encoding cyclical data is to transform the data into two dimensions using a sine and consine transformation.

Let's do this for our hourly data:

```
dateInsertion=pd.to_datetime(Vehicule_police_class["dateInsertion"])
Date_Suspenssion=pd.to_datetime(Vehicule_police_class["Date_Suspenssion"])
DATE_RESILIATION=pd.to_datetime(Vehicule_police_class["DATE_RESILIATION"])
dateEffetPolice=pd.to_datetime(Vehicule_police_class["dateEffetPolice"])

dateInsertion_year=dateInsertion.dt.year
dateInsertion_day_sin=np.sin((2 * np.pi )/(dateInsertion.dt.day*30))
dateInsertion_day_cos= np.cos((2 * np.pi )/(dateInsertion.dt.day*30))
dateInsertion_month_sin=np.sin((2 * np.pi )/(dateInsertion.dt.month*12))
dateInsertion_month_cos = np.cos((2 * np.pi )/(dateInsertion.dt.month*12))

Date_Suspenssion_year=Date_Suspenssion.dt.year
Date_Suspenssion_day_sin=np.sin((2 * np.pi )/(Date_Suspenssion.dt.day*30))
Date_Suspenssion_day_cos= np.cos((2 * np.pi )/(Date_Suspenssion.dt.day*30))
Date_Suspenssion_month_sin=np.sin((2 * np.pi )/(Date_Suspenssion.dt.month*12))
Date_Suspenssion_month_cos = np.cos((2 * np.pi )/(Date_Suspenssion.dt.month*12))


DATE_RESILIATION_year=DATE_RESILIATION.dt.year
DATE_RESILIATION_day_sin=np.sin((2 * np.pi )/(DATE_RESILIATION.dt.day*30))
DATE_RESILIATION_day_cos= np.cos((2 * np.pi )/(DATE_RESILIATION.dt.day*30))
DATE_RESILIATION_month_sin=np.sin((2 * np.pi )/(DATE_RESILIATION.dt.month*12))
DATE_RESILIATION_month_cos = np.cos((2 * np.pi )/(DATE_RESILIATION.dt.month*12))


dateEffetPolice_year=dateEffetPolice.dt.year
dateEffetPolice_day_sin=np.sin((2 * np.pi )/(dateEffetPolice.dt.day*30))
dateEffetPolice_day_cos= np.cos((2 * np.pi )/(dateEffetPolice.dt.day*30))
dateEffetPolice_month_sin=np.sin((2 * np.pi )/(dateEffetPolice.dt.month*12))
dateEffetPolice_month_cos = np.cos((2 * np.pi )/(dateEffetPolice.dt.month*12))
```

```
temp= pd.concat([dateInsertion_year,dateInsertion_day_sin,dateInsertion_day_cos,
                 dateInsertion_month_sin,dateInsertion_month_cos,Date_Suspenssion_year,
                 Date_Suspenssion_day_sin,Date_Suspenssion_day_cos,Date_Suspenssion_month_sin,
                 Date_Suspenssion_month_cos,DATE_RESILIATION_month_cos,DATE_RESILIATION_month_sin,DATE_RESILIATION_day_cos,
                 DATE_RESILIATION_day_sin,DATE_RESILIATION_year,dateEffetPolice_year,dateEffetPolice_day_sin,dateEffetPolice_day_cos,
                 dateEffetPolice_month_sin,dateEffetPolice_month_cos],axis=1)
temp.columns =["dateInsertion_year","dateInsertion_day_sin","dateInsertion_day_cos","dateInsertion_month_sin","dateInsertion_month_cos","Date_Suspenssi
Vehicule_police_class= pd.concat([Vehicule_police_class,temp],axis=1)
Vehicule_police_class.drop(columns =["YEAR_DATE_AFFECTATION","MONTH_DATE_AFFECTATION","DAY_DATE_AFFECTATION","dateEffetPolice","DATE_RESILIATION","Date
Vehicule_police_class
```

Figure 4.16: Encoding time features

we encode categorical data with implementing of one hot encoder method :

```python
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder(handle_unknown='ignore', sparse=False)
x=Vehicule_police_class[['typeImmatriculation']]
x = ohe.fit_transform(x)
col_names = ohe.get_feature_names(['typeImmatriculation'])
pickle.dump( ohe, open( "typeImmatriculation_enc.p", "wb" ) )
```

```python
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder(handle_unknown='ignore', sparse=False)
x=Vehicule_police_class[['typeImmatriculation','energie']]
x = ohe.fit_transform(x)
col_names = ohe.get_feature_names(['typeImmatriculation','energie'])
encoded_frame=pd.DataFrame(x)
encoded_frame.columns =col_names
Vehicule_police_class.drop(columns =['typeImmatriculation','energie'], axis=1, inplace=True)
Vehicule_police_class= pd.concat([Vehicule_police_class,encoded_frame],axis=1)
pickle.dump( enc_police, open( "typeImmatriculation_enc.p", "wb" ) )
Vehicule_police_class
```

Figure 4.17: OneHotEncoder for "typeImmatriculation" and "energie"

4.4 Wrecked car detector

In our case, we managed to collect 248 images for training, 76 for validation, and 6 images for testing which is a pretty small dataset for computer vision models.
So to expand our dataset, we will use the data augmentation technique using the Keras ImageDataGenerator class by applying random transformations on each training image as it is passed to the model. This will not only make your model robust but will also save up on the overhead memory.
As we can see in figure 4.6, we used the following transformations in our ImageDataGenerator instance :

- featurewise_center = True : to set input mean to 0 over the dataset.
- samplewise_center=True : to set each sample mean to 0.
- rotation_range=10 : to set the degree range for random rotations.
- zoom_range = 0.1 : to set the range for random zoom.
- width_shift_range = 0.1 : to set the range for random shifts to the left or the right (horizontal shifts).
- height_shift_range = 0.1 : to set the range for random shifts up or down (vertical shifts)
- horizontal_flip=True: to randomly flip inputs horizontally.
- vertical_flip=False : to randomly flip inputs vertically.

```
datagen = ImageDataGenerator(
    featurewise_center=True,
    samplewise_center=True,
    rotation_range=10,
    zoom_range = 0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=False
    )
```

```
train_it = datagen.flow_from_directory('cars/train/',
                                       target_size=(224, 224),
                                       color_mode='rgb',
                                       class_mode='binary',
                                       batch_size=8)

test_it = datagen.flow_from_directory('cars/validate/',
                                      target_size=(224, 224),
                                      color_mode='rgb',
                                      class_mode='binary',
                                      batch_size=8)
```

```
Found 248 images belonging to 2 classes.
Found 76 images belonging to 2 classes.
```

Figure 4.18: ImageDataGenerator instance

From just one image, we can get multiple images by applying one or multiple transformations as shown in figure 4.7.

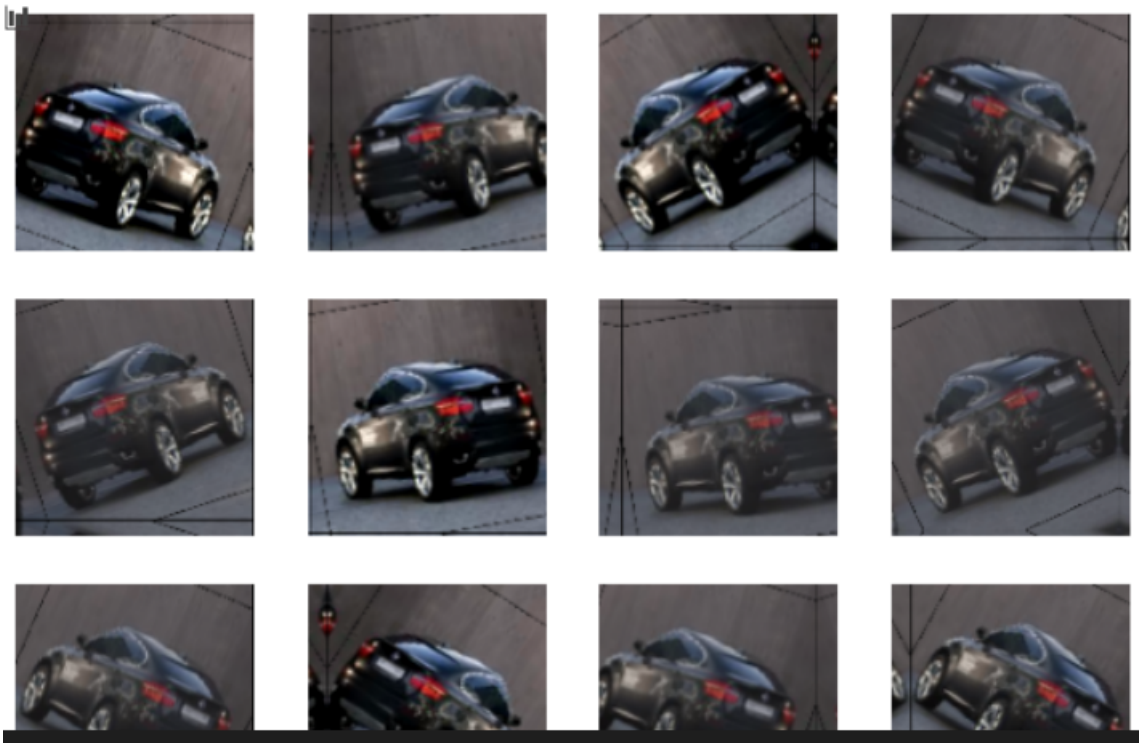


Figure 4.19: Data augmentation result

Chapter 5

Data Visualisation

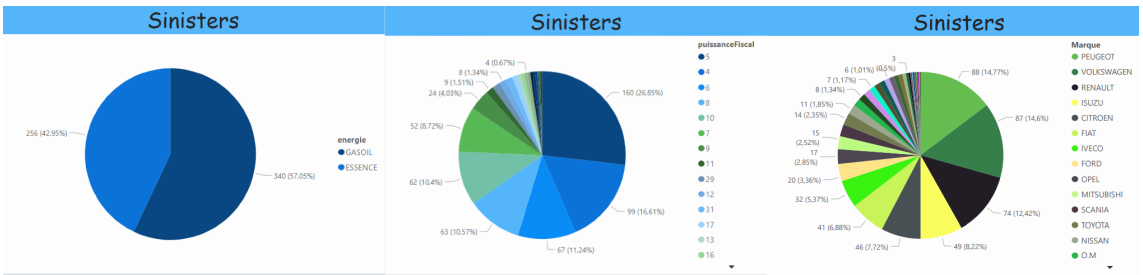


Figure 5.1: Pie charts of sinisters By Energie ,Fiscal power and Vehicle brand

Gasoil and 5 fiscal power vehicles are the main categories in accidents allong with Peugeaut, Volkswagen, Renault and Isuzu vehicles which are involved in half of the sinisters

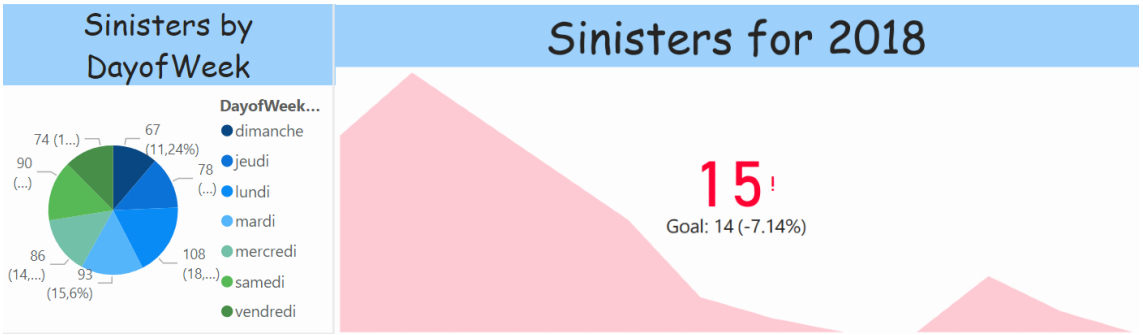


Figure 5.2: Sinisters By Day and Year

- Accidents has increased with 7.14 % in 2018 compared to 2017.
- Accident numbers are lower on weekends

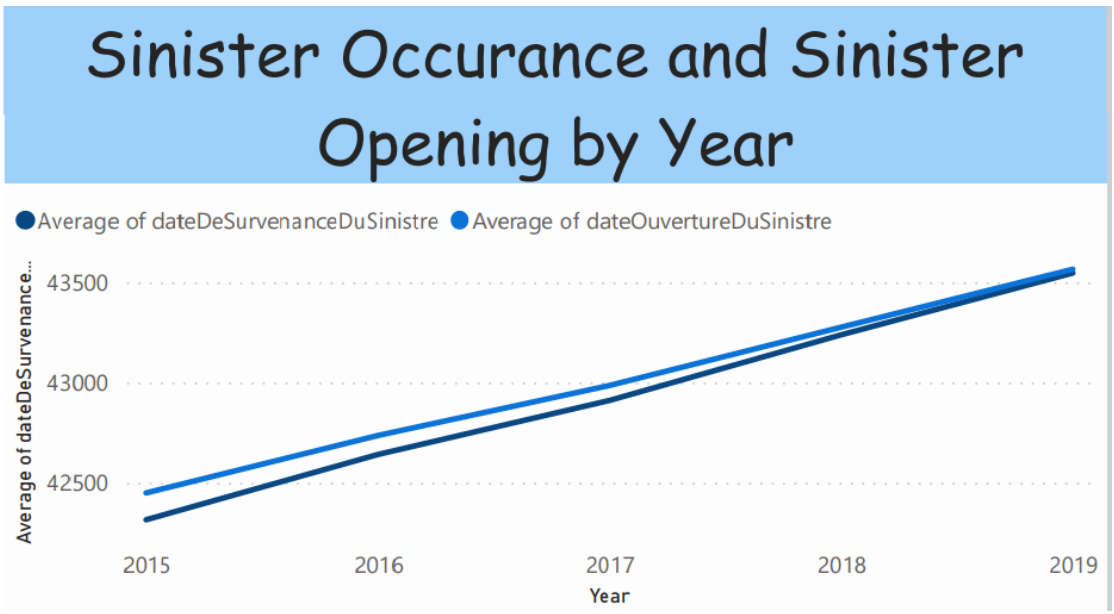


Figure 5.3: Line chart for difference between the Occurence and the Opening of the sinister

Sinister claims handling has improved since the time to open a sinister is constantly decreasing

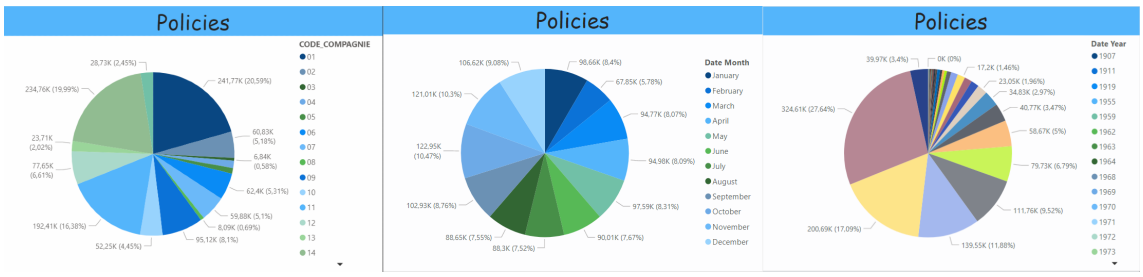


Figure 5.4: Pie charts of Policies by companies , month and year

This pie charts above show that companies 01 and 14 are more likely to sign new policies which are higher in the last quarter of the year.

The count of policies has significantly increased starting from the year 2017

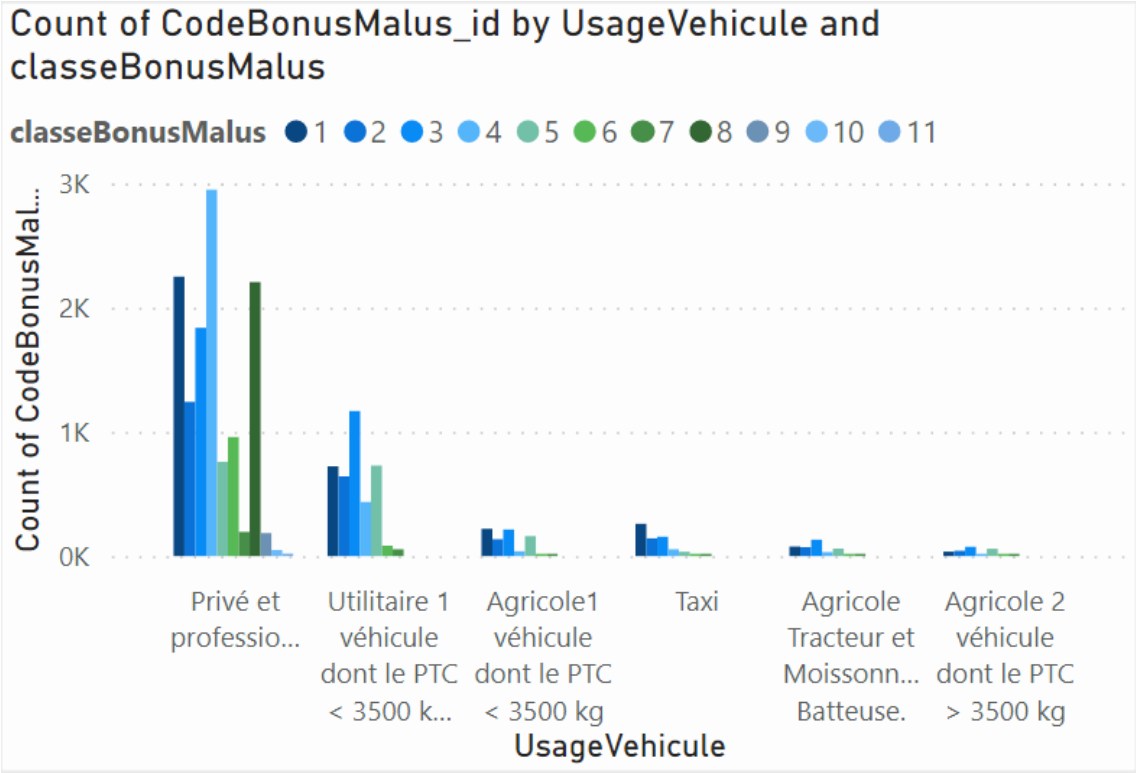


Figure 5.5: Clustered column chart of bonus malus number by UsageVehicle and classBonusMalus

The Clustered column chart shows first that the policies are mostly for Professional and private sector in which the dominating bonus-Malus classes are 4 followed by 1, 8 and 3.

Chapter 6

Data Modeling

6.1 Multi-insurance Fraud detection

We tried to use boosting algorithms which isan ensemble modeling technique which attempts to build a strong classifier from the number of weak classifiers.

6.1.1 BoostingClassifier

```
[ ]  1  estimator.score(X_test_scaled, y_test)
```

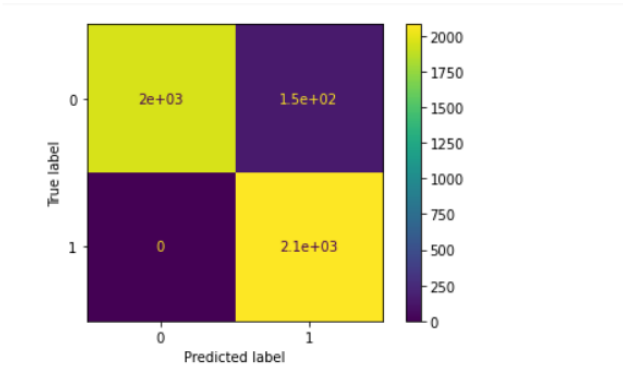
RandomOverSampler 0.9646271510516252

Undersampling 0.94445

ClusterCentroids 0.9219858156028369

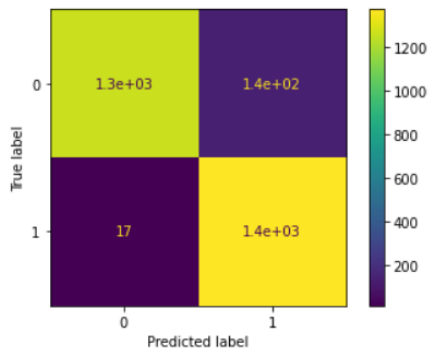
The results obtained for each Technique for extremely unbalanced data :

Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class,we presented the confusion matrix for each model like this:



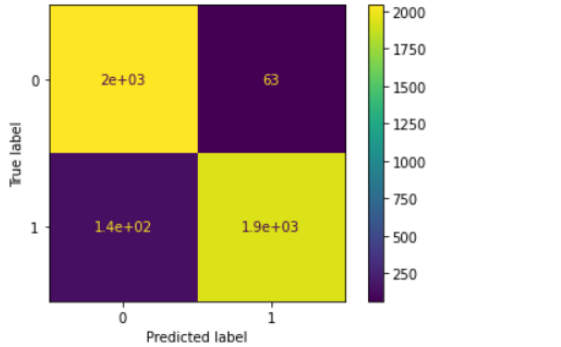
	precision	recall	f1-score	support
0	1.00	0.93	0.96	2104
1	0.93	1.00	0.97	2080
accuracy			0.96	4184
macro avg	0.97	0.96	0.96	4184
weighted avg	0.97	0.96	0.96	4184

Figure 6.1: **Over sampling**



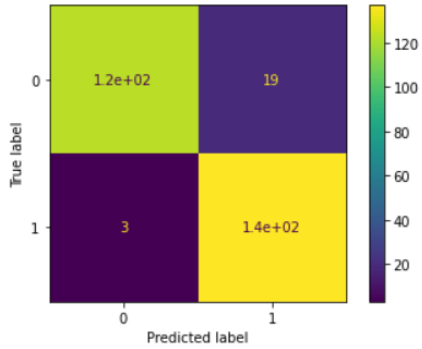
	precision	recall	f1-score	support
0	0.99	0.90	0.94	1405
1	0.91	0.99	0.95	1391
accuracy			0.94	2796
macro avg	0.95	0.94	0.94	2796
weighted avg	0.95	0.94	0.94	2796

Figure 6.2: Confusion matrix for Undersampling



	precision	recall	f1-score	support
0	0.94	0.97	0.95	2104
1	0.97	0.93	0.95	2080
accuracy			0.95	4184
macro avg	0.95	0.95	0.95	4184
weighted avg	0.95	0.95	0.95	4184

Figure 6.3: *SMOTE*



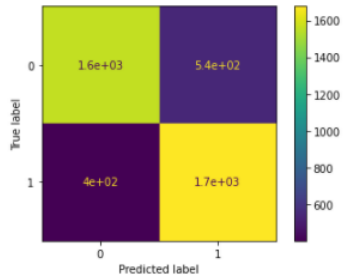
	precision	recall	f1-score	support
0	0.98	0.87	0.92	142
1	0.88	0.98	0.93	140
accuracy			0.92	282
macro avg	0.93	0.92	0.92	282
weighted avg	0.93	0.92	0.92	282

Figure 6.4: Confusion matrix for ClusterCentroids

6.1.2 Gradient Boosting Classifier

model evaluation

```
plot_confusion_matrix(clf,X_test_scaled, y_test)
plt.show()
y_pred=clf.predict(X_test_scaled)
print(classification_report(y_test, y_pred))
```



	precision	recall	f1-score	support
0	0.79	0.74	0.77	2104
1	0.75	0.81	0.78	2080
accuracy			0.77	4184
macro avg	0.77	0.77	0.77	4184
weighted avg	0.77	0.77	0.77	4184

Figure 6.5: Confusion matrix for Gradient Boosting Classifier model

6.1.3 Neural network classification

The results obtained for each Technique for extremely unbalanced data

```
Accuracy on training data: 0.97374618053436
 Error on training data: 0.0262538194656372
Accuracy on test data: 0.9232791662216187%
 Error on test data: 0.07672083377838135
```

```
from sklearn.metrics import confusion_matri
confusion_matrix(y_test, pred_test)
```

```
array([[1810,  294],
       [  27, 2053]])
```

Figure 6.6: **over sampling**

```
Accuracy on training data: 0.9434261322021484%
 Error on training data: 0.05657386779785156
Accuracy on test data: 0.8848354816436768%
 Error on test data: 0.11516451835632324
```

```
pred_test=np.rint(pred_test)
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred_test)
```

```
array([[1214,  191],
       [ 131, 1260]])
```

Figure 6.7: Confusion matrix for Undersampling model

```
Accuracy on training data: 0.9437249898
 Error on training data: 0.05627501010
Accuracy on test data: 0.8793020844459
 Error on test data: 0.120697915554046
```

```
[97] pred_test=np.rint(pred_test)
     from sklearn.metrics import confusion_
     confusion_matrix(y_test, pred_test)
```

```
array([[1793,  311],
       [ 194, 1886]])
```

Figure 6.8: **Confusion matrix for SMOTE model**

```
Accuracy on training data: 1.0%
 Error on training data: 0.0
Accuracy on test data: 0.8971630930900574%
 Error on test data: 0.10283690690994263
```

```
3] pred_test=np.rint(pred_test)
   from sklearn.metrics import confusion_matrix
   confusion_matrix(y_test, pred_test)
```

```
array([[120,  22],
       [  7, 133]])
```

Figure 6.9: Confusion matrix for ClusterCentroids model

6.2 Bonus Malus class prediction

We tried to train our model using decision tree classifier and finally we find the accuracy score as following:

```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(train_X, train_y)
predictions_dt = decision_tree.predict(test_X)
acc_decision_tree = accuracy_score(test_y, predictions_dt)
acc_decision_tree
```

```
0.9960438645197113
```

Figure 6.10: implementing of Decision tree algorithm

Kfold Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds with shuffling:.

Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

```
k_fold = KFold(n_splits = 10, shuffle = True, random_state  =0)
tree = tree.DecisionTreeClassifier()
scoring = 'accuracy'
score = cross_val_score(tree, all_X, all_y, cv= k_fold, n_jobs=1, scoring=scoring)
print(score)
```

```
[0.99597446 0.99736258 0.99666852 0.99569684 0.99625208 0.99569684
 0.99611327 0.99652971 0.99722376 0.9959739 ]
```

Figure 6.11: Decision tree Classifier

we meant the scores that we founded in training the model with K-fold:

```
(np.mean(score))
bagging_cart =np.mean(score)
bagging_cart
```

```
0.9963491945294249
```

Figure 6.12: mean of all scores

Bagging (Bootstrap Aggregation) is used when our goal is to reduce the variance of a decision tree. Here idea is to create several subsets of data from training sample chosen randomly with replacement. Now, each collection of subset data is used to train their decision trees. As a result, we end up with an ensemble of different models. Average of all the predictions from different trees are used which is more robust than a single decision tree.so we tried to used in our model:

```

# Bagged Decision Trees for Classification
X = all_X
Y = all_y
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
cart = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Sett
  FutureWarning
0.9950027723811482
```

Figure 6.13: implementing of bagging decision tree algorithm

We tried to train our model using gradient boosting algorithm which is a method of converting weak learners into strong learners.Finally we find the score as following

```
X = all_X
Y = all_y
seed = 7
num_trees = 100
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:296: FutureW
  FutureWarning
0.9931982135736368
```

Figure 6.14: implementing of GradientBoosting algorithm

	Model	Score
9	bagging CART	0.996349
6	Decision Tree	0.995766
2	Random Forest	0.875625
3	Naive Bayes	0.429067
7	AdaBoost	0.321847
1	KNN	0.292823
8	Ensemble voting	0.268963
0	Logistic Regression	0.239034
4	Perceptron	0.191560
5	Stochastic Gradient Decent	0.190242

Figure 6.15: the performance of the used algorithms

6.3 Sinister Fraud Detection

-Set the weights of classifiers and training the data sample in each iteration such that it ensures the accurate predictions of unusual observations with the help of adaboost classifier

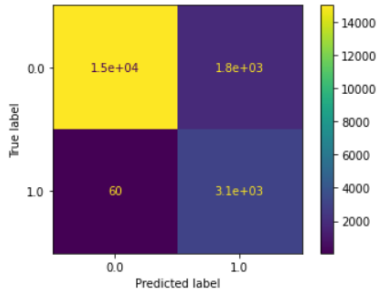
```
[15]
ad = AdaBoostClassifier(algorithm = 'SAMME.R', learning_rate = 0.2, n_estimators = 70)
ad.fit(x_train,y)
print('train score: ',ad.score(x_train,y))
y_pred = ad.predict(x_test)
print('test score',ad.score(x_test, y_test))
```

```
train score:  0.907234434066044
test score 0.9047404740474048
```

Figure 6.16: implementing AdaBoost algorithm

Calculation and make plot of the confusion matrix with the library sklearn.metrics and the function plot-confusion-matrix:

```
from sklearn.metrics import roc_auc_score, roc_curve, plot_confusion_matrix
plot_confusion_matrix(ad, x_test, y_test)
plt.show()
y_pred=ad.predict(x_test)
print(classification_report(y_test, y_pred))
```




```
              precision    recall  f1-score   support

         0.0       1.00      0.89      0.94     16848
         1.0       0.63      0.98      0.76      3150

    accuracy                           0.90     19998
   macro avg       0.81      0.94      0.85     19998
weighted avg       0.94      0.90      0.91     19998
```

Figure 6.17: Confusion matrix AdaBoost

Fit on the training data with the implementation of Support Vector Classification SVC:

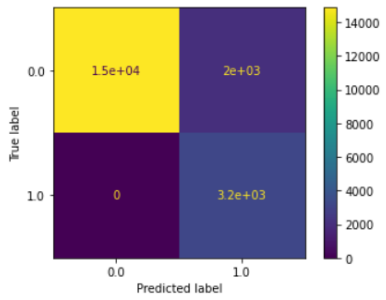
```
from sklearn.svm import SVC

# Instantiate classifier
svc = SVC(kernel = "linear") # changed to "linear"

# fit on training data
svc.fit(x_train, y)

# Seeing the metrics
print("Accuracy on training set: {:.3f}".format(svc.score(x_train, y)))
print("Accuracy on test set: {:.3f}".format(svc.score(x_test, y_test)))
```

```
Accuracy on training set: 0.902
Accuracy on test set: 0.901
```

Figure 6.18: implementing Support Vector Classification SVC algorithm

Calculation and make plot of the confusion matrix with the library sklearn.metrics and the function plot-confusion-matrix:

```python
from sklearn.metrics import roc_auc_score, roc_curve, plot_confusion_matrix
plot_confusion_matrix(svc, x_test, y_test)
plt.show()
y_pred=ad.predict(x_test)
print(classification_report(y_test, y_pred))
```




```
              precision    recall  f1-score   support

         0.0       1.00      0.89      0.94     16848
         1.0       0.63      0.98      0.76      3150

    accuracy                           0.90     19998
   macro avg       0.81      0.94      0.85     19998
weighted avg       0.94      0.90      0.91     19998
```

Figure 6.19: Confusion matrix SVC

Now our objectif for this model is to try as possible to minimize the False positive, so that none of the fraud attempt goes unsuspicious. After running several models we ended up with Adaboost and SVC. As shown on the below confusion matrices, and since the two models has almost the same accuracy, we can notice that the false positive is 60 for the AdaBoost while the SVC model has exactly what we were hoping for which is 0 false positive. Now that we know that SVC is what we are looking for,now we try to tune the hyperparameters by comparing the different performances of each combination using GridSearchCV.

```python
# I'll add a standard scaler since SVC works better if the data is scaled.
pipe = Pipeline([ ('classifier', SVC(kernel = "linear"))])

# Next we'll tune hyperparameters of the estimators separately in the pipeline
param_grid = [
    {'classifier': [SVC()],
    'classifier__gamma': [0.001, 0.01, 0.1, 1, 10],
    'classifier__C': [0.001, 0.01, 0.1, 1, 10]}]
# grid search with cross-validation
grid = GridSearchCV(pipe, param_grid, cv = 5)
grid.fit(x_train, y)

print("Best params:\n{}\n".format(grid.best_params_))
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
print("Test-set score: {:.2f}".format(grid.score(x_test, y_test)))
```

Figure 6.20: Tuning hyper parameters with GridSearchCV

Finally we displayed the new accuracy with the best parameters :

```python
print("Test-set score: {:.4f}".format(grid.score(x_test, y_test)))
```

```
Test-set score: 0.9036
```

Figure 6.21: Final Accuracy

6.4 Wrecked Car Detector

As we know simple Neural Networks don't perform very well on image classification especially colored images with 3 channels, that's why we are opting for a CNN architecture.
Conceptually, a CNNs earlier layers extract low-level features such as edges. Later layers use those lower-level features to extract higher-level features such as shapes, but building a reliable CNN model from scratch can

be very time-consuming and expensive. So to reduce computation and processing costs, we will use the transfer learning technique.
As we can see in figure 6.10, our model will be based on the vgg16 from the University of Oxford pre-trained on the ImageNet large dataset. The ImageNet pre-trained models are often good choices for computer vision transfer learning, as they have learned to classify many many different types of images. In doing this, they have learned to detect many different types of features that could be valuable in image recognition.
The last layer of an ImageNet model is a dense layer of 1000 units, representing the 1000 possible classes in the dataset. In our case, we want it to make a binary classification. Because we want the classification to be different, we are going to remove the last layer of the model. We can do this by setting the flag include_top=False when downloading the model.

```
base_model = keras.applications.VGG16(
    weights='imagenet',
    input_shape=(224, 224, 3),
    include_top=False)
base_model.trainable = True
```

```
inputs = keras.Input(shape=(224, 224, 3))
x = base_model(inputs, training=False)
x = keras.layers.GlobalAveragePooling2D()(x)
x = keras.layers.Dense(32)(x)
x = keras.layers.Dense(32)(x)
outputs = keras.layers.Dense(1)(x)
model = keras.Model(inputs, outputs, name="wrecked car detector")
```

Figure 6.22: Building wrecked car detector model

After removing this top layer, we added a GlobalAveragePooling layer to reduce the number of parameters and to unstack the features as flatten does, then 2 hidden layers to make the model more accurate, and finally an output layer with 1 neuron to gave us the result whether the car is wrecked or not.

```
model.summary()
```

```
Model: "wrecked car detector"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                16416
_________________________________________________________________
dense_4 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
Total params: 14,732,193
Trainable params: 14,732,193
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.23: Wrecked car detector model summary

As an optimizer, we chose the RMSprop, with a small learning rate =0.0001, but our optimizer will adjust it depending on the loss value of the previous epoch.
For the metrics Since it's a two-class classification problem, we will use the binary versions of Crossentropy and Accuracy.

```
model.compile(optimizer=keras.optimizers.RMSprop(learning_rate = .0001),
              loss=keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=[keras.metrics.BinaryAccuracy()])
```

Figure 6.24: Wrecked car detector model compiling

We defined a callback instance that will monitor the fitting process and stop it whenever the model hits 96% accuracy on the validation dataset.

```
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('val_binary_accuracy')>0.96):
      print("\nReached 96% accuracy so cancelling training!")
      self.model.stop_training = True

callbacks = myCallback()
```

Figure 6.25: Callback instance

We begin now with the training process, we set the training, validation sets, and a number of epochs = 50. The model reaches 96% accuracy after 14 epochs.

```
history = model.fit(train_it,
validation_data=test_it,
epochs=50,
callbacks=[callbacks]
)
```

```
31/31 [==============================] - 119s 4s/step - loss: 0.2915 - binary_accuracy: 0.9113 - val_loss: 0.7544 - val_binary_accuracy: 0.6974
Epoch 4/50
31/31 [==============================] - 118s 4s/step - loss: 0.2244 - binary_accuracy: 0.9274 - val_loss: 0.3800 - val_binary_accuracy: 0.9079
Epoch 5/50
31/31 [==============================] - 118s 4s/step - loss: 0.0894 - binary_accuracy: 0.9677 - val_loss: 8.4177 - val_binary_accuracy: 0.7105
Epoch 6/50
31/31 [==============================] - 112s 4s/step - loss: 0.3560 - binary_accuracy: 0.9516 - val_loss: 0.8534 - val_binary_accuracy: 0.7368
Epoch 7/50
31/31 [==============================] - 109s 4s/step - loss: 0.1653 - binary_accuracy: 0.9556 - val_loss: 0.4813 - val_binary_accuracy: 0.8947
Epoch 8/50
31/31 [==============================] - 114s 4s/step - loss: 0.2693 - binary_accuracy: 0.9718 - val_loss: 0.5667 - val_binary_accuracy: 0.7632
Epoch 9/50
31/31 [==============================] - 109s 4s/step - loss: 0.2351 - binary_accuracy: 0.9435 - val_loss: 0.2799 - val_binary_accuracy: 0.9342
Epoch 10/50
31/31 [==============================] - 111s 4s/step - loss: 0.1967 - binary_accuracy: 0.9677 - val_loss: 0.4551 - val_binary_accuracy: 0.8289
Epoch 11/50
31/31 [==============================] - 108s 3s/step - loss: 0.1843 - binary_accuracy: 0.9637 - val_loss: 0.3950 - val_binary_accuracy: 0.8816
Epoch 12/50
31/31 [==============================] - 108s 3s/step - loss: 0.3886 - binary_accuracy: 0.9315 - val_loss: 0.4026 - val_binary_accuracy: 0.8553
Epoch 13/50
31/31 [==============================] - 107s 3s/step - loss: 0.1381 - binary_accuracy: 0.9597 - val_loss: 1.8106 - val_binary_accuracy: 0.7368
Epoch 14/50
31/31 [==============================] - ETA: 0s - loss: 0.1667 - binary_accuracy: 0.9476
Reached 96% accuracy so cancelling training!
31/31 [==============================] - 107s 3s/step - loss: 0.1667 - binary_accuracy: 0.9476 - val_loss: 0.1425 - val_binary_accuracy: 0.9605
```

Figure 6.26: Fitting the model

In the plots here as we can see the evolution of validation and training loss and accuracy along with the number of epochs.

```
history_frame = pd.DataFrame(history.history)
history_frame.loc[:, ['loss', 'val_loss']].plot()
history_frame.loc[:, ['binary_accuracy', 'val_binary_accuracy']].plot()
```

```
<AxesSubplot:>
```

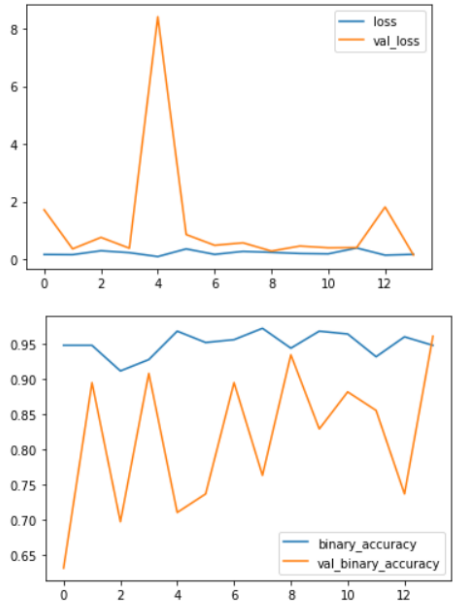


Figure 6.27: Loss and accuracy plots

After fitting the model, it's the moment for testing it on new images. We will use 2 functions to show images and make_predictions to show each image in the test dataset and its prediction whether it"s wrecked or not.

```
def show_image(image_path):
    image = mpimg.imread(image_path)
    plt.imshow(image)

def make_predictions(image_path):
    show_image(image_path)
    image = image_utils.load_img(image_path, target_size=(224, 224))
    image = image_utils.img_to_array(image)
    image = image.reshape(1,224,224,3)
    image = preprocess_input(image)
    preds = model.predict(image)
    print(preds)
    if preds[0] < 0:
        print("This vehicle is totally wrecked!")
    else:
        print("Fraud attempt! This car is not totally wrecked")
```

Figure 6.28: Visualization helpful functions

We test now our model on new images from the test dataset, we chose a wrecked car, slightly damaged car, and a new car, and our model predicted them correctly.

Figure 6.29: Wrecked car

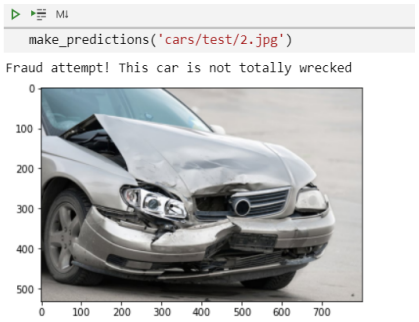


Figure 6.30: Slightly damaged car

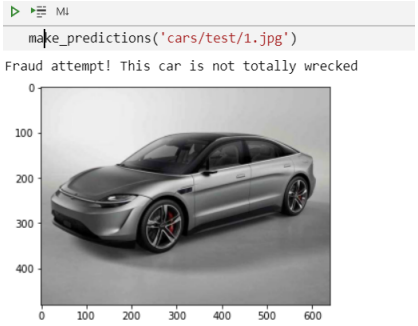


Figure 6.31: Brand new car

In the deployment phase, we have to serialize the model to a JSON file and export the weights to an h5 file.

```
# serialize model to JSON
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

Figure 6.32: Exporting the model

Chapter 7

Deployment

7.1 Introduction

Deploying machine learning models into production can be done in a wide variety of ways

7.2 Tools &Technology

7.2.1 Microsoft Power BI

7.2.1.1 Introduction

Power BI is a cloud-based tool that requires no capital expenditure or infrastructure support regardless of the size of a business. The modern iteration of the tool is free from legacy software constraints and its users do not need any specialized training in order to generate business intelligence insights.

Figure 7.1: Power BI logo

7.2.1.2 Screenshots

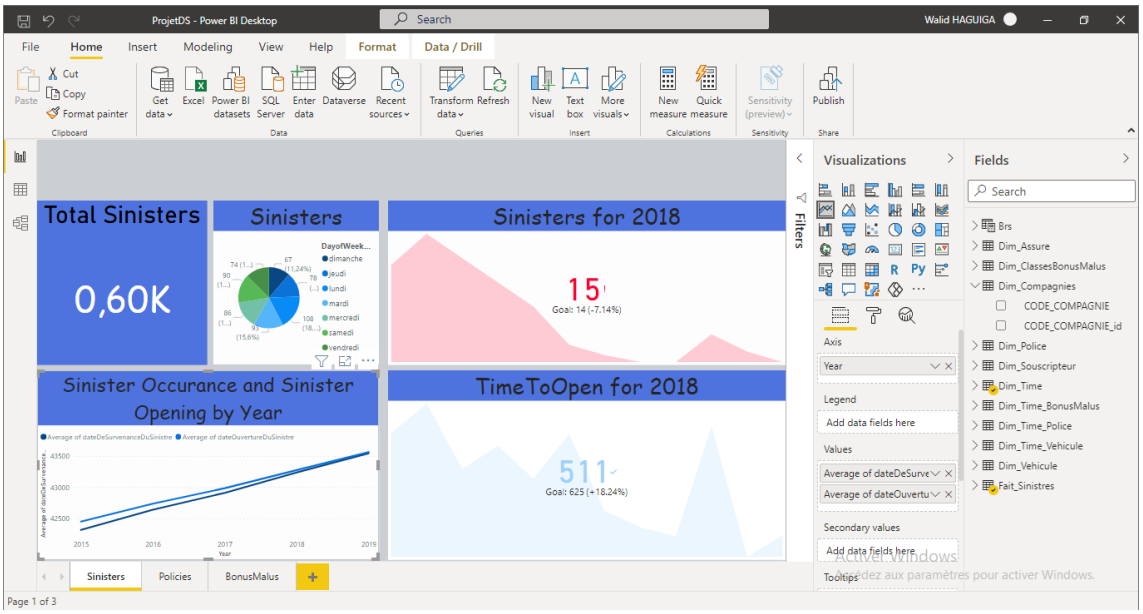


Figure 7.2: Dashboard overview

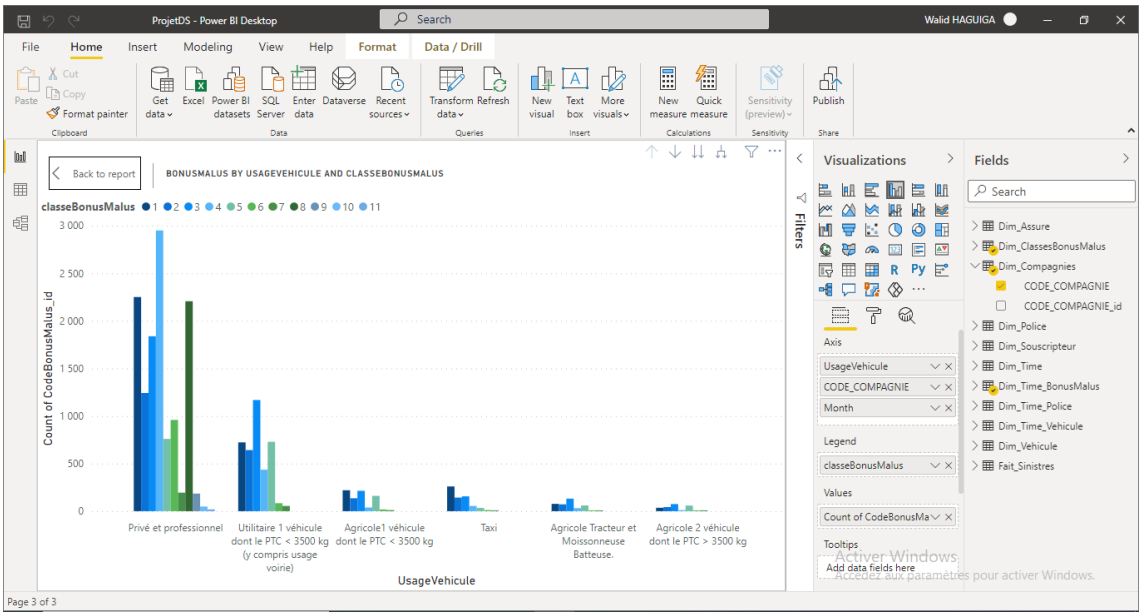


Figure 7.3: Visuals Configurations

7.2.2 Django

7.2.2.1 Introduction

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

Figure 7.4: Django logo

7.2.2.2 Screenshots

The user uploads a car image and presses "Submit", to detect whether the car is repairable or not.

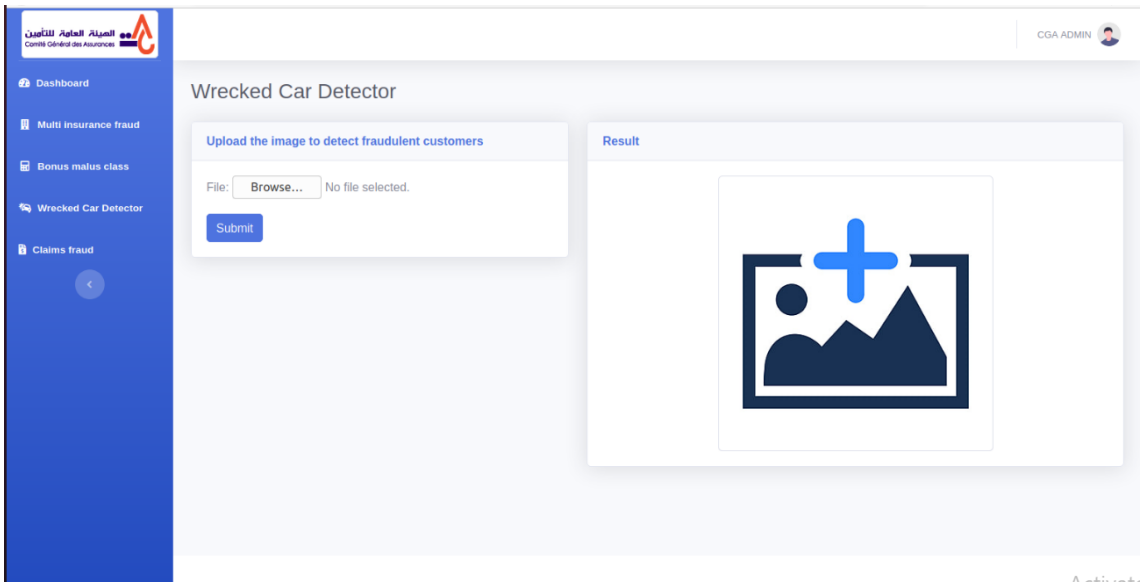


Figure 7.5: Wrecked car Detector interface

The user indicates all the requested information and presses "Submit", the result is displayed indicating the class of bonus malus expected.

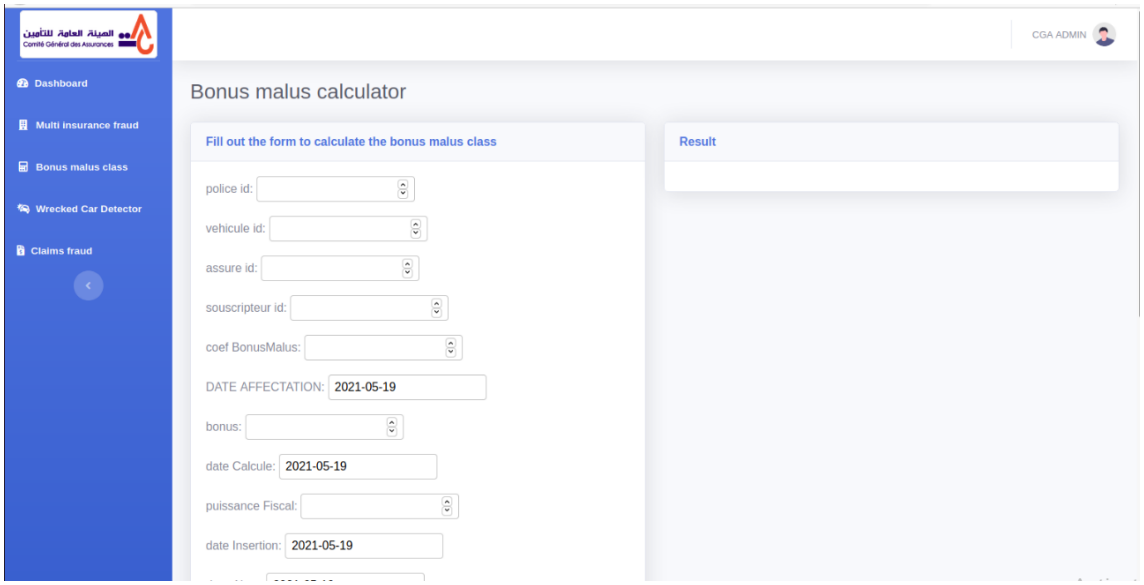


Figure 7.6: Bonus malus calculator interface

The user downloads a file that contains all the requested information and presses "Submit", another web page is displayed showing the list of fraudulent customers.

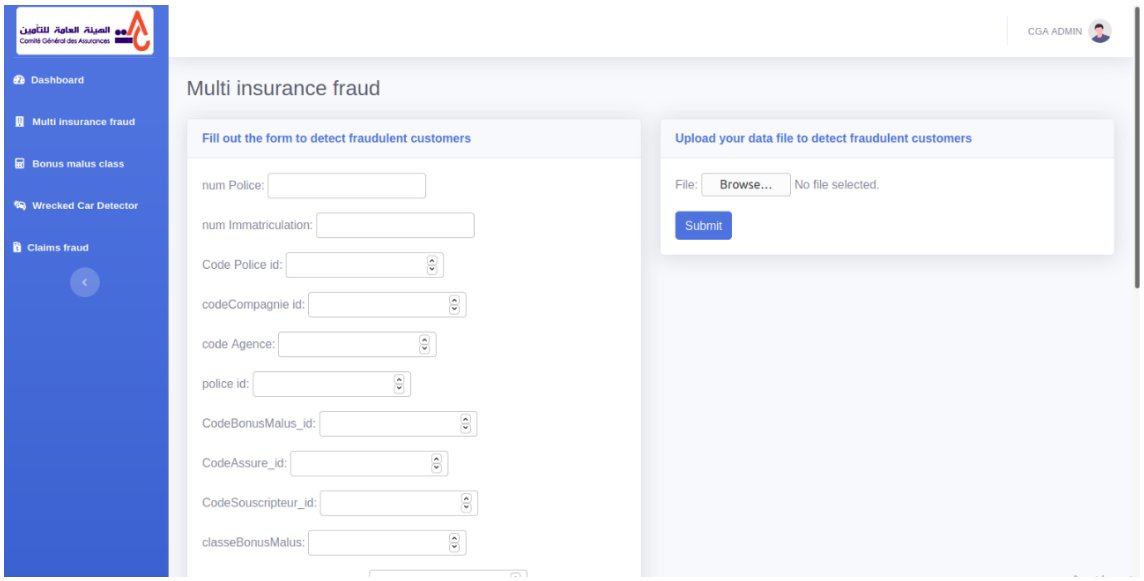


Figure 7.7: Multi-insurance fraud interface

A new web page is displayed containing a form to indicate the characteristics of the disaster that we want to classify.

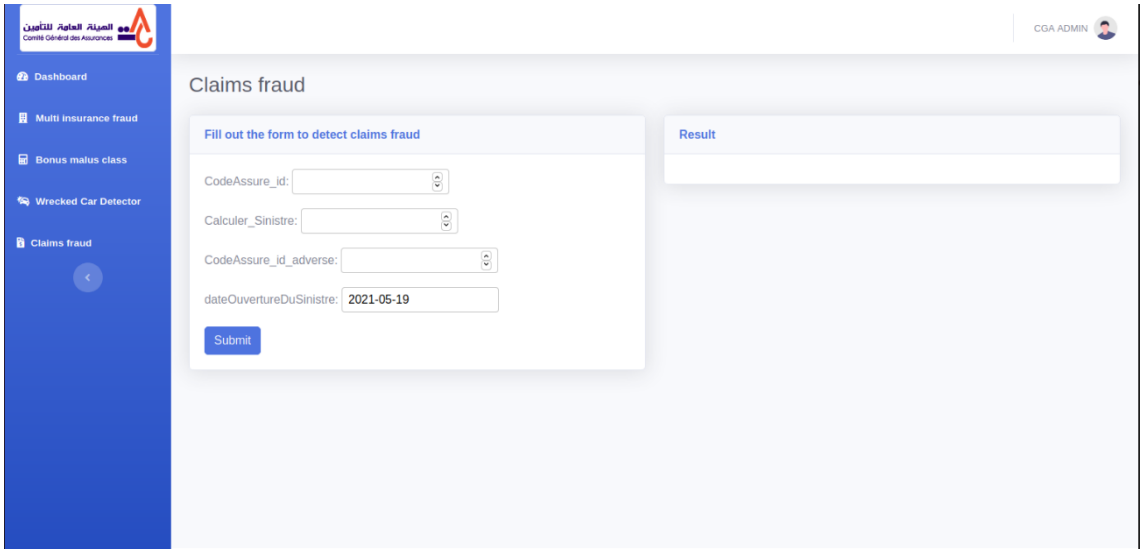


Figure 7.8: Claims fraud interface

In this page, the user can consult the data of sinister and different plots.

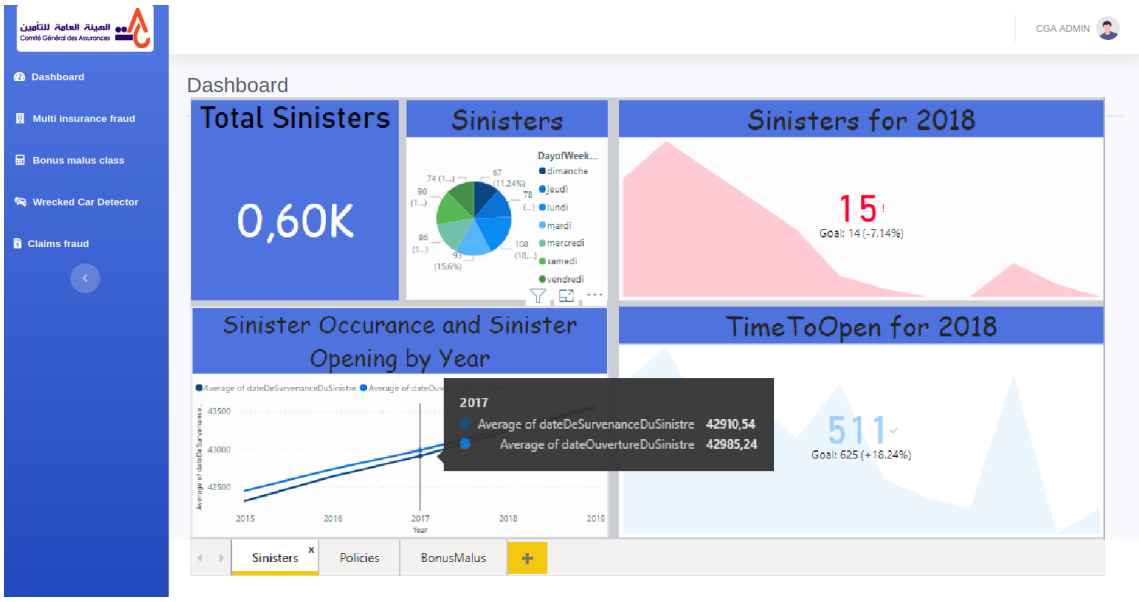


Figure 7.9: Dashboard interface

In this page, the user can consult the data of insurance policy and different plots.

Figure 7.10: Dashboard interface

Chapter 8

Conclusion and perspectives

This project aims to provide for the CGA a whole AI solution from centralizing the data in a single data warehouse, to automation in Bonus-Malus score calculation and detection of all types of frauds (multi insurance, organized sinister, damage exaggeration...) that insurance companies have suffered from for many years.
For this purpose, we used the CRISP-DM methodology as described in six major steps:

- Business Understanding:
  Every project, regardless of its size, begins with understanding the activities of the company: define the problem and identify business objectives and solution requirements from a commercial point of view.

- Data Understanding:
  After collecting the required data, We used descriptive statistics and visualization techniques that can help a data scientist understand the content of the data, to assess their quality, and to discover the initial information relating to this data.

- Data Preparation:
  The data preparation stage includes all the activities used to build the data set used in the modeling stage. It includes data cleaning, combining data from multiple sources, and data transformation. and the transformation of data into more useful variables. Feature engineering and text analysis can be used to derive new structured variables, thus enriching the set of predictions. the set of predictors and improving the accuracy of the model.

- Modeling:
  After gathering and going through the entire data preparing process, we used several models with the aim of answering the business objectives set beforehand.

- Evaluation:
  A data scientist evaluates the quality of the model and checks if it solves the problem efficiently, and in a complete and appropriate way. Compute various diagnostic metrics, as well as other results, such as tables and graphs, using the Calculate various diagnostic measures, as well as other results, such as tables and graphs, using a test set for a predictive model.

- Deployment:
  Once a satisfactory model has been developed and approved by the business sponsors, it is deployed in the production environment or in a comparable test environment. The deployment of a model in an operational business process usually involves several groups, skills, and technologies.

We faced many challenges while performing our tasks such as difficulties in collecting data. However, this only created a great opportunity for us to acquire and develop new skills. We do believe this project ended successfully delivering positive outcome.