Objectif

L'objectif de l'exercice est de proposer des différents défis liés à la mise en qualité des données et le Data visualisation pour mettre en évidence la matrice des plusieurs techniques de traitement de la data. En suite on passera à l'interprétation de l'information contenue dans cette base de données du secteur assurance. A la fin de l'exercice, on utilisera quelques concepts statistiques pour s'introduire à l'analyse économétrique.

Description des Bases de Donées

Les deux jeux de données sont disponibles en ligne sur la page freakonometrics dans le lien suivante data_ptf et data_sin.

La base data_ptf est constituée de 100,027 polices pour les années 2009 et 2010. Elle est composée de 15 variables :

PolNum: Numéro de police
CalYear: Année calendaire de souscription
Gender: Genre du conducteur
Type: Type de véhicule
Category: Catégorie du véhicule
Occupation: Profession
Age: Âge du conducteur
Group1: Groupe du véhicule
Bonus: Bonus Malus
Poldur: Ancienneté du contrat
Value: Valeur du véhicule
Adind: Indicateur d'une garantie dommages
SubGroup2: Sous-région d'habitation

La base data_sin est composée de 13 301 lignes et 4 variables :

nb_sin: Nombre de sinistres
chg_sin: Valeurs du sinistre
PolNum: Numéro de police

Import & paramétrage

1 Chargement des modules et fonctions

In [1]:
```
import os #Acceder aux répertoires. Résultat en format str
from pathlib import Path #Acceder aux répertoires. Résultat en format Path
import numpy as np # Manipulation des matrices et fonctions mathematiques
import pandas as pd # Traitemen et analyse des données, tables et series temporelles
from matplotlib import pyplot as plt # Dataviz
# from sklearn.preprocessing import StandardScaler
#**from scipy.stats import kendalltau, spearmanr, chi2_contingency, ttest_ind, bartlett, kruskal, mannwhitneyu,
#**import seaborn as sns
# from matplotlib.pylab import rcParams
# !pip install openpyxl # si la fonction read_excel necessite le package
```

```python
# rcParams['figure.figsize'] = 15, 5
```

2. Gestion du répertoire courant

In [2]:
```python
os.getcwd() # Affiche le répertoire courant de travail en format str
```

Out[2]:
```
'C:\\Users\\IDEAPAD5\\DU_Big Data\\Traitement de données'
```

In [3]:
```python
print(type(os.getcwd()))
```

```
<class 'str'>
```

In [4]:
```python
os.chdir("C:\\Users\\IDEAPAD5\\Documents\\Archivos Alejo\\Alejo\\Docs estudio y material clase\\Estudio U\\Mate
#Modifie le répertoire courant (de travail)
```

In [5]:
```python
Path.cwd() # Affiche le répertoire courant de travail en format Path (plus fonctionelle)
```

Out[5]:
```
WindowsPath('C:/Users/IDEAPAD5/Documents/Archivos Alejo/Alejo/Docs estudio y material clase/Estudio U/Material
de clases/Montpellier/DU Big Data/Traitement de données')
```

In [6]:
```python
print(type(Path.cwd()))
```

```
<class 'pathlib.WindowsPath'>
```

3 Lecture/écriture de bases de données

In [8]:
```python
base_sin = pd.read_csv("data_sin.csv", sep=";", decimal=",") # Import d'un fichier csv
```

In [9]:
```python
base_ptf = pd.read_excel("data_ptf.xlsx", sheet_name = "PTF") # Import d'un fichier excel + onglet spécifique
```

In [10]:
```python
base_expo = pd.read_excel("data_ptf.xlsx", sheet_name = "Expo") # Import d'un fichier excel + onglet spécifique
```

In [11]:
```python
base_sin.to_excel("base_sin_exporte.xlsx", sheet_name = "Sin",index=False) # Export vers une fichier Excel
```

In [12]:
```python
base_expo.to_csv("base_expo_exporte.csv",sep = ";",index=False) # Export vers une fichier csv
```

In [13]:
```python
base_ptf #Il montre qqs lignes et colonnes de la BBDD
```

Out[13]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Der
0	200114871	2009	Male	C	Small	Self-employed	27	3	-20	0	8590	0	P20	P	43.843
1	200114872	2009	Female	E	Large	Unemployed	60	20	-30	0	27445	0	NaN	L	66.066
2	200114873	2009	Female	D	Medium	Housewife	62	13	-30	9	11290	1	NaN	R	276.335
3	200114874	2009	Female	B	Large	Employed	27	16	50	3	26985	0	NaN	T	30.462
4	200114875	2009	Male	F	Large	Housewife	37	16	80	3	39705	1	NaN	R	285.621
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
100022	200285801	2010	Male	F	Medium	Housewife	45	11	30	0	19700	0	L40	L	76.052
100023	200285802	2010	Male	E	Medium	Retired	53	8	-30	6	10980	1	NaN	U	61.794
100024	200285803	2010	Male	C	Large	Employed	47	10	-10	9	21980	0	NaN	L	45.669
100025	200285804	2010	Female	D	Large	Retired	46	7	-50	1	28925	1	NaN	U	54.931
100026	200285805	2010	Female	C	Medium	Retired	67	17	-50	9	14525	1	NaN	L	73.252

100027 rows × 15 columns

In [14]:
```python
#nom_fichier=input("Veuillez renseigner le fichier à importer : ") # Pour demander de saisir le nom de fichier
```

Export Base de données

In [15]:
```python
base_sin.to_csv("base_cm.csv",sep = ";",index=False)
```

In [16]:
```python
base_expo.to_csv("base_freq.csv",sep = ";",index=False)
```

Analyse structurelle & Extractions

4. Analyse de format

In [21]:
```python
base = base_ptf.copy()
```

In [22]:
```python
base.head()
```

Out[22]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Density
0	200114871	2009	Male	C	Small	Self-employed	27	3	-20	0	8590	0	P20	P	43.843798
1	200114872	2009	Female	E	Large	Unemployed	60	20	-30	0	27445	0	NaN	L	66.066684
2	200114873	2009	Female	D	Medium	Housewife	62	13	-30	9	11290	1	NaN	R	276.335565
3	200114874	2009	Female	B	Large	Employed	27	16	50	3	26985	0	NaN	T	30.462442
4	200114875	2009	Male	F	Large	Housewife	37	16	80	3	39705	1	NaN	R	285.621744

In [23]:
```
base.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100027 entries, 0 to 100026
Data columns (total 15 columns):
 #   Column      Non-Null Count   Dtype  
---  ------      --------------   -----  
 0   PolNum      100027 non-null  int64  
 1   CalYear     100027 non-null  int64  
 2   Gender      100022 non-null  object 
 3   Type        100027 non-null  object 
 4   Category    100027 non-null  object 
 5   Occupation  100027 non-null  object 
 6   Age         100027 non-null  int64  
 7   Group1      100027 non-null  int64  
 8   Bonus       100027 non-null  int64  
 9   Poldur      100027 non-null  int64  
 10  Value       99242 non-null   object 
 11  Adind       100027 non-null  int64  
 12  SubGroup2   11598 non-null   object 
 13  Group2      100027 non-null  object 
 14  Density     100027 non-null  float64
dtypes: float64(1), int64(7), object(7)
memory usage: 11.4+ MB
```

In [24]:
```
type(base) # type df de pandas
```
Out[24]:
```
pandas.core.frame.DataFrame
```

In [25]:
```
base.shape # les dimensions
```
Out[25]:
```
(100027, 15)
```

In [26]:
```
base.size # renvoi le nombre d'éléments total
```
Out[26]:
```
1500405
```

In [27]:
```
len(base) # nb des lignes
```
Out[27]:
```
100027
```

In [28]:
```
base.columns # Nom des colonnes (variables)
```
Out[28]:
```
Index(['PolNum', 'CalYear', 'Gender', 'Type', 'Category', 'Occupation', 'Age',
       'Group1', 'Bonus', 'Poldur', 'Value', 'Adind', 'SubGroup2', 'Group2',
       'Density'],
      dtype='object')
```

In [29]:
```
len(base.columns) # nombre de columns
```
Out[29]:
```
15
```

In [30]:
```
base.index # le nom des lignes (l'index)
```
Out[30]:
```
RangeIndex(start=0, stop=100027, step=1)
```

In [31]:
```
base = base.set_index(['Category']); # définition de l'index
base
```

Out[31]:

Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Density
Small	200114871	2009	Male	C	Self-employed	27	3	-20	0	8590	0	P20	P	43.843798
Large	200114872	2009	Female	E	Unemployed	60	20	-30	0	27445	0	NaN	L	66.066684
Medium	200114873	2009	Female	D	Housewife	62	13	-30	9	11290	1	NaN	R	276.335565
Large	200114874	2009	Female	B	Employed	27	16	50	3	26985	0	NaN	T	30.462442
Large	200114875	2009	Male	F	Housewife	37	16	80	3	39705	1	NaN	R	285.621744
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Medium	200285801	2010	Male	F	Housewife	45	11	30	0	19700	0	L40	L	76.052726
Medium	200285802	2010	Male	E	Retired	53	8	-30	6	10980	1	NaN	U	61.794759
Large	200285803	2010	Male	C	Employed	47	10	-10	9	21980	0	NaN	L	45.669823
Large	200285804	2010	Female	D	Retired	46	7	-50	1	28925	1	NaN	U	54.931812
Medium	200285805	2010	Female	C	Retired	67	17	-50	9	14525	1	NaN	L	73.252499

100027 rows × 14 columns

In [32]:
```
base = base.reset_index() # reset d el'index
```

In [33]:
```
base.sort_values(by = ['PolNum','SubGroup2'],ascending=[True, True], inplace = False)
# tri selon 2 critères
# inplace: affectation permanent du fichier (Il remplace faire (base = ))
```

Out[33]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Der
0	Small	200114871	2009	Male	C	Self-employed	27	3	-20	0	8590	0	P20	P	43.84
1	Large	200114872	2009	Female	E	Unemployed	60	20	-30	0	27445	0	NaN	L	66.06
2	Medium	200114873	2009	Female	D	Housewife	62	13	-30	9	11290	1	NaN	R	276.33
3	Large	200114874	2009	Female	B	Employed	27	16	50	3	26985	0	NaN	T	30.46
4	Large	200114875	2009	Male	F	Housewife	37	16	80	3	39705	1	NaN	R	285.62
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
100022	Medium	200285801	2010	Male	F	Housewife	45	11	30	0	19700	0	L40	L	76.05
100023	Medium	200285802	2010	Male	E	Retired	53	8	-30	6	10980	1	NaN	U	61.79
100024	Large	200285803	2010	Male	C	Employed	47	10	-10	9	21980	0	NaN	L	45.66
100025	Large	200285804	2010	Female	D	Retired	46	7	-50	1	28925	1	NaN	U	54.93
100026	Medium	200285805	2010	Female	C	Retired	67	17	-50	9	14525	1	NaN	L	73.25

100027 rows × 15 columns

Typologie des variables

In [34]:
```
base.dtypes # type de chaque colonne
```

Out[34]:
```
Category       object
PolNum          int64
CalYear         int64
Gender         object
Type           object
Occupation     object
Age             int64
Group1          int64
Bonus           int64
Poldur          int64
Value          object
Adind           int64
SubGroup2      object
Group2         object
Density       float64
dtype: object
```

In [35]:
```
for col in base.columns: # boucle sur les colonnes du dataframe # parametre col créee comme pivot
    print(col + " : " + str(base[col].dtype))
```

```
Category : object
PolNum : int64
CalYear : int64
Gender : object
Type : object
Occupation : object
Age : int64
Group1 : int64
Bonus : int64
Poldur : int64
Value : object
Adind : int64
SubGroup2 : object
Group2 : object
Density : float64
```

In [36]:
```
base.count() # valeurs non vides
```

Out[36]:
```
Category      100027
PolNum        100027
CalYear       100027
Gender        100022
Type          100027
Occupation    100027
Age           100027
Group1        100027
Bonus         100027
Poldur        100027
Value          99242
Adind         100027
SubGroup2      11598
Group2        100027
Density       100027
dtype: int64
```

In [37]:
```
base.Gender.unique() #le nom des modalités dans Genre mais il dit pas le nombre des observations
#nan = données manquantes
```

Out[37]:
```
array(['Male', 'Female', 'H', 'F', 'h', nan], dtype=object)
```

In [38]:
```
base.Gender.nunique() #nombre de modalites
```

Out[38]:
```
5
```

In [39]:
```
base['Gender'].value_counts(dropna=False)
```

Out[39]:
```
Male      63437
Female    36574
H             5
F             5
NaN           5
h             1
Name: Gender, dtype: int64
```

In [40]:
```
pd.value_counts(base.Gender, dropna=False)
```

Out[40]:
```
Male      63437
Female    36574
H             5
F             5
NaN           5
h             1
Name: Gender, dtype: int64
```

In [41]:
```
pd.value_counts(base.Gender) # ignore par défaut les NA
```

Out[41]:
```
Male      63437
Female    36574
H             5
F             5
h             1
Name: Gender, dtype: int64
```

In [42]:
```
base['Gender'].isnull() # représente un vecteur de booléen
```

Out[42]:
```
0         False
1         False
2         False
3         False
4         False
          ...
100022    False
100023    False
100024    False
100025    False
100026    False
Name: Gender, Length: 100027, dtype: bool
```

In [43]:
```
base.Gender.notnull()
```

Out[43]:
```
0         True
1         True
2         True
3         True
4         True
          ...
100022    True
100023    True
100024    True
100025    True
100026    True
Name: Gender, Length: 100027, dtype: bool
```

In [44]:
```
base[base['Gender'].isnull()] # lignes où Gender est nulle
```

Out[44]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Dens
97125	Small	200282905	2010	NaN	B	Employed	24	12	50	12	26400	1	NaN	L	49.533
98313	Medium	200284093	2010	NaN	E	Housewife	20	1	20	7	9025	0	NaN	M	183.895
99117	Small	200284896	2010	NaN	A	Employed	36	8	-40	10	2020	0	NaN	L	66.945
99765	Medium	200285544	2010	NaN	A	Self-employed	20	15	10	14	14245	1	NaN	L	49.632
99910	Small	200285689	2010	NaN	D	Unemployed	29	1	20	2	17470	1	NaN	R	275.282

In [45]:
```
base.isna() # les éléments sont manquantes ou pas
```

Out[45]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Density
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
100022	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
100023	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False
100024	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False
100025	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False
100026	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False

100027 rows × 15 columns

In [46]:
```
base.Gender.notna() #l'invers = tous qui est non nulle
```

Out[46]:
```
0         True
1         True
2         True
3         True
4         True
          ...
100022    True
100023    True
100024    True
100025    True
100026    True
Name: Gender, Length: 100027, dtype: bool
```

5. filtre de la base par indiçage

In [47]:
```
base[base.PolNum==200114994] # acceès à l'info de la ligne via un valeur spécifque
```

Out[47]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Density
124	Large	200114994	2009	Male	E	Employed	20	11	30	2	22370	1	NaN	O	39.550411
125	Large	200114994	2009	Male	E	Employed	20	11	30	2	22370	1	O38	O	39.550411

In [48]:
```
base.loc[0,"PolNum"] # accès à un élément via num ligne et le nom de la colonne, même chose avec base.at
```

Out[48]:
```
200114871
```

In [49]:
```
base.iloc[0,0] # accès à un élément via num ligne et colonne rappel : [ligne,colonne] commencent à 0
# base.iat[0,0] # même chose avec iat qui renvoi les singletons
```

```
# i = Numéro ligne
```

Out[49]:
```
'Small'
```

In [50]:
```
base.iloc[:,6] # accès toutes les lignes de la colonne
```

Out[50]:
```
0         27
1         60
2         62
3         27
4         37
          ..
100022    45
100023    53
100024    47
100025    46
100026    67
Name: Age, Length: 100027, dtype: int64
```

In [51]:
```
base.loc[:,'Age'] # accès toutes les lignes de la colonne
```

Out[51]:
```
0         27
1         60
2         62
3         27
4         37
          ..
100022    45
100023    53
100024    47
100025    46
100026    67
Name: Age, Length: 100027, dtype: int64
```

In [52]:
```
base.loc[0:2,'Age'] # accès à la matrice. 1° via num ligne et puis le nom de la colonne
```

Out[52]:
```
0    27
1    60
2    62
Name: Age, dtype: int64
```

In [53]:
```
base.iloc[0:2,6] # accès à la matrice. 1° via num ligne et puis numéro de la colonne
```

Out[53]:
```
0    27
1    60
Name: Age, dtype: int64
```

In [54]:
```
base.loc[:,['Gender', 'Age']][0:2] # accès 1° à la colonne et puis aux éléments
```

Out[54]:

	Gender	Age
0	Male	27
1	Female	60

In [55]:
```
base.iloc[4] # accès à toute l'info d'une ligne
```

Out[55]:
```
Category           Large
PolNum         200114875
CalYear             2009
Gender              Male
Type                   F
Occupation     Housewife
Age                   37
Group1                16
Bonus                 80
Poldur                 3
Value              39705
Adind                  1
SubGroup2            NaN
Group2                 R
Density       285.621744
Name: 4, dtype: object
```

In [56]:
```
base[9:13] # info par ligne de i à i-1
```

Out[56]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Density
9	Large	200114880	2009	Male	B	Unemployed	25	3	40	12	48945	0	NaN	M	190.051565
10	Small	200114881	2009	Male	E	Housewife	35	2	-40	5	6595	1	NaN	Q	213.255655
11	Small	200114882	2009	Female	D	Employed	36	14	-50	2	8415	1	NaN	M	201.656907
12	Medium	200114883	2009	Male	A	Housewife	31	6	-30	0	16510	1	NaN	Q	129.315608

In [57]:
```
base.Age # accès à la colonne
```

Out[57]:
```
0         27
1         60
2         62
3         27
4         37
          ..
100022    45
100023    53
100024    47
100025    46
100026    67
Name: Age, Length: 100027, dtype: int64
```

In [58]:
```
base['Age'] # accès à la colonne
```

Out[58]:
```
0         27
1         60
2         62
3         27
4         37
          ..
100022    45
100023    53
100024    47
100025    46
100026    67
Name: Age, Length: 100027, dtype: int64
```

In [59]:
```
base[['Age']] # accès à la colonne sous la forme de ps.df
# type(base['Age']) # accès à la colonne sous la forme de pd.series
```

Out[59]:

	Age
0	27
1	60
2	62
3	27
4	37
...	...
100022	45
100023	53
100024	47
100025	46
100026	67

100027 rows × 1 columns

In [60]:
```
base[['Age','Gender']] # accès à plusieurs colonnes
```

Out[60]:

	Age	Gender
0	27	Male
1	60	Female
2	62	Female
3	27	Female
4	37	Male
...	...	...
100022	45	Male
100023	53	Male
100024	47	Male
100025	46	Female
100026	67	Female

100027 rows × 2 columns

6. Echantillonnage aléatoire de la base

In [61]:
```
base.sample(frac=0.1) # échantillonnage aleatoire 10% de la base
```

Out[61]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Dens
89141	Small	200274921	2010	Male	D	Retired	72	16	-20	4	7950	0	NaN	O	31.6097
37440	Large	200152288	2009	Male	A	Unemployed	29	7	40	0	28120	0	NaN	Q	205.4307
51858	Large	200237638	2010	Female	E	Employed	59	2	-50	11	29080	1	NaN	L	67.5645
72640	Medium	200258420	2010	Female	C	Unemployed	51	18	20	10	18580	0	NaN	Q	167.9366
23871	Large	200138720	2009	Male	A	Unemployed	21	12	10	15	25090	0	NaN	R	208.8164
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
64285	Medium	200250065	2010	Female	A	Unemployed	42	17	-50	11	10310	0	NaN	M	94.3646
69584	Large	200255364	2010	Male	F	Employed	21	9	0	7	19130	1	O28	O	29.1257
56251	Large	200242031	2010	Female	B	Self-employed	52	16	-30	3	46305	0	NaN	L	53.4320
28412	Large	200143260	2009	Male	B	Employed	51	1	60	2	49025	1	NaN	Q	123.0199
17910	Small	200132759	2009	Female	E	Housewife	42	7	120	2	1110	0	NaN	M	156.1042

10003 rows × 15 columns

7. filtre de la base par conditionnement (requêtes)

Rappel : & pour ET, | pour OU, et ~ pour la négation

a. Les assurés âgés de plus de 20 ans

In [62]:
```
base['Age']>20 # représente un vecteur de booléen
```
Out[62]:
```
0         True
1         True
2         True
3         True
4         True
          ...
100022    True
100023    True
100024    True
100025    True
100026    True
Name: Age, Length: 100027, dtype: bool
```

In [63]:
```
base[base['Age']>20] # lignes ou Age dépasse 20 ans
```
Out[63]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Der
0	Small	200114871	2009	Male	C	Self-employed	27	3	-20	0	8590	0	P20	P	43.843
1	Large	200114872	2009	Female	E	Unemployed	60	20	-30	0	27445	0	NaN	L	66.066
2	Medium	200114873	2009	Female	D	Housewife	62	13	-30	9	11290	1	NaN	R	276.335
3	Large	200114874	2009	Female	B	Employed	27	16	50	3	26985	0	NaN	T	30.462
4	Large	200114875	2009	Male	F	Housewife	37	16	80	3	39705	1	NaN	R	285.621
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
100022	Medium	200285801	2010	Male	F	Housewife	45	11	30	0	19700	0	L40	L	76.052
100023	Medium	200285802	2010	Male	E	Retired	53	8	-30	6	10980	1	NaN	U	61.794
100024	Large	200285803	2010	Male	C	Employed	47	10	-10	9	21980	0	NaN	L	45.669
100025	Large	200285804	2010	Female	D	Retired	46	7	-50	1	28925	1	NaN	U	54.931
100026	Medium	200285805	2010	Female	C	Retired	67	17	-50	9	14525	1	NaN	L	73.252

94785 rows × 15 columns

In [64]:
```
base.CalYear[base['Age']>20] #lignes de la colonne CalYear où l'Age dépasse 20 ans
```
Out[64]:
```
0         2009
1         2009
2         2009
3         2009
4         2009
          ...
100022    2010
100023    2010
100024    2010
100025    2010
100026    2010
Name: CalYear, Length: 94785, dtype: int64
```

b. Les assurés de 20 ans

In [65]:
```python
base[base.Age == 20] # lignes où Age vaut 20
```

Out[65]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Der
41	Small	200114912	2009	Male	E	Employed	20	1	-10	15	4760	0	NaN	R	291.31
54	Medium	200114925	2009	Male	C	Unemployed	20	8	-10	8	18530	0	NaN	N	25.83
121	Large	200114991	2009	Male	C	Unemployed	20	14	0	0	44665	0	NaN	L	81.90
124	Large	200114994	2009	Male	E	Employed	20	11	30	2	22370	1	NaN	O	39.55
125	Large	200114994	2009	Male	E	Employed	20	11	30	2	22370	1	O38	O	39.55
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99841	Small	200285620	2010	Male	D	Employed	20	11	0	1	8545	0	NaN	R	245.33
99948	Large	200285727	2010	Male	D	Self-employed	20	10	0	6	18325	1	NaN	R	270.46
99952	Large	200285731	2010	Male	D	Unemployed	20	11	-10	11	28420	0	NaN	U	54.93
99976	Large	200285755	2010	Female	D	Unemployed	20	2	-10	10	21280	0	NaN	M	190.05
100001	Large	200285780	2010	Female	D	Employed	20	11	0	4	28130	0	NaN	Q	141.48

1858 rows × 15 columns

c. Les assurés de moins de 18 ans ou de plus de 100 ans

In [66]:
```python
base[(base.Age<18)|(base.Age>100)]
```

Out[66]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Dens
18861	Large	200133710	2009	Male	B	Self-employed	10	7	70	10	22705	1	NaN	Q	150.223
21601	Small	200136450	2009	Female	D	Self-employed	250	12	-50	14	28145	0	NaN	U	123.969
23874	Large	200138723	2009	Male	F	Unemployed	4	4	80	3	18820	0	Q14	Q	160.343

d. Les assurés d'au moins 20 ans dont le bonus est inférieur à 50 (3 mèthodes)

In [67]:
```python
base[(base['Age'] >= 20) & (base['Bonus']< 50)]
```

Out[67]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Der
0	Small	200114871	2009	Male	C	Self-employed	27	3	-20	0	8590	0	P20	P	43.84
1	Large	200114872	2009	Female	E	Unemployed	60	20	-30	0	27445	0	NaN	L	66.06
2	Medium	200114873	2009	Female	D	Housewife	62	13	-30	9	11290	1	NaN	R	276.33
5	Medium	200114876	2009	Male	D	Employed	38	4	-40	22	18655	1	NaN	U	123.96
6	Small	200114877	2009	Female	B	Housewife	31	1	-10	14	7540	0	R34	R	276.99
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
100022	Medium	200285801	2010	Male	F	Housewife	45	11	30	0	19700	0	L40	L	76.05
100023	Medium	200285802	2010	Male	E	Retired	53	8	-30	6	10980	1	NaN	U	61.79
100024	Large	200285803	2010	Male	C	Employed	47	10	-10	9	21980	0	NaN	L	45.66
100025	Large	200285804	2010	Female	D	Retired	46	7	-50	1	28925	1	NaN	U	54.93
100026	Medium	200285805	2010	Female	C	Retired	67	17	-50	9	14525	1	NaN	L	73.25

82002 rows × 15 columns

In [68]:
```python
base[(base.Age >= 20) & (base.Bonus < 50)]
```

Out[68]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Der
0	Small	200114871	2009	Male	C	Self-employed	27	3	-20	0	8590	0	P20	P	43.84
1	Large	200114872	2009	Female	E	Unemployed	60	20	-30	0	27445	0	NaN	L	66.06
2	Medium	200114873	2009	Female	D	Housewife	62	13	-30	9	11290	1	NaN	R	276.33
5	Medium	200114876	2009	Male	D	Employed	38	4	-40	22	18655	1	NaN	U	123.96
6	Small	200114877	2009	Female	B	Housewife	31	1	-10	14	7540	0	R34	R	276.99
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
100022	Medium	200285801	2010	Male	F	Housewife	45	11	30	0	19700	0	L40	L	76.05
100023	Medium	200285802	2010	Male	E	Retired	53	8	-30	6	10980	1	NaN	U	61.79
100024	Large	200285803	2010	Male	C	Employed	47	10	-10	9	21980	0	NaN	L	45.66
100025	Large	200285804	2010	Female	D	Retired	46	7	-50	1	28925	1	NaN	U	54.93
100026	Medium	200285805	2010	Female	C	Retired	67	17	-50	9	14525	1	NaN	L	73.25

82002 rows × 15 columns

In [69]:

```
base.query('Age >= 20 and Bonus < 50') # Commande assez utile pour concaténer des conditions facultatives
```

Out[69]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Der
0	Small	200114871	2009	Male	C	Self-employed	27	3	-20	0	8590	0	P20	P	43.84
1	Large	200114872	2009	Female	E	Unemployed	60	20	-30	0	27445	0	NaN	L	66.06
2	Medium	200114873	2009	Female	D	Housewife	62	13	-30	9	11290	1	NaN	R	276.33
5	Medium	200114876	2009	Male	D	Employed	38	4	-40	22	18655	1	NaN	U	123.96
6	Small	200114877	2009	Female	B	Housewife	31	1	-10	14	7540	0	R34	R	276.99
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
100022	Medium	200285801	2010	Male	F	Housewife	45	11	30	0	19700	0	L40	L	76.05
100023	Medium	200285802	2010	Male	E	Retired	53	8	-30	6	10980	1	NaN	U	61.79
100024	Large	200285803	2010	Male	C	Employed	47	10	-10	9	21980	0	NaN	L	45.66
100025	Large	200285804	2010	Female	D	Retired	46	7	-50	1	28925	1	NaN	U	54.93
100026	Medium	200285805	2010	Female	C	Retired	67	17	-50	9	14525	1	NaN	L	73.25

82002 rows × 15 columns

e. Les assurés dont l'âge est compris dans une liste ou dans un rang

In [70]:

```
base[base['Age'].isin([20,40])] # lignes ou l'Age est 20 et 40 ans
```

Out[70]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Der
28	Medium	200114899	2009	Female	D	Employed	40	4	-50	5	17900	1	T19	T	18.23
41	Small	200114912	2009	Male	E	Employed	20	1	-10	15	4760	0	NaN	R	291.31
54	Medium	200114925	2009	Male	C	Unemployed	20	8	-10	8	18530	0	NaN	N	25.83
77	Large	200114948	2009	Male	E	Self-employed	40	13	-20	3	23645	1	NaN	L	60.07
84	Medium	200114955	2009	Male	F	Unemployed	40	14	100	3	19160	0	NaN	U	46.84
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99948	Large	200285727	2010	Male	D	Self-employed	20	10	0	6	18325	1	NaN	R	270.46
99952	Large	200285731	2010	Male	D	Unemployed	20	11	-10	11	28420	0	NaN	U	54.93
99975	Medium	200285754	2010	Male	C	Housewife	40	8	-30	4	9090	0	NaN	Q	149.99
99976	Large	200285755	2010	Female	D	Unemployed	20	2	-10	10	21280	0	NaN	M	190.05
100001	Large	200285780	2010	Female	D	Employed	20	11	0	4	28130	0	NaN	Q	141.48

4519 rows × 15 columns

In [71]:

```
base[base['Age'].isin(np.arange(18,100))] # lignes où Age est dans le rang
```

Out[71]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Der
0	Small	200114871	2009	Male	C	Self-employed	27	3	-20	0	8590	0	P20	P	43.84
1	Large	200114872	2009	Female	E	Unemployed	60	20	-30	0	27445	0	NaN	L	66.06
2	Medium	200114873	2009	Female	D	Housewife	62	13	-30	9	11290	1	NaN	R	276.33
3	Large	200114874	2009	Female	B	Employed	27	16	50	3	26985	0	NaN	T	30.46
4	Large	200114875	2009	Male	F	Housewife	37	16	80	3	39705	1	NaN	R	285.62
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
100022	Medium	200285801	2010	Male	F	Housewife	45	11	30	0	19700	0	L40	L	76.05
100023	Medium	200285802	2010	Male	E	Retired	53	8	-30	6	10980	1	NaN	U	61.79
100024	Large	200285803	2010	Male	C	Employed	47	10	-10	9	21980	0	NaN	L	45.66
100025	Large	200285804	2010	Female	D	Retired	46	7	-50	1	28925	1	NaN	U	54.93
100026	Medium	200285805	2010	Female	C	Retired	67	17	-50	9	14525	1	NaN	L	73.25

100024 rows × 15 columns

In [72]:

```
base[~ base['Age'].isin(np.arange(18,100))] # lignes ou Age n'est pas dans le rang
```

Out[72]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Dens
18861	Large	200133710	2009	Male	B	Self-employed	10	7	70	10	22705	1	NaN	Q	150.223
21601	Small	200136450	2009	Female	D	Self-employed	250	12	-50	14	28145	0	NaN	U	123.969
23874	Large	200138723	2009	Male	F	Unemployed	4	4	80	3	18820	0	Q14	Q	160.343

8. Effectuer un tri

In [73]:

```
base.sort_values(by = 'Age') #sens croissant par age
#base['Value'].rank()
```

Out[73]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Dens
23874	Large	200138723	2009	Male	F	Unemployed	4	4	80	3	18820	0	Q14	Q	160.343
18861	Large	200133710	2009	Male	B	Self-employed	10	7	70	10	22705	1	NaN	Q	150.223
73769	Small	200259549	2010	Male	D	Housewife	18	6	0	2	8260	0	NaN	Q	114.146
66697	Medium	200252477	2010	Male	F	Unemployed	18	5	0	2	18765	1	NaN	M	156.104
3734	Medium	200118584	2009	Male	E	Housewife	18	6	0	12	17235	1	NaN	N	133.288
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
72751	Small	200258531	2010	Male	D	Self-employed	75	14	-30	6	5885	1	NaN	M	103.472
32150	Small	200146998	2009	Male	B	Self-employed	75	13	-40	11	8540	1	Q29	Q	239.455
41140	Small	200155988	2009	Female	C	Housewife	75	11	0	14	8360	0	NaN	L	106.580
64872	Small	200250652	2010	Male	E	Retired	75	10	-40	0	7690	1	NaN	L	69.701
21601	Small	200136450	2009	Female	D	Self-employed	250	12	-50	14	28145	0	NaN	U	123.969

100027 rows × 15 columns

In [74]:

```
base.sort_values(by = ['Age', 'Poldur'], ascending = [True, False]) # tri selon 2 clefs
```

Out[74]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Dens
23874	Large	200138723	2009	Male	F	Unemployed	4	4	80	3	18820	0	Q14	Q	160.343
18861	Large	200133710	2009	Male	B	Self-employed	10	7	70	10	22705	1	NaN	Q	150.223
194	Large	200115056	2009	Female	B	Unemployed	18	14	0	15	47560	1	NaN	Q	95.693
1932	Small	200116782	2009	Male	B	Self-employed	18	10	0	15	6270	0	NaN	N	65.295
2488	Large	200117338	2009	Female	E	Self-employed	18	10	0	15	21490	0	NaN	Q	126.140
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
92936	Large	200278716	2010	Male	B	Housewife	75	9	-30	0	24650	1	NaN	P	28.386
94399	Small	200280179	2010	Female	B	Self-employed	75	13	-10	0	2590	1	NaN	N	93.382
95766	Small	200281546	2010	Female	B	Retired	75	6	-40	0	5955	1	NaN	Q	185.134
96163	Large	200281943	2010	Male	E	Retired	75	14	-10	0	47530	1	NaN	M	128.543
21601	Small	200136450	2009	Female	D	Self-employed	250	12	-50	14	28145	0	NaN	U	123.969

100027 rows × 15 columns

9. Créer des sous-groupes d'individus

In [75]:
```
g = base.groupby('Gender') # scission par le sexe
g.get_group('Male') # accès au sous-groupe des hommes
#g.get_group('Male').shape  # Dimension
```

Out[75]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Den
0	Small	200114871	2009	Male	C	Self-employed	27	3	-20	0	8590	0	P20	P	43.84
4	Large	200114875	2009	Male	F	Housewife	37	16	80	3	39705	1	NaN	R	285.62
5	Medium	200114876	2009	Male	D	Employed	38	4	-40	22	18655	1	NaN	U	123.96
8	Small	200114879	2009	Male	C	Self-employed	43	7	-50	4	8140	0	NaN	M	184.47
9	Large	200114880	2009	Male	B	Unemployed	25	3	40	12	48945	0	NaN	M	190.05
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
100017	Large	200285796	2010	Male	E	Self-employed	50	10	-30	3	20490	1	NaN	Q	169.78
100021	Medium	200285800	2010	Male	B	Retired	69	8	-40	11	9380	1	NaN	U	123.01
100022	Medium	200285801	2010	Male	F	Housewife	45	11	30	0	19700	0	L40	L	76.05
100023	Medium	200285802	2010	Male	E	Retired	53	8	-30	6	10980	1	NaN	U	61.79
100024	Large	200285803	2010	Male	C	Employed	47	10	-10	9	21980	0	NaN	L	45.66

63437 rows × 15 columns

10. Tableau de contingence (Tableau croissé dynamique)

In [76]:
```
base['Gender'].value_counts()
```

Out[76]:
```
Male      63437
Female    36574
H             5
F             5
h             1
Name: Gender, dtype: int64
```

In [77]:
```
base[['Gender','CalYear']].value_counts()
```

Out[77]:
```
Gender  CalYear
Male    2009       31859
        2010       31578
Female  2010       18409
        2009       18165
F       2010           5
H       2010           3
        2009           2
h       2010           1
dtype: int64
```

In [78]:
```
freqAge = pd.value_counts(base.Age) # Tableau de fréquence absolues : par ordre d'appararition
```

```
freqAge.sort_index()
# freqAge.sort_values() #sort_values tri de manière croissante
```

Out[78]:
```
4         1
10        1
18     1688
19     1694
20     1858
       ...
72      680
73      660
74      642
75      655
250       1
Name: Age, Length: 61, dtype: int64
```

In [79]:
```
pd.value_counts(base.Age, normalize=True).sort_index() # normalize : Tableau des fréquences relatives (%)
```

Out[79]:
```
4      0.000010
10     0.000010
18     0.016875
19     0.016935
20     0.018575
         ...
72     0.006798
73     0.006598
74     0.006418
75     0.006548
250    0.000010
Name: Age, Length: 61, dtype: float64
```

In [80]:
```
pd.value_counts(base.Gender) # ignore par défaut les NA
```

Out[80]:
```
Male      63437
Female    36574
H             5
F             5
h             1
Name: Gender, dtype: int64
```

In [81]:
```
pd.value_counts(base.Gender, dropna=False)
```

Out[81]:
```
Male      63437
Female    36574
H             5
F             5
NaN           5
h             1
Name: Gender, dtype: int64
```

In [82]:
```
pd.crosstab(base['Age'], [base['Gender'],base['Category']],
            values=base['Poldur'],
            aggfunc=pd.Series.mean) # via crosstab
```

Out[82]:

Gender	F			Female			H			Male			h
Category	Large	Medium	???	Large	Medium	Small	Large	Medium	???	Large	Medium	Small	Medium
Age													
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.000000	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	10.000000	NaN	NaN	NaN
18	NaN	NaN	NaN	4.972973	5.356846	5.328947	NaN	NaN	NaN	5.348837	5.242667	5.426791	NaN
19	NaN	NaN	NaN	5.327354	5.543568	5.877049	NaN	NaN	1.0	5.763889	5.107438	5.150150	NaN
20	NaN	NaN	NaN	5.166667	5.222615	5.213115	NaN	NaN	NaN	5.610429	5.206186	5.495868	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
72	11.0	NaN	NaN	6.842697	6.378788	5.746032	NaN	NaN	NaN	6.317568	6.775148	6.680556	NaN
73	NaN	NaN	NaN	6.204082	7.375000	6.666667	NaN	NaN	NaN	5.680000	6.350993	6.420732	NaN
74	NaN	NaN	NaN	6.367647	6.317460	5.984375	NaN	NaN	NaN	6.212903	7.163265	6.551724	NaN
75	NaN	NaN	NaN	6.589286	5.868852	6.372549	NaN	NaN	NaN	6.035503	6.173653	6.152318	NaN
250	NaN	NaN	NaN	NaN	NaN	14.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN

61 rows × 13 columns

In [83]:
```
base.pivot_table(index=['Age'],columns=['Gender'],
                 values=['Poldur'],
                 aggfunc=np.mean) # via pivot_table
```

Out[83]:

	Poldur				
Gender	F	Female	H	Male	h
Age					
4	NaN	NaN	NaN	3.000000	NaN
10	NaN	NaN	NaN	10.000000	NaN
18	NaN	5.224313	NaN	5.334002	NaN
19	NaN	5.590395	NaN	5.305274	NaN
20	NaN	5.201540	NaN	5.426184	NaN
...	...	...	...	...	...
72	11.0	6.385321	NaN	6.598698	NaN
73	NaN	6.800000	NaN	6.134694	NaN
74	NaN	6.225641	NaN	6.635347	NaN
75	NaN	6.261905	NaN	6.119097	NaN
250	NaN	14.000000	NaN	NaN	NaN

61 rows × 5 columns

In [84]:
```
base.groupby(['Gender', 'Age']).Poldur.mean() # duration moyenne du contrat groupée par age et sexe
```

Out[84]:
```
Gender  Age
F       34      0.000000
        36      3.000000
        37      6.000000
        43     10.000000
        72     11.000000
                  ...
Male    72      6.598698
        73      6.134694
        74      6.635347
        75      6.119097
h       23     12.000000
Name: Poldur, Length: 130, dtype: float64
```

In [85]:
```
base.groupby(['Gender','Category']).Age.mean()
```

Out[85]:
```
Gender  Category
F       Large       46.500000
        Medium      36.000000
Female  ???         42.555556
        Large       40.095667
        Medium      39.937710
        Small       39.322777
H       Large       53.000000
        Medium      36.000000
Male    ???         35.900000
        Large       42.212123
        Medium      41.822748
        Small       41.644922
h       Medium      23.000000
Name: Age, dtype: float64
```

Qualité des données : détection et traitement des anomalies

11 Gestion des doublons

In [86]:
```
base = base_ptf.copy()
```

In [87]:
```
len(base.PolNum) - base.PolNum.nunique() # compte la quantité  doublons avec PolNum comme clé primaire
```

Out[87]:
```
27
```

In [88]:
```
sum(base.duplicated(subset = "PolNum")) # somme les doublons selon la clé primaire
```

Out[88]:
```
27
```

In [89]:
```
base.duplicated() # renvoi un vecteur booléen : true à partir de la seconde occurence (doublon pur)
```

```
Out[89]: 0         False
         1         False
         2         False
         3         False
         4         False
                   ...
         100022    False
         100023    False
         100024    False
         100025    False
         100026    False
         Length: 100027, dtype: bool
```

```python
In [90]: sum(base.duplicated()) # somme les doublons purs
```

```
Out[90]: 22
```

```python
In [91]: base[base.duplicated(subset = "PolNum",keep = False)].sort_values(by = 'PolNum')
         #isole et trie les doublons
         #keep = False pour mettre en evidence les doublons
         #un boléen entre [] envoie les lignes qui sont vraies
```

Out[91]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Dens
107	200114978	2009	Male	C	Medium	Employed	25	18	90	3	15080	0	NaN	L	72.0128
108	200114978	2009	Male	C	Medium	Employed	25	18	90	3	15080	0	NaN	L	72.0128
124	200114994	2009	Male	E	Large	Employed	20	11	30	2	22370	1	NaN	O	39.5504
125	200114994	2009	Male	E	Large	Employed	20	11	30	2	22370	1	O38	O	39.5504
132	200115001	2009	Female	E	Large	Unemployed	42	11	150	0	39650	0	NaN	Q	169.5291
133	200115001	2009	Female	E	Large	Unemployed	42	11	150	0	39650	0	Q28	Q	169.5291
143	200115011	2009	Female	C	Medium	Housewife	21	5	0	0	12600	1	NaN	L	58.8946
144	200115011	2009	Female	C	Medium	Housewife	21	5	0	0	12600	1	NaN	L	58.8946
148	200115015	2009	Female	D	Small	Employed	33	12	30	10	9065	0	NaN	N	109.6318
149	200115015	2009	Female	D	Small	Employed	33	12	30	10	9065	0	NaN	N	109.6318
150	200115016	2009	Female	D	Large	Employed	26	13	40	7	27335	1	NaN	N	47.9826
151	200115016	2009	Female	D	Large	Employed	26	13	40	7	27335	1	NaN	N	47.9826
159	200115023	2009	Female	C	Small	Unemployed	20	7	80	13	7710	0	NaN	Q	77.7373
158	200115023	2009	Female	C	Small	Unemployed	20	7	80	13	7710	0	NaN	Q	77.7373
179	200115043	2009	Female	B	Medium	Employed	29	3	-20	12	8965	0	NaN	R	272.9669
180	200115043	2009	Female	B	Medium	Employed	29	3	-20	12	8965	0	R19	R	272.9669
185	200115048	2009	Male	E	Large	Unemployed	31	3	-40	10	21030	1	NaN	R	251.4328
186	200115048	2009	Male	E	Large	Unemployed	31	3	-40	10	21030	1	NaN	R	251.4328
201	200115063	2009	Male	D	Large	Employed	35	7	120	1	19995	1	NaN	Q	144.9989
202	200115063	2009	Male	D	Large	Employed	35	7	120	1	19995	1	NaN	Q	144.9989
207	200115068	2009	Male	C	Medium	Employed	27	11	30	0	18395	0	NaN	Q	166.5549
208	200115068	2009	Male	C	Medium	Employed	27	11	30	0	18395	0	NaN	Q	166.5549
210	200115070	2009	Female	A	Medium	Housewife	65	9	-30	15	11880	0	NaN	R	223.3085
211	200115070	2009	Female	A	Medium	Housewife	65	9	-30	15	11880	0	R35	R	223.3085
216	200115074	2009	Female	A	Large	Unemployed	25	9	20	2	23130	0	NaN	M	107.8170
215	200115074	2009	Female	A	Large	Unemployed	25	9	20	2	23130	0	NaN	M	107.8170
219	200115076	2009	Male	D	Medium	Unemployed	24	13	50	8	15680	0	NaN	O	32.9239
218	200115076	2009	Male	D	Medium	Unemployed	24	13	50	8	15680	0	NaN	O	32.9239
233	200115090	2009	Male	D	Large	Employed	21	11	20	0	33920	0	NaN	M	102.8188
234	200115090	2009	Male	D	Large	Employed	21	11	20	0	33920	0	NaN	M	102.8188
241	200115097	2009	Female	A	Medium	Employed	29	9	-50	0	11370	0	NaN	T	32.2125
242	200115097	2009	Female	A	Medium	Employed	29	9	-50	0	11370	0	NaN	T	32.2125
244	200115099	2009	Male	D	Large	Unemployed	22	14	-10	6	23130	0	NaN	R	285.6217
245	200115099	2009	Male	D	Large	Unemployed	22	14	-10	6	23130	0	NaN	R	285.6217
249	200115103	2009	Female	D	Medium	Unemployed	21	18	-20	5	8695	0	NaN	Q	112.4047
250	200115103	2009	Female	D	Medium	Unemployed	21	18	-20	5	8695	0	NaN	Q	112.4047
267	200115120	2009	Male	A	Medium	Self-employed	18	8	0	2	11925	0	NaN	R	297.3851
268	200115120	2009	Male	A	Medium	Self-employed	18	8	0	2	11925	0	NaN	R	297.3851
285	200115137	2009	Male	D	Small	Employed	34	6	50	3	4190	0	NaN	O	31.4912

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Dens
286	200115137	2009	Male	D	Small	Employed	34	6	50	3	4190	0	NaN	O	31.4912
319	200115169	2009	Male	B	Small	Employed	41	5	150	1	5020	0	NaN	L	62.0625
318	200115169	2009	Male	B	Small	Employed	41	5	150	31	5020	0	NaN	L	62.0625
5707	200120557	2009	Female	C	Small	Employed	18	8	0	1	5445	0	NaN	U	91.5417
5708	200120557	2009	Female	C	Small	Employed	18	8	0	1	5445	0	NaN	U	91.5417
24451	200139300	2009	Female	D	Medium	Housewife	58	6	-40	8	11360	1	NaN	Q	147.9690
24452	200139300	2009	Female	D	Medium	Housewife	58	6	-40	8	11360	1	NaN	Q	147.9690
41186	200156034	2009	Male	C	Large	Housewife	18	15	0	8	41080	0	NaN	Q	124.3443
41187	200156034	2009	Male	C	Large	Housewife	18	15	0	8	41080	0	NaN	Q	124.3443
43261	200158108	2009	Female	A	Small	Self-employed	36	13	-40	5	7625	1	NaN	Q	157.2455
43262	200158108	2009	Female	A	Small	Self-employed	36	13	-40	5	7625	1	NaN	Q	157.2455
46630	200161476	2009	Male	B	Large	Unemployed	31	2	100	9	26270	0	NaN	T	24.8949
46631	200161476	2009	Male	B	Large	Unemployed	31	2	100	9	26270	0	NaN	T	24.8949
99074	200284854	2010	Male	A	Small	Housewife	35	20	-50	3	3355	1	NaN	R	295.7970
99075	200284854	2010	Male	A	Small	Housewife	35	20	-50	3	3355	1	NaN	R	295.7970

In [92]:

```
doublons = base[base.duplicated(subset = "PolNum", keep = False)].sort_values(by = ['PolNum','SubGroup2']) # Do
doublons
```

Out[92]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Dens
107	200114978	2009	Male	C	Medium	Employed	25	18	90	3	15080	0	NaN	L	72.0128
108	200114978	2009	Male	C	Medium	Employed	25	18	90	3	15080	0	NaN	L	72.0128
125	200114994	2009	Male	E	Large	Employed	20	11	30	2	22370	1	O38	O	39.5504
124	200114994	2009	Male	E	Large	Employed	20	11	30	2	22370	1	NaN	O	39.5504
133	200115001	2009	Female	E	Large	Unemployed	42	11	150	0	39650	0	Q28	Q	169.5291
132	200115001	2009	Female	E	Large	Unemployed	42	11	150	0	39650	0	NaN	Q	169.5291
143	200115011	2009	Female	C	Medium	Housewife	21	5	0	0	12600	1	NaN	L	58.8946
144	200115011	2009	Female	C	Medium	Housewife	21	5	0	0	12600	1	NaN	L	58.8946
148	200115015	2009	Female	D	Small	Employed	33	12	30	10	9065	0	NaN	N	109.6318
149	200115015	2009	Female	D	Small	Employed	33	12	30	10	9065	0	NaN	N	109.6318
150	200115016	2009	Female	D	Large	Employed	26	13	40	7	27335	1	NaN	N	47.9826
151	200115016	2009	Female	D	Large	Employed	26	13	40	7	27335	1	NaN	N	47.9826
158	200115023	2009	Female	C	Small	Unemployed	20	7	80	13	7710	0	NaN	Q	77.7373
159	200115023	2009	Female	C	Small	Unemployed	20	7	80	13	7710	0	NaN	Q	77.7373
180	200115043	2009	Female	B	Medium	Employed	29	3	-20	12	8965	0	R19	R	272.9669
179	200115043	2009	Female	B	Medium	Employed	29	3	-20	12	8965	0	NaN	R	272.9669
185	200115048	2009	Male	E	Large	Unemployed	31	3	-40	10	21030	1	NaN	R	251.4328
186	200115048	2009	Male	E	Large	Unemployed	31	3	-40	10	21030	1	NaN	R	251.4328
201	200115063	2009	Male	D	Large	Employed	35	7	120	1	19995	1	NaN	Q	144.9989
202	200115063	2009	Male	D	Large	Employed	35	7	120	1	19995	1	NaN	Q	144.9989
207	200115068	2009	Male	C	Medium	Employed	27	11	30	0	18395	0	NaN	Q	166.5549
208	200115068	2009	Male	C	Medium	Employed	27	11	30	0	18395	0	NaN	Q	166.5549
211	200115070	2009	Female	A	Medium	Housewife	65	9	-30	15	11880	0	R35	R	223.3085
210	200115070	2009	Female	A	Medium	Housewife	65	9	-30	15	11880	0	NaN	R	223.3085
215	200115074	2009	Female	A	Large	Unemployed	25	9	20	2	23130	0	NaN	M	107.8170
216	200115074	2009	Female	A	Large	Unemployed	25	9	20	2	23130	0	NaN	M	107.8170
218	200115076	2009	Male	D	Medium	Unemployed	24	13	50	8	15680	0	NaN	O	32.9239
219	200115076	2009	Male	D	Medium	Unemployed	24	13	50	8	15680	0	NaN	O	32.9239
233	200115090	2009	Male	D	Large	Employed	21	11	20	0	33920	0	NaN	M	102.8188
234	200115090	2009	Male	D	Large	Employed	21	11	20	0	33920	0	NaN	M	102.8188
241	200115097	2009	Female	A	Medium	Employed	29	9	-50	0	11370	0	NaN	T	32.2125
242	200115097	2009	Female	A	Medium	Employed	29	9	-50	0	11370	0	NaN	T	32.2125
244	200115099	2009	Male	D	Large	Unemployed	22	14	-10	6	23130	0	NaN	R	285.6217
245	200115099	2009	Male	D	Large	Unemployed	22	14	-10	6	23130	0	NaN	R	285.6217

249	200115103	2009	Female	D	Medium	Unemployed	21	18	-20	5	8695	0	NaN	Q	112.4047
250	200115103	2009	Female	D	Medium	Unemployed	21	18	-20	5	8695	0	NaN	Q	112.4047
267	200115120	2009	Male	A	Medium	Self-employed	18	8	0	2	11925	0	NaN	R	297.3851
268	200115120	2009	Male	A	Medium	Self-employed	18	8	0	2	11925	0	NaN	R	297.3851
285	200115137	2009	Male	D	Small	Employed	34	6	50	3	4190	0	NaN	O	31.4912
286	200115137	2009	Male	D	Small	Employed	34	6	50	3	4190	0	NaN	O	31.4912
318	200115169	2009	Male	B	Small	Employed	41	5	150	31	5020	0	NaN	L	62.0625
319	200115169	2009	Male	B	Small	Employed	41	5	150	1	5020	0	NaN	L	62.0625
5707	200120557	2009	Female	C	Small	Employed	18	8	0	1	5445	0	NaN	U	91.5417
5708	200120557	2009	Female	C	Small	Employed	18	8	0	1	5445	0	NaN	U	91.5417
24451	200139300	2009	Female	D	Medium	Housewife	58	6	-40	8	11360	1	NaN	Q	147.9690
24452	200139300	2009	Female	D	Medium	Housewife	58	6	-40	8	11360	1	NaN	Q	147.9690
41186	200156034	2009	Male	C	Large	Housewife	18	15	0	8	41080	0	NaN	Q	124.3443
41187	200156034	2009	Male	C	Large	Housewife	18	15	0	8	41080	0	NaN	Q	124.3443
43261	200158108	2009	Female	A	Small	Self-employed	36	13	-40	5	7625	1	NaN	Q	157.2455
43262	200158108	2009	Female	A	Small	Self-employed	36	13	-40	5	7625	1	NaN	Q	157.2455
46630	200161476	2009	Male	B	Large	Unemployed	31	2	100	9	26270	0	NaN	T	24.8949
46631	200161476	2009	Male	B	Large	Unemployed	31	2	100	9	26270	0	NaN	T	24.8949
99074	200284854	2010	Male	A	Small	Housewife	35	20	-50	3	3355	1	NaN	R	295.7970
99075	200284854	2010	Male	A	Small	Housewife	35	20	-50	3	3355	1	NaN	R	295.7970

In [93]:

```
doublons[~doublons.duplicated(keep=False)]
#sert à identifier où se trouve la différence entre les doublons (dans ce cas SubGroup2)
```

Out[93]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Density
125	200114994	2009	Male	E	Large	Employed	20	11	30	2	22370	1	O38	O	39.550411
124	200114994	2009	Male	E	Large	Employed	20	11	30	2	22370	1	NaN	O	39.550411
133	200115001	2009	Female	E	Large	Unemployed	42	11	150	0	39650	0	Q28	Q	169.529148
132	200115001	2009	Female	E	Large	Unemployed	42	11	150	0	39650	0	NaN	Q	169.529148
180	200115043	2009	Female	B	Medium	Employed	29	3	-20	12	8965	0	R19	R	272.966995
179	200115043	2009	Female	B	Medium	Employed	29	3	-20	12	8965	0	NaN	R	272.966995
211	200115070	2009	Female	A	Medium	Housewife	65	9	-30	15	11880	0	R35	R	223.308572
210	200115070	2009	Female	A	Medium	Housewife	65	9	-30	15	11880	0	NaN	R	223.308572
318	200115169	2009	Male	B	Small	Employed	41	5	150	31	5020	0	NaN	L	62.062526
319	200115169	2009	Male	B	Small	Employed	41	5	150	1	5020	0	NaN	L	62.062526

In [94]:

```
base.sort_values(by = ['PolNum','SubGroup2'], inplace = True),
base.shape
```

Out[94]:

```
(100027, 15)
```

In [95]:

```
base.drop_duplicates(subset="PolNum", keep='first', inplace=True); # garde la première occurence de la clé prim
base.shape
```

Out[95]:

```
(100000, 15)
```

In [96]:

```
sum(base.duplicated(subset = "PolNum")) # Verifier si les doublons ont été bien supprimés
```

Out[96]:

```
0
```

In [97]:

```
base[base.PolNum==200114994] # ligne gardée
```

Out[97]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Density
125	200114994	2009	Male	E	Large	Employed	20	11	30	2	22370	1	O38	O	39.550411

Autre façon pour eliminer les doublons

In [98]:

```
base = base_ptf.copy()
```

In [99]:
```
base.sort_values(by = ['PolNum','SubGroup2','Poldur'],ascending=[True, True, False], inplace = True);
base.shape
```
Out[99]:
```
(100027, 15)
```
In [100…
```
base.drop_duplicates(subset="PolNum", keep='first', inplace=True);
base.shape
```
Out[100]:
```
(100000, 15)
```
In [101…
```
sum(base.duplicated(subset = "PolNum"))
```
Out[101]:
```
0
```
In [102…
```
base[base.PolNum==200114994] # ligne gardée
```
Out[102]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Density
125	200114994	2009	Male	E	Large	Employed	20	11	30	2	22370	1	O38	O	39.550411

In [103…
```
doublons[doublons.PolNum==200114994] # review de la base doublons pour voir le differnce
```
Out[103]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Density
125	200114994	2009	Male	E	Large	Employed	20	11	30	2	22370	1	O38	O	39.550411
124	200114994	2009	Male	E	Large	Employed	20	11	30	2	22370	1	NaN	O	39.550411

12 Gestion des DM

Détection

In [104…
```
base.count()
```
Out[104]:
```
PolNum        100000
CalYear       100000
Gender         99995
Type          100000
Category      100000
Occupation    100000
Age           100000
Group1        100000
Bonus         100000
Poldur        100000
Value          99215
Adind         100000
SubGroup2      11598
Group2        100000
Density       100000
dtype: int64
```
In [105…
```
base.isna() # Une Matrice en boléen
#base[base.isna()] # une autre façon de faire le même = Crée une liste
```
Out[105]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	Density
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
100022	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
100023	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False
100024	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False
100025	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False
100026	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False

100000 rows × 15 columns

In [106…
```
base.isna().sum() # valeurs manquentes par variable
```

Out[106]:
```
PolNum           0
CalYear          0
Gender           5
Type             0
Category         0
Occupation       0
Age              0
Group1           0
Bonus            0
Poldur           0
Value          785
Adind            0
SubGroup2    88402
Group2           0
Density          0
dtype: int64
```

In [107…
```
base.notna().sum()
```

Out[107]:
```
PolNum        100000
CalYear       100000
Gender         99995
Type          100000
Category      100000
Occupation    100000
Age           100000
Group1        100000
Bonus         100000
Poldur        100000
Value          99215
Adind         100000
SubGroup2      11598
Group2        100000
Density       100000
dtype: int64
```

Suppression

In [108…
```
base.dropna() # supprime toutes les lignes ayant un NA. Attention à la perte d'information
```

Out[108]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	SubGroup2	Group2	De
0	200114871	2009	Male	C	Small	Self-employed	27	3	-20	0	8590	0	P20	P	43.84
6	200114877	2009	Female	B	Small	Housewife	31	1	-10	14	7540	0	R34	R	276.9
17	200114888	2009	Male	C	Small	Employed	45	19	-40	2	4565	1	Q63	Q	125.9
25	200114896	2009	Female	E	Small	Employed	46	7	-50	3	2945	1	R36	R	276.3
28	200114899	2009	Female	D	Medium	Employed	40	4	-50	5	17900	1	T19	T	18.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99980	200285759	2010	Male	B	Medium	Employed	30	14	-40	11	14830	0	U10	U	137.8
99984	200285763	2010	Female	D	Small	Employed	32	9	-50	7	7060	0	R5	R	258.9
99992	200285771	2010	Male	D	Small	Unemployed	45	11	-50	1	4110	0	Q25	Q	140.4
99995	200285774	2010	Female	A	Large	Employed	22	1	-30	6	19860	0	R26	R	295.7
100022	200285801	2010	Male	F	Medium	Housewife	45	11	30	0	19700	0	L40	L	76.0

11597 rows × 15 columns

In [109…
```
base.dropna(axis=0).count() # supprime toutes les lignes avec NA, par défaut
```

Out[109]:
```
PolNum        11597
CalYear       11597
Gender        11597
Type          11597
Category      11597
Occupation    11597
Age           11597
Group1        11597
Bonus         11597
Poldur        11597
Value         11597
Adind         11597
SubGroup2     11597
Group2        11597
Density       11597
dtype: int64
```

In [110…
```
base.dropna(axis=1).count() # supprimes toutes les colonnes avec NA
```

Out[110]:
```
PolNum        100000
CalYear       100000
Type          100000
Category      100000
Occupation    100000
Age           100000
Group1        100000
Bonus         100000
Poldur        100000
Adind         100000
Group2        100000
Density       100000
dtype: int64
```

In [111…
```
base.drop('SubGroup2', axis = 1,inplace=True) # suppression d'une variable
# base.drop(columns =['SubGroup2'])
```

In [112…
```
base.count()
```

Out[112]:
```
PolNum        100000
CalYear       100000
Gender         99995
Type          100000
Category      100000
Occupation    100000
Age           100000
Group1        100000
Bonus         100000
Poldur        100000
Value          99215
Adind         100000
Group2        100000
Density       100000
dtype: int64
```

In [113…
```
base2 = base.dropna(axis=0)
base2.count()
```

Out[113]:
```
PolNum        99210
CalYear       99210
Gender        99210
Type          99210
Category      99210
Occupation    99210
Age           99210
Group1        99210
Bonus         99210
Poldur        99210
Value         99210
Adind         99210
Group2        99210
Density       99210
dtype: int64
```

In [114…
```
base2.shape
```

Out[114]:
```
(99210, 14)
```

Imputation : univariée

Imputation de variable quantitative

In [115…
```
base.head()
```

Out[115]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density
0	200114871	2009	Male	C	Small	Self-employed	27	3	-20	0	8590	0	P	43.843798
1	200114872	2009	Female	E	Large	Unemployed	60	20	-30	0	27445	0	L	66.066684
2	200114873	2009	Female	D	Medium	Housewife	62	13	-30	9	11290	1	R	276.335565
3	200114874	2009	Female	B	Large	Employed	27	16	50	3	26985	0	T	30.462442
4	200114875	2009	Male	F	Large	Housewife	37	16	80	3	39705	1	R	285.621744

In [116…
```
base.Age[1] = np.nan # Imputation par NaN dans l'index 1
base.head()
```

```
C:\Users\IDEAPAD5\AppData\Local\Temp\ipykernel_12872\1551978113.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  base.Age[1] = np.nan # Imputation par NaN dans l'index 1
```

Out[116]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density
0	200114871	2009	Male	C	Small	Self-employed	27.0	3	-20	0	8590	0	P	43.843798
1	200114872	2009	Female	E	Large	Unemployed	NaN	20	-30	0	27445	0	L	66.066684
2	200114873	2009	Female	D	Medium	Housewife	62.0	13	-30	9	11290	1	R	276.335565
3	200114874	2009	Female	B	Large	Employed	27.0	16	50	3	26985	0	T	30.462442
4	200114875	2009	Male	F	Large	Housewife	37.0	16	80	3	39705	1	R	285.621744

In [117…

```
base.count() #on remarque le donnée manquente de index 1 variable Age
```

Out[117]:

```
PolNum        100000
CalYear       100000
Gender         99995
Type          100000
Category      100000
Occupation    100000
Age            99999
Group1        100000
Bonus         100000
Poldur        100000
Value          99215
Adind         100000
Group2        100000
Density       100000
dtype: int64
```

In [118…

```
base.Age = base.Age.fillna(base.Age.median()); # imputation du NaN de la ligne 1 par la moyenne de la variable
base.head()
```

Out[118]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density
0	200114871	2009	Male	C	Small	Self-employed	27.0	3	-20	0	8590	0	P	43.843798
1	200114872	2009	Female	E	Large	Unemployed	40.0	20	-30	0	27445	0	L	66.066684
2	200114873	2009	Female	D	Medium	Housewife	62.0	13	-30	9	11290	1	R	276.335565
3	200114874	2009	Female	B	Large	Employed	27.0	16	50	3	26985	0	T	30.462442
4	200114875	2009	Male	F	Large	Housewife	37.0	16	80	3	39705	1	R	285.621744

Imputation de variable qualitative

In [119…

```
base.Gender.value_counts(dropna=False)
```

Out[119]:

```
Male      63423
Female    36561
H             5
F             5
NaN           5
h             1
Name: Gender, dtype: int64
```

In [120…

```
base.Gender.mode()
```

Out[120]:

```
0    Male
Name: Gender, dtype: object
```

In [121…

```
base.Gender = base.Gender.fillna(base.Gender.mode()[0]) ; base # imputation par la mode
```

Out[121]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density
0	200114871	2009	Male	C	Small	Self-employed	27.0	3	-20	0	8590	0	P	43.843798
1	200114872	2009	Female	E	Large	Unemployed	40.0	20	-30	0	27445	0	L	66.066684
2	200114873	2009	Female	D	Medium	Housewife	62.0	13	-30	9	11290	1	R	276.335565
3	200114874	2009	Female	B	Large	Employed	27.0	16	50	3	26985	0	T	30.462442
4	200114875	2009	Male	F	Large	Housewife	37.0	16	80	3	39705	1	R	285.621744
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
100022	200285801	2010	Male	F	Medium	Housewife	45.0	11	30	0	19700	0	L	76.052726
100023	200285802	2010	Male	E	Medium	Retired	53.0	8	-30	6	10980	1	U	61.794759
100024	200285803	2010	Male	C	Large	Employed	47.0	10	-10	9	21980	0	L	45.669823
100025	200285804	2010	Female	D	Large	Retired	46.0	7	-50	1	28925	1	U	54.931812
100026	200285805	2010	Female	C	Medium	Retired	67.0	17	-50	9	14525	1	L	73.252499

100000 rows × 14 columns

In [122…

```
base.Gender.value_counts(dropna=False)
```

Out[122]:
```
Male      63428
Female    36561
H             5
F             5
h             1
Name: Gender, dtype: int64
```

In [123…
```
base2 = base[base.Gender.notna()] # selection des lignes sans NA
base2.count()
```

Out[123]:
```
PolNum        100000
CalYear       100000
Gender        100000
Type          100000
Category      100000
Occupation    100000
Age           100000
Group1        100000
Bonus         100000
Poldur        100000
Value          99215
Adind         100000
Group2        100000
Density       100000
dtype: int64
```

Imputation : multivariée

In [124…
```
#Tcd par categorie et value car on a besoin de faire une moyenne conditionel, alors on doit confirmer s'il y a
# Ne fonctionne pas = TypeError: unsupported operand type(s) for +: 'int' and 'str'
agg_value_cat = base.groupby('Category').Value.mean() ; agg_value_cat
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
File ~\anaconda3\lib\site-packages\pandas\core\groupby\groupby.py:1578, in GroupBy._cython_agg_general.<locals>
.array_func(values)
   1577 try:
-> 1578     result = self.grouper._cython_operation(
   1579         "aggregate", values, how, axis=data.ndim - 1, min_count=min_count
   1580     )
   1581 except NotImplementedError:
   1582     # generally if we have numeric_only=False
   1583     # and non-applicable functions
   1584     # try to python agg
   1585     # TODO: shouldn't min_count matter?

File ~\anaconda3\lib\site-packages\pandas\core\groupby\ops.py:939, in BaseGrouper._cython_operation(self, kind,
values, how, axis, min_count, **kwargs)
    938 ngroups = self.ngroups
--> 939 return cy_op.cython_operation(
    940     values=values,
    941     axis=axis,
    942     min_count=min_count,
    943     comp_ids=ids,
    944     ngroups=ngroups,
    945     **kwargs,
    946 )

File ~\anaconda3\lib\site-packages\pandas\core\groupby\ops.py:626, in WrappedCythonOp.cython_operation(self, va
lues, axis, min_count, comp_ids, ngroups, **kwargs)
    618     return self._ea_wrap_cython_operation(
    619         values,
    620         min_count=min_count,
   (...)
    623         **kwargs,
    624     )
--> 626 return self._cython_op_ndim_compat(
    627     values,
    628     min_count=min_count,
    629     ngroups=ngroups,
    630     comp_ids=comp_ids,
    631     mask=None,
    632     **kwargs,
    633 )

File ~\anaconda3\lib\site-packages\pandas\core\groupby\ops.py:451, in WrappedCythonOp._cython_op_ndim_compat(se
lf, values, min_count, ngroups, comp_ids, mask, result_mask, **kwargs)
    450     result_mask = result_mask[None, :]
--> 451 res = self._call_cython_op(
    452     values2d,
    453     min_count=min_count,
    454     ngroups=ngroups,
    455     comp_ids=comp_ids,
    456     mask=mask,
    457     result_mask=result_mask,
    458     **kwargs,
    459 )
    460 if res.shape[0] == 1:
```

```
File ~\anaconda3\lib\site-packages\pandas\core\groupby\ops.py:516, in WrappedCythonOp._call_cython_op(self, val
ues, min_count, ngroups, comp_ids, mask, result_mask, **kwargs)
    515 out_shape = self._get_output_shape(ngroups, values)
--> 516 func, values = self.get_cython_func_and_vals(values, is_numeric)
    517 out_dtype = self.get_out_dtype(values.dtype)

File ~\anaconda3\lib\site-packages\pandas\core\groupby\ops.py:199, in WrappedCythonOp.get_cython_func_and_vals(
self, values, is_numeric)
    197     return func, values
--> 199 func = self._get_cython_function(kind, how, values.dtype, is_numeric)
    201 if values.dtype.kind in ["i", "u"]:

File ~\anaconda3\lib\site-packages\pandas\core\groupby\ops.py:164, in WrappedCythonOp._get_cython_function(cls,
kind, how, dtype, is_numeric)
    162 if "object" not in f.__signatures__:
    163     # raise NotImplementedError here rather than TypeError later
--> 164     raise NotImplementedError(
    165         f"function is not implemented for this dtype: "
    166         f"[how->{how},dtype->{dtype_str}]"
    167     )
    168 return f

NotImplementedError: function is not implemented for this dtype: [how->mean,dtype->object]

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
Cell In[124], line 3
      1 #Tcd par categorie et value car on a besoin de faire une moyenne conditionel, alors on doit confirmer s
'il y a une cotrrelation entre les variables
      2 # Ne fonctionne pas = TypeError: unsupported operand type(s) for +: 'int' and 'str'
----> 3 agg_value_cat = base.groupby('Category').Value.mean() ; agg_value_cat

File ~\anaconda3\lib\site-packages\pandas\core\groupby\groupby.py:1956, in GroupBy.mean(self, numeric_only, eng
ine, engine_kwargs)
   1954     return self._numba_agg_general(sliding_mean, engine_kwargs, "groupby_mean")
   1955 else:
-> 1956     result = self._cython_agg_general(
   1957         "mean",
   1958         alt=lambda x: Series(x).mean(numeric_only=numeric_only_bool),
   1959         numeric_only=numeric_only_bool,
   1960     )
   1961     return result.__finalize__(self.obj, method="groupby")

File ~\anaconda3\lib\site-packages\pandas\core\groupby\groupby.py:1592, in GroupBy._cython_agg_general(self, ho
w, alt, numeric_only, min_count)
   1588     return result
   1590 # TypeError -> we may have an exception in trying to aggregate
   1591 #  continue and exclude the block
-> 1592 new_mgr = data.grouped_reduce(array_func, ignore_failures=True)
   1594 if not is_ser and len(new_mgr) < len(data):
   1595     warn_dropping_nuisance_columns_deprecated(type(self), how)

File ~\anaconda3\lib\site-packages\pandas\core\internals\base.py:199, in SingleDataManager.grouped_reduce(self,
func, ignore_failures)
    193 """
    194 ignore_failures : bool, default False
    195     Not used; for compatibility with ArrayManager/BlockManager.
    196 """
    198 arr = self.array
--> 199 res = func(arr)
    200 index = default_index(len(res))
    202 mgr = type(self).from_array(res, index)

File ~\anaconda3\lib\site-packages\pandas\core\groupby\groupby.py:1586, in GroupBy._cython_agg_general.<locals>
.array_func(values)
   1578     result = self.grouper._cython_operation(
   1579         "aggregate", values, how, axis=data.ndim - 1, min_count=min_count
   1580     )
   1581 except NotImplementedError:
   1582     # generally if we have numeric_only=False
   1583     # and non-applicable functions
   1584     # try to python agg
   1585     # TODO: shouldn't min_count matter?
-> 1586     result = self._agg_py_fallback(values, ndim=data.ndim, alt=alt)
   1588 return result

File ~\anaconda3\lib\site-packages\pandas\core\groupby\groupby.py:1540, in GroupBy._agg_py_fallback(self, value
s, ndim, alt)
   1535     ser = df.iloc[:, 0]
   1537 # We do not get here with UDFs, so we know that our dtype
   1538 #  should always be preserved by the implemented aggregations
   1539 # TODO: Is this exactly right; see WrappedCythonOp get_result_dtype?
-> 1540 res_values = self.grouper.agg_series(ser, alt, preserve_dtype=True)
   1542 if isinstance(values, Categorical):
   1543     # Because we only get here with known dtype-preserving
   1544     #  reductions, we cast back to Categorical.
   1545     # TODO: if we ever get "rank" working, exclude it here.
```

```
   1546     res_values = type(values)._from_sequence(res_values, dtype=values.dtype)

File ~\anaconda3\lib\site-packages\pandas\core\groupby\ops.py:981, in BaseGrouper.agg_series(self, obj, func, p
reserve_dtype)
    978     preserve_dtype = True
    980 else:
--> 981     result = self._aggregate_series_pure_python(obj, func)
    983 npvalues = lib.maybe_convert_objects(result, try_float=False)
    984 if preserve_dtype:

File ~\anaconda3\lib\site-packages\pandas\core\groupby\ops.py:1005, in BaseGrouper._aggregate_series_pure_pytho
n(self, obj, func)
   1003 for i, group in enumerate(splitter):
   1004     group = group.__finalize__(obj, method="groupby")
-> 1005     res = func(group)
   1006     res = libreduction.extract_result(res)
   1008     if not initialized:
   1009         # We only do this validation on the first iteration

File ~\anaconda3\lib\site-packages\pandas\core\groupby\groupby.py:1958, in GroupBy.mean.<locals>.<lambda>(x)
   1954     return self._numba_agg_general(sliding_mean, engine_kwargs, "groupby_mean")
   1955 else:
   1956     result = self._cython_agg_general(
   1957         "mean",
-> 1958         alt=lambda x: Series(x).mean(numeric_only=numeric_only_bool),
   1959         numeric_only=numeric_only_bool,
   1960     )
   1961     return result.__finalize__(self.obj, method="groupby")

File ~\anaconda3\lib\site-packages\pandas\core\generic.py:11117, in NDFrame._add_numeric_operations.<locals>.me
an(self, axis, skipna, level, numeric_only, **kwargs)
  11099 @doc(
  11100     _num_doc,
  11101     desc="Return the mean of the values over the requested axis.",
   (...)
  11115     **kwargs,
  11116 ):
> 11117     return NDFrame.mean(self, axis, skipna, level, numeric_only, **kwargs)

File ~\anaconda3\lib\site-packages\pandas\core\generic.py:10687, in NDFrame.mean(self, axis, skipna, level, num
eric_only, **kwargs)
  10679 def mean(
  10680     self,
  10681     axis: Axis | None | lib.NoDefault = lib.no_default,
   (...)
  10685     **kwargs,
  10686 ) -> Series | float:
> 10687     return self._stat_function(
  10688         "mean", nanops.nanmean, axis, skipna, level, numeric_only, **kwargs
  10689     )

File ~\anaconda3\lib\site-packages\pandas\core\generic.py:10639, in NDFrame._stat_function(self, name, func, ax
is, skipna, level, numeric_only, **kwargs)
  10629     warnings.warn(
  10630         "Using the level keyword in DataFrame and Series aggregations is "
  10631         "deprecated and will be removed in a future version. Use groupby "
   (...)
  10634         stacklevel=find_stack_level(),
  10635     )
  10636     return self._agg_by_level(
  10637         name, axis=axis, level=level, skipna=skipna, numeric_only=numeric_only
  10638     )
> 10639 return self._reduce(
  10640     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_only
  10641 )

File ~\anaconda3\lib\site-packages\pandas\core\series.py:4471, in Series._reduce(self, op, name, axis, skipna,
numeric_only, filter_type, **kwds)
   4467     raise NotImplementedError(
   4468         f"Series.{name} does not implement {kwd_name}."
   4469     )
   4470 with np.errstate(all="ignore"):
-> 4471     return op(delegate, skipna=skipna, **kwds)

File ~\anaconda3\lib\site-packages\pandas\core\nanops.py:93, in disallow.__call__.<locals>._f(*args, **kwargs)
     91 try:
     92     with np.errstate(invalid="ignore"):
---> 93         return f(*args, **kwargs)
     94 except ValueError as e:
     95     # we want to transform an object array
     96     # ValueError message to the more typical TypeError
     97     # e.g. this is normally a disallowed function on
     98     # object arrays that contain strings
     99     if is_object_dtype(args[0]):

File ~\anaconda3\lib\site-packages\pandas\core\nanops.py:155, in bottleneck_switch.__call__.<locals>.f(values,
axis, skipna, **kwds)
    153         result = alt(values, axis=axis, skipna=skipna, **kwds)
    154 else:
```

```
--> 155     result = alt(values, axis=axis, skipna=skipna, **kwds)
    157 return result

File ~\anaconda3\lib\site-packages\pandas\core\nanops.py:410, in _datetimelike_compat.<locals>.new_func(values, axis, skipna, mask, **kwargs)
    407 if datetimelike and mask is None:
    408     mask = isna(values)
--> 410 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwargs)
    412 if datetimelike:
    413     result = _wrap_results(result, orig_values.dtype, fill_value=iNaT)

File ~\anaconda3\lib\site-packages\pandas\core\nanops.py:698, in nanmean(values, axis, skipna, mask)
    695     dtype_count = dtype
    697 count = _get_counts(values.shape, mask, axis, dtype=dtype_count)
--> 698 the_sum = _ensure_numeric(values.sum(axis, dtype=dtype_sum))
    700 if axis is not None and getattr(the_sum, "ndim", False):
    701     count = cast(np.ndarray, count)

File ~\anaconda3\lib\site-packages\numpy\core\_methods.py:49, in _sum(a, axis, dtype, out, keepdims, initial, where)
     47 def _sum(a, axis=None, dtype=None, out=None, keepdims=False,
     48          initial= NoValue, where=True):
---> 49     return umr_sum(a, axis, dtype, out, keepdims, initial, where)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

In [125…

```
base.dtypes #type de la variable
```

Out[125]:

```
PolNum          int64
CalYear         int64
Gender         object
Type           object
Category       object
Occupation     object
Age           float64
Group1          int64
Bonus           int64
Poldur          int64
Value          object
Adind           int64
Group2         object
Density       float64
dtype: object
```

In [126…

```
pd.to_numeric(base["Value"]) # corriger l'erreur "??"
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
File ~\anaconda3\lib\site-packages\pandas\_libs\lib.pyx:2315, in pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string "??"

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
Cell In[126], line 1
----> 1 pd.to_numeric(base["Value"]) # corriger l'erreur "??"

File ~\anaconda3\lib\site-packages\pandas\core\tools\numeric.py:184, in to_numeric(arg, errors, downcast)
    182 coerce_numeric = errors not in ("ignore", "raise")
    183 try:
--> 184     values, _ = lib.maybe_convert_numeric(
    185         values, set(), coerce_numeric=coerce_numeric
    186     )
    187 except (ValueError, TypeError):
    188     if errors == "raise":

File ~\anaconda3\lib\site-packages\pandas\_libs\lib.pyx:2357, in pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string "??" at position 103
```

In [127…

```
base.Value[103] # contenu de la ligne 103
```

Out[127]:

```
'??'
```

In [128…

```
base.Value.replace({'??': np.nan},inplace=True) # remplacement de ?? par nan
```

In [129…

```
base['Value_num'] = pd.to_numeric(base["Value"]) # conversion de la variable "Value" tout en creant une nouvell
#pd.to_numeric(base["Value"]) # conversion sans crér une nouvelle colonne
```

In [130…

```
agg_value_cat = base.groupby('Category').Value.mean() ; agg_value_cat # TC Catégory - Value
```

Out[130]:

```
Category
???       14392.241379
Large     28808.727041
Medium    14529.104272
Small      6202.288346
Name: Value, dtype: float64
```

In [131…
```
base[base.Value.isna()] # filtre sur les NaN de la colonne "Value"
```

Out[131]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value_r
103	200114974	2009	Female	A	Medium	Employed	18.0	7	0	4	NaN	0	U	103.949399	I
182	200115045	2009	Male	B	Large	Unemployed	49.0	10	-50	1	NaN	0	Q	122.449695	I
229	200115086	2009	Female	C	Medium	Housewife	46.0	6	120	13	NaN	1	S	34.810955	I
541	200115391	2009	Male	C	Small	Self-employed	35.0	1	30	7	NaN	1	S	25.009820	I
687	200115537	2009	Male	A	Medium	Unemployed	25.0	15	-30	7	NaN	0	R	245.331155	I
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99596	200285375	2010	Female	C	Small	Housewife	21.0	5	10	12	NaN	1	M	201.656907	I
99639	200285418	2010	Male	F	Large	Retired	57.0	2	-30	5	NaN	1	T	18.238046	I
99644	200285423	2010	Female	A	Small	Unemployed	44.0	11	-50	9	NaN	1	L	78.339935	I
99703	200285482	2010	Male	C	Large	Employed	30.0	11	130	1	NaN	0	T	28.136303	I
99986	200285765	2010	Male	C	Large	Retired	58.0	7	-10	15	NaN	1	R	210.189841	I

786 rows × 15 columns

In [132…
```
ordre = base.columns ; ordre
```

Out[132]:
```
Index(['PolNum', 'CalYear', 'Gender', 'Type', 'Category', 'Occupation', 'Age',
       'Group1', 'Bonus', 'Poldur', 'Value', 'Adind', 'Group2', 'Density',
       'Value_num'],
      dtype='object')
```

In [133…
```
base = base.set_index(['Category']) # définition de l'index
```

In [134…
```
base['Value_num'] = base['Value_num'].fillna(agg_value_cat) # Imputation DM par moyene conditionnelle
```

In [135…
```
base[base.Value.isna()]
```

Out[135]:

	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value_num
Category														
Medium	200114974	2009	Female	A	Employed	18.0	7	0	4	NaN	0	U	103.949399	14529.104272
Large	200115045	2009	Male	B	Unemployed	49.0	10	-50	1	NaN	0	Q	122.449695	28808.727041
Medium	200115086	2009	Female	C	Housewife	46.0	6	120	13	NaN	1	S	34.810955	14529.104272
Small	200115391	2009	Male	C	Self-employed	35.0	1	30	7	NaN	1	S	25.009820	6202.288346
Medium	200115537	2009	Male	A	Unemployed	25.0	15	-30	7	NaN	0	R	245.331155	14529.104272
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Small	200285375	2010	Female	C	Housewife	21.0	5	10	12	NaN	1	M	201.656907	6202.288346
Large	200285418	2010	Male	F	Retired	57.0	2	-30	5	NaN	1	T	18.238046	28808.727041
Small	200285423	2010	Female	A	Unemployed	44.0	11	-50	9	NaN	1	L	78.339935	6202.288346
Large	200285482	2010	Male	C	Employed	30.0	11	130	1	NaN	0	T	28.136303	28808.727041
Large	200285765	2010	Male	C	Retired	58.0	7	-10	15	NaN	1	R	210.189841	28808.727041

786 rows × 14 columns

In [136…
```
base = base.reset_index() # reset de l'index
```

In [137…
```
base[base.Value.isna()] # analyse des résultats. ici on vois sur le tableau que : une nouvelle colonne est crée
# si je veux recueper ma base initiale alors (base = base_ptf.copy())
```

Out[137]:

	Category	PolNum	CalYear	Gender	Type	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value
103	Medium	200114974	2009	Female	A	Employed	18.0	7	0	4	NaN	0	U	103.949399	14529.1
174	Large	200115045	2009	Male	B	Unemployed	49.0	10	-50	1	NaN	0	Q	122.449695	28808.7
215	Medium	200115086	2009	Female	C	Housewife	46.0	6	120	13	NaN	1	S	34.810955	14529.1
520	Small	200115391	2009	Male	C	Self-employed	35.0	1	30	7	NaN	1	S	25.009820	6202.2
666	Medium	200115537	2009	Male	A	Unemployed	25.0	15	-30	7	NaN	0	R	245.331155	14529.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99569	Small	200285375	2010	Female	C	Housewife	21.0	5	10	12	NaN	1	M	201.656907	6202.2
99612	Large	200285418	2010	Male	F	Retired	57.0	2	-30	5	NaN	1	T	18.238046	28808.7
99617	Small	200285423	2010	Female	A	Unemployed	44.0	11	-50	9	NaN	1	L	78.339935	6202.2
99676	Large	200285482	2010	Male	C	Employed	30.0	11	130	1	NaN	0	T	28.136303	28808.7
99959	Large	200285765	2010	Male	C	Retired	58.0	7	-10	15	NaN	1	R	210.189841	28808.7

786 rows × 15 columns

In [138…

```
base = base[ordre]; # rétablissement de l'ordre
base
```

Out[138]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value
0	200114871	2009	Male	C	Small	Self-employed	27.0	3	-20	0	8590.0	0	P	43.843798	8
1	200114872	2009	Female	E	Large	Unemployed	40.0	20	-30	0	27445.0	0	L	66.066684	27
2	200114873	2009	Female	D	Medium	Housewife	62.0	13	-30	9	11290.0	1	R	276.335565	11
3	200114874	2009	Female	B	Large	Employed	27.0	16	50	3	26985.0	0	T	30.462442	26
4	200114875	2009	Male	F	Large	Housewife	37.0	16	80	3	39705.0	1	R	285.621744	39
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99995	200285801	2010	Male	F	Medium	Housewife	45.0	11	30	0	19700.0	0	L	76.052726	19
99996	200285802	2010	Male	E	Medium	Retired	53.0	8	-30	6	10980.0	1	U	61.794759	10
99997	200285803	2010	Male	C	Large	Employed	47.0	10	-10	9	21980.0	0	L	45.669823	21
99998	200285804	2010	Female	D	Large	Retired	46.0	7	-50	1	28925.0	1	U	54.931812	28
99999	200285805	2010	Female	C	Medium	Retired	67.0	17	-50	9	14525.0	1	L	73.252499	14

100000 rows × 15 columns

In [139…

```
base.Value = base.Value_num  # remplacement de la variable Value par Value_num
base.drop("Value_num", axis=1) # supprimer la variable Value_num
```

Out[139]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density
0	200114871	2009	Male	C	Small	Self-employed	27.0	3	-20	0	8590.0	0	P	43.843798
1	200114872	2009	Female	E	Large	Unemployed	40.0	20	-30	0	27445.0	0	L	66.066684
2	200114873	2009	Female	D	Medium	Housewife	62.0	13	-30	9	11290.0	1	R	276.335565
3	200114874	2009	Female	B	Large	Employed	27.0	16	50	3	26985.0	0	T	30.462442
4	200114875	2009	Male	F	Large	Housewife	37.0	16	80	3	39705.0	1	R	285.621744
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
99995	200285801	2010	Male	F	Medium	Housewife	45.0	11	30	0	19700.0	0	L	76.052726
99996	200285802	2010	Male	E	Medium	Retired	53.0	8	-30	6	10980.0	1	U	61.794759
99997	200285803	2010	Male	C	Large	Employed	47.0	10	-10	9	21980.0	0	L	45.669823
99998	200285804	2010	Female	D	Large	Retired	46.0	7	-50	1	28925.0	1	U	54.931812
99999	200285805	2010	Female	C	Medium	Retired	67.0	17	-50	9	14525.0	1	L	73.252499

100000 rows × 14 columns

13 Gestion des DI (Données incohérents)

Univarié

Quantitative

In [140…

```
base.Age.describe()
```

Out[140]:
```
count    100000.000000
mean         41.125550
std          14.315416
min           4.000000
25%          30.000000
50%          40.000000
75%          51.000000
max         250.000000
Name: Age, dtype: float64
```

In [141…
```
base.Age.value_counts().sort_index()
```

Out[141]:
```
4.0         1
10.0        1
18.0     1685
19.0     1694
20.0     1856
         ...
72.0      680
73.0      660
74.0      642
75.0      655
250.0       1
Name: Age, Length: 61, dtype: int64
```

In [142…
```
base = base[(base.Age>=18)&(base.Age<=100)] # selection des lignes cohérentes (supression des anomalies)
```

In [143…
```
base[~(base.Age>=18)&(base.Age<=100)].Age = np.nan # imputation des incohérences par DM
```

In [144…
```
base.Age = base.Age.fillna(base.Age.median()) #imputation des DM par la médiane
```

```
C:\Users\IDEAPAD5\AppData\Local\Temp\ipykernel_12872\300790324.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  base.Age = base.Age.fillna(base.Age.median()) #imputation des DM par la médiane
```

Qualtitative

In [145…
```
base.Gender.unique()
```

Out[145]:
```
array(['Male', 'Female', 'H', 'F', 'h'], dtype=object)
```

In [146…
```
base.Gender.value_counts()
```

Out[146]:
```
Male      63426
Female    36560
H             5
F             5
h             1
Name: Gender, dtype: int64
```

In [147…
```
base.Gender.replace({'H': "Male", "F":"Female","h":"Male"},inplace=True) # Imputation des modalités
```

```
C:\Users\IDEAPAD5\AppData\Local\Temp\ipykernel_12872\2534945844.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  base.Gender.replace({'H': "Male", "F":"Female","h":"Male"},inplace=True) # Imputation des modalités
```

In [148…
```
base = base[base.Gender.isin(["Male","Female"])] # Selection des lignes
```

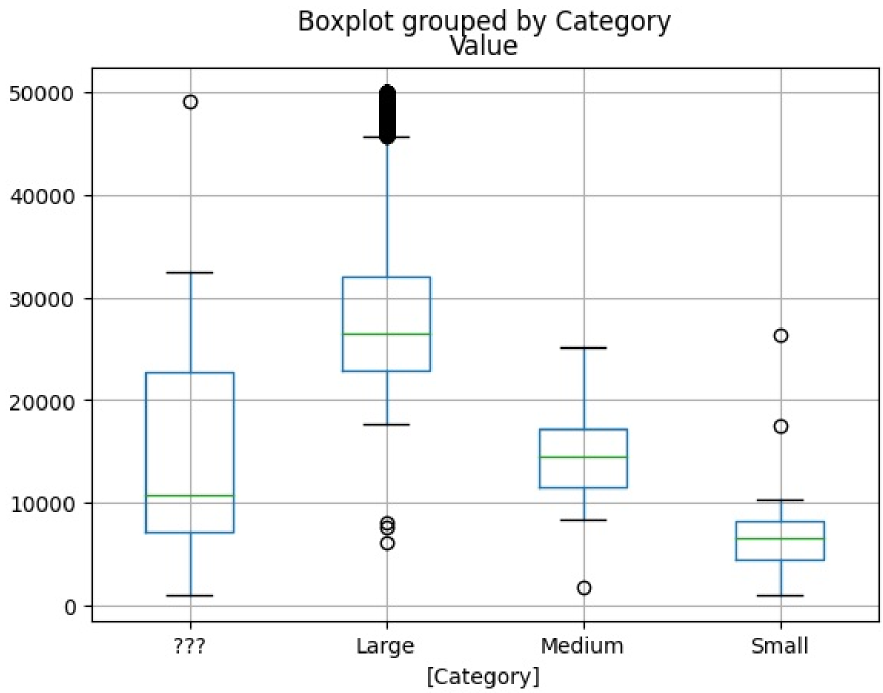
In [149…
```
base.Gender.value_counts()
```

Out[149]:
```
Male      63432
Female    36565
Name: Gender, dtype: int64
```

Imputation aléatoire

In [150…
```
base.boxplot(['Value'], by = ['Category'])
```

Out[150]:
```
<AxesSubplot:title={'center':'Value'}, xlabel='[Category]'>
```



In [151…
```
modCat = base.Category.value_counts(normalize = True) ; modCat
#Frequence relative ou proportion du catégorie
```

Out[151]:
```
Medium    0.362461
Large     0.320110
Small     0.317140
???       0.000290
Name: Category, dtype: float64
```

In [152…
```
modCat.index
```

Out[152]:
```
Index(['Medium', 'Large', 'Small', '???'], dtype='object')
```

In [153…
```
modCat = modCat[modCat.index != "???"] ; modCat
#Retirer les lignes avec ??? du dataframe pour ne pas tenir en compte dans le tirage aléatoire
```

Out[153]:
```
Medium    0.362461
Large     0.320110
Small     0.317140
Name: Category, dtype: float64
```

In [154…
```
sum(modCat)
```

Out[154]:
```
0.9997099912997389
```

In [155…
```
modCat = modCat/sum(modCat);modCat
#Frequence relative ou proportion du catégorie
```

Out[155]:
```
Medium    0.362566
Large     0.320202
Small     0.317232
Name: Category, dtype: float64
```

In [156…
```
modCat.index.to_numpy()
```

Out[156]:
```
array(['Medium', 'Large', 'Small'], dtype=object)
```

In [157…
```
#np.random.choice(base.Category.unique(), size=None, replace=True, p=None)[0=ligne]
np.random.choice(modCat.index, size =1, p = modCat)[0]
```

Out[157]:
```
'Medium'
```

In [158…
```
base0 = base.copy()
```

In [159…
```
base['Category'] = base['Category'].apply(lambda x:
                                          np.random.choice(modCat.index.to_numpy(), size = 1, p = modCat)[0] if
```

```
#.apply: fonction qui sert à appliquer une opération soit sur les lignes, soit sur les colonnes
# x= catégorie, après la fonction du tirage, qu'on lance uniquement quand x=??? (fonction if)
```

In [160…

```
base.Category.value_counts()
```

Out[160]:

```
Medium    36251
Large     32016
Small     31730
Name: Category, dtype: int64
```

In [161…

```
base0.Category.value_counts()
```

Out[161]:

```
Medium    36245
Large     32010
Small     31713
???          29
Name: Category, dtype: int64
```

Multivarié

Analyse de cohérence entre Age et PolDur (Poldur: ancienneté du contrat)

In [162…

```
for col in base:
    print(base[col].describe()); # Statistiques de chaque colonne
```

```
count    9.999700e+04
mean     2.002003e+08
std      6.216658e+04
min      2.001149e+08
25%      2.001399e+08
50%      2.002358e+08
75%      2.002608e+08
max      2.002858e+08
Name: PolNum, dtype: float64
count    99997.000000
mean      2009.500015
std          0.500002
min       2009.000000
25%       2009.000000
50%       2010.000000
75%       2010.000000
max       2010.000000
Name: CalYear, dtype: float64
count     99997
unique        2
top        Male
freq      63432
Name: Gender, dtype: object
count     99997
unique        6
top           A
freq      27756
Name: Type, dtype: object
count      99997
unique         3
top       Medium
freq       36251
Name: Category, dtype: object
count        99997
unique           5
top       Employed
freq         31139
Name: Occupation, dtype: object
count    99997.000000
mean        41.124144
std         14.299563
min         18.000000
25%         30.000000
50%         40.000000
75%         51.000000
max         75.000000
Name: Age, dtype: float64
count    99997.000000
mean        10.692931
std          4.687380
min          1.000000
25%          7.000000
50%         11.000000
75%         14.000000
max         20.000000
Name: Group1, dtype: float64
count    99997.000000
mean        -6.931208
std         48.627095
min        -50.000000
25%        -40.000000
50%        -30.000000
75%         10.000000
```

```
max        150.000000
Name: Bonus, dtype: float64
count    99997.000000
mean         5.473324
std          4.600138
min         -9.000000
25%          1.000000
50%          4.000000
75%          9.000000
max         55.000000
Name: Poldur, dtype: float64
count    99997.000000
mean     16459.288098
std      10496.820938
min       1000.000000
25%       8370.000000
50%      14570.000000
75%      22600.000000
max      49995.000000
Name: Value, dtype: float64
count    99997.000000
mean         0.512205
std          0.499854
min          0.000000
25%          0.000000
50%          1.000000
75%          1.000000
max          1.000000
Name: Adind, dtype: float64
count      99997
unique        10
top            L
freq       23730
Name: Group2, dtype: object
count    99997.000000
mean       117.156050
std         79.499992
min         14.377142
25%         50.625783
50%         94.364623
75%        174.644525
max        297.385170
Name: Density, dtype: float64
count    99997.000000
mean     16459.288098
std      10496.820938
min       1000.000000
25%       8370.000000
50%      14570.000000
75%      22600.000000
max      49995.000000
Name: Value_num, dtype: float64
```

In [163…

```
verif = base.Age - base.Poldur
#Cohérence si => 18
```

In [164…

```
verif.value_counts().sort_index()
```

Out[164]:

```
-10.0      1
-4.0       1
-1.0       1
 3.0      57
 4.0      93
        ...
 71.0    266
 72.0    235
 73.0    166
 74.0     99
 75.0     56
Length: 76, dtype: int64
```

In [165…

```
verif.describe()
```

Out[165]:

```
count    99997.000000
mean        35.650820
std         14.799473
min        -10.000000
25%         24.000000
50%         34.000000
75%         46.000000
max         75.000000
dtype: float64
```

In [166…

```
verif<18
```

Out[166]:
```
0        False
1        False
2        False
3        False
4        False
         ...  
99995    False
99996    False
99997    False
99998    False
99999    False
Length: 99997, dtype: bool
```

In [167…
```
verif[verif<18] #represente les lignes avec l'incohérence
```

Out[167]:
```
5        16.0
6        17.0
9        13.0
41        5.0
54       12.0
         ... 
99949    10.0
99950    11.0
99968    16.0
99974    16.0
99975    15.0
Length: 10485, dtype: float64
```

In [168…
```
sum(verif<18) #nombre d'incoherences
```

Out[168]:
```
10485
```

In [169…
```
seuil = 0 #Seuil d'identification d'anomalies
```

In [170…
```
sum(verif <= seuil) #sum(verif < 0 )
```

Out[170]:
```
3
```

In [171…
```
base[verif < 0] #liste des anomalies où < seuil
```

Out[171]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value
927	200115798	2009	Female	D	Medium	Employed	25.0	15	40	35	12355.0	1	U	112.803438	12
28545	200143416	2009	Male	D	Medium	Self-employed	49.0	7	-50	53	9500.0	0	L	79.159448	9
83190	200268996	2010	Male	E	Medium	Employed	32.0	18	50	33	18125.0	0	R	250.841326	18

In [172…
```
base[verif > seuil] #observations cohérents
```

Out[172]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value
0	200114871	2009	Male	C	Small	Self-employed	27.0	3	-20	0	8590.0	0	P	43.843798	8
1	200114872	2009	Female	E	Large	Unemployed	40.0	20	-30	0	27445.0	0	L	66.066684	27
2	200114873	2009	Female	D	Medium	Housewife	62.0	13	-30	9	11290.0	1	R	276.335565	11
3	200114874	2009	Female	B	Large	Employed	27.0	16	50	3	26985.0	0	T	30.462442	26
4	200114875	2009	Male	F	Large	Housewife	37.0	16	80	3	39705.0	1	R	285.621744	39
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99995	200285801	2010	Male	F	Medium	Housewife	45.0	11	30	0	19700.0	0	L	76.052726	19
99996	200285802	2010	Male	E	Medium	Retired	53.0	8	-30	6	10980.0	1	U	61.794759	10
99997	200285803	2010	Male	C	Large	Employed	47.0	10	-10	9	21980.0	0	L	45.669823	21
99998	200285804	2010	Female	D	Large	Retired	46.0	7	-50	1	28925.0	1	U	54.931812	28
99999	200285805	2010	Female	C	Medium	Retired	67.0	17	-50	9	14525.0	1	L	73.252499	14

99994 rows × 15 columns

In [173…
```
base.Poldur[verif>=0].mean() #ancianité des contratas moyenne pour les bones lignes
#base[verif > seuil].Poldur.mean()
```

Out[173]:
```
5.472278336700202
```

Fonction testant age < anc et : imputant (poldur) la moyenne observée

In [174…
```
def cor_poldur(age,anc):
    if (age < anc):
```

```
        res = base.Poldur[verif>=0].mean() #moyenne de bonnes lignes
    else :
        res = anc
    return res;

cor_poldur(20,25) #on test la fontion
```

Out[174]:

```
5.472278336700202
```

In [175…

```
base.Poldur = base.apply(lambda x:  cor_poldur(x['Age'],x['Poldur']),axis=1)
#On applique l'opération cor_poldur sur la colonne des variable définies
```

In [176…

```
base[verif<0] # ou seuil # On constate que les anomalies ont été remplacés par la moyenne du Poldur
```

Out[176]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Valu
927	200115798	2009	Female	D	Medium	Employed	25.0	15	40	5.472278	12355.0	1	U	112.803438	
28545	200143416	2009	Male	D	Medium	Self-employed	49.0	7	-50	5.472278	9500.0	0	L	79.159448	
83190	200268996	2010	Male	E	Medium	Employed	32.0	18	50	5.472278	18125.0	0	R	250.841326	

Concaténation et jointure

14 Concaténation

In [178…

```
base_expo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100021 entries, 0 to 100020
Data columns (total 2 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   PolNum   100021 non-null  int64
 1   Expdays  100021 non-null  int64
dtypes: int64(2)
memory usage: 1.5 MB
```

In [179…

```
pd.concat([base_expo, base_expo]) #Concatenation horizontale
```

Out[179]:

	PolNum	Expdays
0	200114978	365
1	200114994	365
2	200115001	365
3	200115011	365
4	200115015	365
...	...	...
100016	200285801	365
100017	200285802	365
100018	200285803	365
100019	200285804	365
100020	200285805	365

200042 rows × 2 columns

In [180…

```
pd.concat([base_expo, base_expo], axis = 1) #Concatenation horizontale
```

Out[180]:

	PolNum	Expdays	PolNum	Expdays
0	200114978	365	200114978	365
1	200114994	365	200114994	365
2	200115001	365	200115001	365
3	200115011	365	200115011	365
4	200115015	365	200115015	365
...	...	...	...	...
100016	200285801	365	200285801	365
100017	200285802	365	200285802	365
100018	200285803	365	200285803	365
100019	200285804	365	200285804	365
100020	200285805	365	200285805	365

100021 rows × 4 columns

15 Jointures

In [181…

```
base.shape
```

Out[181]:

```
(99997, 15)
```

In [182…

```
base_expo.shape
```

Out[182]:

```
(100021, 2)
```

In [183…

```
sum(base_expo.duplicated())
```

Out[183]:

```
21
```

In [184…

```
# Nb des Doublons à partir PolNum
len(base_expo.PolNum) - base_expo.PolNum.nunique()
```

Out[184]:

```
21
```

In [185…

```
# Supprimer des Doublons
base_expo2 = base_expo[~base_expo.duplicated()]
base_expo2.shape
```

Out[185]:

```
(100000, 2)
```

In [187…

```
# Concat
pd.merge(base, base_expo2, on = ['PolNum'])
```

Out[187]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value
0	200114871	2009	Male	C	Small	Self-employed	27.0	3	-20	0.0	8590.0	0	P	43.843798	8
1	200114872	2009	Female	E	Large	Unemployed	40.0	20	-30	0.0	27445.0	0	L	66.066684	27
2	200114873	2009	Female	D	Medium	Housewife	62.0	13	-30	9.0	11290.0	1	R	276.335565	11
3	200114874	2009	Female	B	Large	Employed	27.0	16	50	3.0	26985.0	0	T	30.462442	26
4	200114875	2009	Male	F	Large	Housewife	37.0	16	80	3.0	39705.0	1	R	285.621744	39
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99992	200285801	2010	Male	F	Medium	Housewife	45.0	11	30	0.0	19700.0	0	L	76.052726	19
99993	200285802	2010	Male	E	Medium	Retired	53.0	8	-30	6.0	10980.0	1	U	61.794759	10
99994	200285803	2010	Male	C	Large	Employed	47.0	10	-10	9.0	21980.0	0	L	45.669823	21
99995	200285804	2010	Female	D	Large	Retired	46.0	7	-50	1.0	28925.0	1	U	54.931812	28
99996	200285805	2010	Female	C	Medium	Retired	67.0	17	-50	9.0	14525.0	1	L	73.252499	14

99997 rows × 16 columns

In [188…

```
base_v2 =  pd.merge(base, base_expo2, on = ['PolNum'],how = 'inner')
base_v2
```

Out[188]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value
0	200114871	2009	Male	C	Small	Self-employed	27.0	3	-20	0.0	8590.0	0	P	43.843798	8
1	200114872	2009	Female	E	Large	Unemployed	40.0	20	-30	0.0	27445.0	0	L	66.066684	27
2	200114873	2009	Female	D	Medium	Housewife	62.0	13	-30	9.0	11290.0	1	R	276.335565	11
3	200114874	2009	Female	B	Large	Employed	27.0	16	50	3.0	26985.0	0	T	30.462442	26
4	200114875	2009	Male	F	Large	Housewife	37.0	16	80	3.0	39705.0	1	R	285.621744	39
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99992	200285801	2010	Male	F	Medium	Housewife	45.0	11	30	0.0	19700.0	0	L	76.052726	19
99993	200285802	2010	Male	E	Medium	Retired	53.0	8	-30	6.0	10980.0	1	U	61.794759	10
99994	200285803	2010	Male	C	Large	Employed	47.0	10	-10	9.0	21980.0	0	L	45.669823	21
99995	200285804	2010	Female	D	Large	Retired	46.0	7	-50	1.0	28925.0	1	U	54.931812	28
99996	200285805	2010	Female	C	Medium	Retired	67.0	17	-50	9.0	14525.0	1	L	73.252499	14

99997 rows × 16 columns

In [189…

```
base_sin.shape
```

Out[189]: (13300, 3)

In [190…

```
len(base_sin) - base_sin.PolNum.nunique()
```

Out[190]: 99

In [191…

```
base_sin[base_sin.duplicated('PolNum',keep=False)].sort_values('PolNum',ascending=True)
```

Out[191]:

	nb_sin	chg_sin	PolNum
0	1	0.00	200114978
35	1	362.62	200114978
1	1	0.00	200114994
36	1	495.59	200114994
2	2	0.00	200115001
...	...	...	...
12527	2	7051.78	200294761
13152	1	6086.84	200295166
12346	1	5100.52	200295166
12914	3	434.13	200295421
12928	3	7802.52	200295421

197 rows × 3 columns

In [192…

```
base_sin_v2 = base_sin.groupby(['PolNum']).sum()
base_sin_v2
```

Out[192]:

PolNum	nb_sin	chg_sin
200114878	1	740.30
200114880	1	207.32
200114890	1	803.30
200114894	1	867.68
200114895	2	1745.50
...	...	...
200295737	3	1389.54
200295742	3	2546.39
200295749	3	4808.93
200295750	2	3688.13
200295769	2	4284.23

13201 rows × 2 columns

In [193…

```
# Joint "Outer"
```

```
base_tot = pd.merge(base_v2,base_sin_v2,on = "PolNum", how = 'outer') ; base_tot
```

Out[193]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Valu
0	200114871	2009.0	Male	C	Small	Self-employed	27.0	3.0	-20.0	0.0	8590.0	0.0	P	43.843798	
1	200114872	2009.0	Female	E	Large	Unemployed	40.0	20.0	-30.0	0.0	27445.0	0.0	L	66.066684	2
2	200114873	2009.0	Female	D	Medium	Housewife	62.0	13.0	-30.0	9.0	11290.0	1.0	R	276.335565	1
3	200114874	2009.0	Female	B	Large	Employed	27.0	16.0	50.0	3.0	26985.0	0.0	T	30.462442	2
4	200114875	2009.0	Male	F	Large	Housewife	37.0	16.0	80.0	3.0	39705.0	1.0	R	285.621744	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
100938	200295737	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
100939	200295742	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
100940	200295749	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
100941	200295750	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
100942	200295769	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

100943 rows × 18 columns

In [194…

```
ano_sin = base_tot[base_tot.CalYear.isna()] ; ano_sin
```

Out[194]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value_nur
99997	200136450	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
99998	200285809	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
99999	200285812	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
100000	200285815	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
100001	200285824	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
100938	200295737	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
100939	200295742	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
100940	200295749	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
100941	200295750	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
100942	200295769	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal

946 rows × 18 columns

In [195…

```
# New df sinistrés
cnt_sin = base_tot[~base_tot.CalYear.isna() & ~base_tot.chg_sin.isna()] ; cnt_sin
```

Out[195]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value
7	200114878	2009.0	Female	A	Large	Housewife	41.0	10.0	100.0	0.0	24940.0	1.0	R	272.966995	24
9	200114880	2009.0	Male	B	Large	Unemployed	25.0	3.0	40.0	12.0	48945.0	0.0	M	190.051565	48
19	200114890	2009.0	Female	F	Small	Employed	29.0	17.0	30.0	7.0	1525.0	0.0	R	225.043089	1
23	200114894	2009.0	Male	A	Medium	Self-employed	47.0	17.0	20.0	12.0	18480.0	1.0	M	129.419475	18
24	200114895	2009.0	Female	C	Small	Employed	47.0	11.0	-10.0	7.0	8690.0	0.0	R	290.132719	8
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99928	200285737	2010.0	Male	E	Medium	Unemployed	25.0	9.0	60.0	1.0	10265.0	0.0	U	94.657516	10
99936	200285745	2010.0	Male	A	Large	Housewife	54.0	10.0	30.0	1.0	21610.0	0.0	R	250.841326	21
99947	200285756	2010.0	Male	A	Small	Employed	22.0	2.0	-10.0	11.0	6910.0	1.0	R	295.797092	6
99960	200285769	2010.0	Male	A	Medium	Employed	51.0	13.0	-30.0	0.0	11955.0	1.0	P	24.826528	11
99982	200285791	2010.0	Male	D	Medium	Self-employed	21.0	15.0	50.0	1.0	12100.0	1.0	R	259.004060	12

12255 rows × 18 columns

In [196…

```
# Joitn "Inner"
pd.merge(base_v2,base_sin_v2,on = "PolNum", how = 'inner')
```

Out[196]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value
0	200114878	2009	Female	A	Large	Housewife	41.0	10	100	0.0	24940.0	1	R	272.966995	24
1	200114880	2009	Male	B	Large	Unemployed	25.0	3	40	12.0	48945.0	0	M	190.051565	48
2	200114890	2009	Female	F	Small	Employed	29.0	17	30	7.0	1525.0	0	R	225.043089	1
3	200114894	2009	Male	A	Medium	Self-employed	47.0	17	20	12.0	18480.0	1	M	129.419475	18
4	200114895	2009	Female	C	Small	Employed	47.0	11	-10	7.0	8690.0	0	R	290.132719	8
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
12250	200285737	2010	Male	E	Medium	Unemployed	25.0	9	60	1.0	10265.0	0	U	94.657516	10
12251	200285745	2010	Male	A	Large	Housewife	54.0	10	30	1.0	21610.0	0	R	250.841326	21
12252	200285756	2010	Male	A	Small	Employed	22.0	2	-10	11.0	6910.0	1	R	295.797092	6
12253	200285769	2010	Male	A	Medium	Employed	51.0	13	-30	0.0	11955.0	1	P	24.826528	11
12254	200285791	2010	Male	D	Medium	Self-employed	21.0	15	50	1.0	12100.0	1	R	259.004060	12

12255 rows × 18 columns

In [197…

```
# New df non sinistrés
cnt_nonsin = base_tot[~base_tot.CalYear.isna() & base_tot.chg_sin.isna()] ; cnt_nonsin
```

Out[197]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value
0	200114871	2009.0	Male	C	Small	Self-employed	27.0	3.0	-20.0	0.0	8590.0	0.0	P	43.843798	8
1	200114872	2009.0	Female	E	Large	Unemployed	40.0	20.0	-30.0	0.0	27445.0	0.0	L	66.066684	27
2	200114873	2009.0	Female	D	Medium	Housewife	62.0	13.0	-30.0	9.0	11290.0	1.0	R	276.335565	11
3	200114874	2009.0	Female	B	Large	Employed	27.0	16.0	50.0	3.0	26985.0	0.0	T	30.462442	26
4	200114875	2009.0	Male	F	Large	Housewife	37.0	16.0	80.0	3.0	39705.0	1.0	R	285.621744	39
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99992	200285801	2010.0	Male	F	Medium	Housewife	45.0	11.0	30.0	0.0	19700.0	0.0	L	76.052726	19
99993	200285802	2010.0	Male	E	Medium	Retired	53.0	8.0	-30.0	6.0	10980.0	1.0	U	61.794759	10
99994	200285803	2010.0	Male	C	Large	Employed	47.0	10.0	-10.0	9.0	21980.0	0.0	L	45.669823	21
99995	200285804	2010.0	Female	D	Large	Retired	46.0	7.0	-50.0	1.0	28925.0	1.0	U	54.931812	28
99996	200285805	2010.0	Female	C	Medium	Retired	67.0	17.0	-50.0	9.0	14525.0	1.0	L	73.252499	14

87742 rows × 18 columns

In [198…

```
len(cnt_nonsin) + len(cnt_sin)
```

Out[198]: 99997

In [199…

```
len(base_sin)
```

Out[199]: 13300

In [200…

```
# Joitn "Inner"
base_freq = pd.merge(base_v2,base_sin_v2,on = "PolNum", how = 'left')  ; base_freq
```

Out[200]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value
0	200114871	2009	Male	C	Small	Self-employed	27.0	3	-20	0.0	8590.0	0	P	43.843798	8
1	200114872	2009	Female	E	Large	Unemployed	40.0	20	-30	0.0	27445.0	0	L	66.066684	27
2	200114873	2009	Female	D	Medium	Housewife	62.0	13	-30	9.0	11290.0	1	R	276.335565	11
3	200114874	2009	Female	B	Large	Employed	27.0	16	50	3.0	26985.0	0	T	30.462442	26
4	200114875	2009	Male	F	Large	Housewife	37.0	16	80	3.0	39705.0	1	R	285.621744	39
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99992	200285801	2010	Male	F	Medium	Housewife	45.0	11	30	0.0	19700.0	0	L	76.052726	19
99993	200285802	2010	Male	E	Medium	Retired	53.0	8	-30	6.0	10980.0	1	U	61.794759	10
99994	200285803	2010	Male	C	Large	Employed	47.0	10	-10	9.0	21980.0	0	L	45.669823	21
99995	200285804	2010	Female	D	Large	Retired	46.0	7	-50	1.0	28925.0	1	U	54.931812	28
99996	200285805	2010	Female	C	Medium	Retired	67.0	17	-50	9.0	14525.0	1	L	73.252499	14

99997 rows × 18 columns

In [201…

```
base_freq.fillna(0, inplace=True) ; base_freq
```

Out[201]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value
0	200114871	2009	Male	C	Small	Self-employed	27.0	3	-20	0.0	8590.0	0	P	43.843798	8
1	200114872	2009	Female	E	Large	Unemployed	40.0	20	-30	0.0	27445.0	0	L	66.066684	27
2	200114873	2009	Female	D	Medium	Housewife	62.0	13	-30	9.0	11290.0	1	R	276.335565	11
3	200114874	2009	Female	B	Large	Employed	27.0	16	50	3.0	26985.0	0	T	30.462442	26
4	200114875	2009	Male	F	Large	Housewife	37.0	16	80	3.0	39705.0	1	R	285.621744	39
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99992	200285801	2010	Male	F	Medium	Housewife	45.0	11	30	0.0	19700.0	0	L	76.052726	19
99993	200285802	2010	Male	E	Medium	Retired	53.0	8	-30	6.0	10980.0	1	U	61.794759	10
99994	200285803	2010	Male	C	Large	Employed	47.0	10	-10	9.0	21980.0	0	L	45.669823	21
99995	200285804	2010	Female	D	Large	Retired	46.0	7	-50	1.0	28925.0	1	U	54.931812	28
99996	200285805	2010	Female	C	Medium	Retired	67.0	17	-50	9.0	14525.0	1	L	73.252499	14

99997 rows × 18 columns

In [202…

```
base_cm = pd.merge(base_v2,base_sin,on = "PolNum", how = 'inner')  ; base_cm
```

Out[202]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value
0	200114878	2009	Female	A	Large	Housewife	41.0	10	100	0.0	24940.0	1	R	272.966995	24
1	200114880	2009	Male	B	Large	Unemployed	25.0	3	40	12.0	48945.0	0	M	190.051565	48
2	200114890	2009	Female	F	Small	Employed	29.0	17	30	7.0	1525.0	0	R	225.043089	1
3	200114894	2009	Male	A	Medium	Self-employed	47.0	17	20	12.0	18480.0	1	M	129.419475	18
4	200114895	2009	Female	C	Small	Employed	47.0	11	-10	7.0	8690.0	0	R	290.132719	8
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
12283	200285756	2010	Male	A	Small	Employed	22.0	2	-10	11.0	6910.0	1	R	295.797092	6
12284	200285769	2010	Male	A	Medium	Employed	51.0	13	-30	0.0	11955.0	1	P	24.826528	11
12285	200285769	2010	Male	A	Medium	Employed	51.0	13	-30	0.0	11955.0	1	P	24.826528	11
12286	200285791	2010	Male	D	Medium	Self-employed	21.0	15	50	1.0	12100.0	1	R	259.004060	12
12287	200285791	2010	Male	D	Medium	Self-employed	21.0	15	50	1.0	12100.0	1	R	259.004060	12

12288 rows × 18 columns

In [203…

```
base_freq.to_csv("base_freq.csv",sep = ";",index=False)
```

In [204…

```
base_cm.to_csv("base_cm.csv",sep = ";",index=False)
```

In [205…

```
base_cm = pd.read_csv("base_cm.csv", sep=";", decimal=".") ; base_cm.head()
```

Out[205]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value_nur
0	200114878	2009	Female	A	Large	Housewife	41.0	10	100	0.0	24940.0	1	R	272.966995	24940.
1	200114880	2009	Male	B	Large	Unemployed	25.0	3	40	12.0	48945.0	0	M	190.051565	48945.
2	200114890	2009	Female	F	Small	Employed	29.0	17	30	7.0	1525.0	0	R	225.043089	1525.
3	200114894	2009	Male	A	Medium	Self-employed	47.0	17	20	12.0	18480.0	1	M	129.419475	18480.
4	200114895	2009	Female	C	Small	Employed	47.0	11	-10	7.0	8690.0	0	R	290.132719	8690.

In [206…

```
#base_freq = pd.read_csv("base_freq.csv", sep=";", decimal=".")  ; base_freq.head()
base= pd.read_csv("base_freq.csv", sep=";", decimal=".")  ; base.head()
```

Out[206]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Value_nur
0	200114871	2009	Male	C	Small	Self-employed	27.0	3	-20	0.0	8590.0	0	P	43.843798	8590.
1	200114872	2009	Female	E	Large	Unemployed	40.0	20	-30	0.0	27445.0	0	L	66.066684	27445.
2	200114873	2009	Female	D	Medium	Housewife	62.0	13	-30	9.0	11290.0	1	R	276.335565	11290.
3	200114874	2009	Female	B	Large	Employed	27.0	16	50	3.0	26985.0	0	T	30.462442	26985.
4	200114875	2009	Male	F	Large	Housewife	37.0	16	80	3.0	39705.0	1	R	285.621744	39705.

Analyse Descriptive

In [207…

```
criteres_TC=['Age', 'Gender', 'Type', 'Category', 'Occupation', 'Group1', 'Bonus', 'Poldur', 'Value', 'Group2']
for var in criteres_TC:
    print(round(pd.crosstab(base.nb_sin, base[var], normalize='columns'),3))
```

```
Age      18.0   19.0   20.0   21.0   22.0   23.0   24.0   25.0   26.0   27.0  \
nb_sin
0.0     0.720  0.748  0.738  0.758  0.760  0.777  0.782  0.789  0.809  0.806
1.0     0.216  0.198  0.194  0.179  0.188  0.177  0.169  0.162  0.149  0.153
2.0     0.051  0.043  0.051  0.048  0.038  0.036  0.037  0.037  0.032  0.031
3.0     0.011  0.008  0.013  0.012  0.010  0.008  0.009  0.009  0.008  0.006
4.0     0.001  0.002  0.002  0.002  0.003  0.003  0.002  0.001  0.002  0.003
5.0     0.000  0.000  0.001  0.000  0.000  0.000  0.000  0.000  0.000  0.001
6.0     0.001  0.001  0.002  0.001  0.001  0.000  0.000  0.000  0.000  0.000
7.0     0.000  0.000  0.000  0.001  0.000  0.000  0.000  0.001  0.000  0.000

Age     ...   66.0   67.0   68.0   69.0   70.0   71.0   72.0   73.0   74.0  \
nb_sin  ...
0.0     ...  0.953  0.940  0.941  0.949  0.957  0.950  0.938  0.924  0.927
1.0     ...  0.044  0.057  0.056  0.044  0.042  0.047  0.057  0.067  0.069
2.0     ...  0.001  0.003  0.003  0.007  0.001  0.003  0.004  0.009  0.003
3.0     ...  0.001  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.002
4.0     ...  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
5.0     ...  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
6.0     ...  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
7.0     ...  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000

Age      75.0
nb_sin
0.0     0.940
1.0     0.049
2.0     0.009
3.0     0.002
4.0     0.000
5.0     0.000
6.0     0.000
7.0     0.000

[8 rows x 58 columns]
Gender  Female   Male
nb_sin
0.0      0.893  0.868
1.0      0.093  0.109
2.0      0.011  0.018
3.0      0.002  0.004
4.0      0.000  0.001
5.0      0.000  0.000
6.0      0.000  0.000
7.0      0.000  0.000
Type        A      B      C      D      E      F
nb_sin
0.0     0.892  0.885  0.876  0.867  0.859  0.851
1.0     0.093  0.099  0.106  0.109  0.113  0.123
2.0     0.012  0.014  0.015  0.018  0.021  0.021
3.0     0.002  0.002  0.003  0.004  0.005  0.003
4.0     0.001  0.000  0.000  0.001  0.001  0.001
```

```
5.0     0.000  0.000  0.000  0.000  0.000  0.000
6.0     0.000  0.000  0.000  0.000  0.000  0.000
7.0     0.000  0.000  0.000  0.000  0.000  0.000
Category  Large  Medium  Small
nb_sin
0.0       0.869   0.879  0.885
1.0       0.110   0.102  0.098
2.0       0.016   0.016  0.014
3.0       0.004   0.002  0.003
4.0       0.001   0.001  0.001
5.0       0.000   0.000  0.000
6.0       0.000   0.000  0.000
7.0       0.000   0.000  0.000
Occupation  Employed  Housewife  Retired  Self-employed  Unemployed
nb_sin
0.0            0.862      0.854    0.965          0.898       0.837
1.0            0.116      0.121    0.033          0.089       0.133
2.0            0.018      0.019    0.002          0.012       0.022
3.0            0.003      0.004    0.000          0.001       0.006
4.0            0.001      0.001    0.000          0.001       0.001
5.0            0.000      0.000    0.000          0.000       0.000
6.0            0.000      0.000    0.000          0.000       0.000
7.0            0.000      0.000    0.000          0.000       0.000
Group1     1      2      3      4      5      6      7      8      9      10  \
nb_sin
0.0     0.929  0.928  0.919  0.916  0.915  0.907  0.896  0.891  0.891  0.883
1.0     0.067  0.063  0.071  0.077  0.077  0.082  0.092  0.096  0.094  0.100
2.0     0.005  0.008  0.009  0.005  0.007  0.010  0.011  0.010  0.013  0.013
3.0     0.000  0.001  0.000  0.001  0.001  0.001  0.001  0.002  0.001  0.003
4.0     0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.001  0.000
5.0     0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
6.0     0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
7.0     0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000

Group1     11     12     13     14     15     16     17     18     19     20
nb_sin
0.0     0.871  0.873  0.867  0.864  0.856  0.844  0.835  0.825  0.820  0.826
1.0     0.107  0.107  0.110  0.112  0.117  0.125  0.137  0.141  0.139  0.135
2.0     0.018  0.015  0.018  0.018  0.022  0.024  0.020  0.025  0.031  0.028
3.0     0.003  0.004  0.004  0.004  0.004  0.006  0.007  0.005  0.007  0.008
4.0     0.001  0.001  0.001  0.001  0.001  0.001  0.001  0.001  0.003  0.002
5.0     0.000  0.000  0.000  0.001  0.000  0.000  0.000  0.001  0.000  0.000
6.0     0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.001  0.001  0.001
7.0     0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.001  0.000  0.000
Bonus    -50    -40    -30    -20    -10     0      10     20     30     40   \
nb_sin
0.0     0.943  0.940  0.935  0.900  0.861  0.814  0.850  0.831  0.811  0.789
1.0     0.054  0.057  0.061  0.091  0.119  0.151  0.129  0.137  0.157  0.168
2.0     0.003  0.003  0.004  0.008  0.017  0.028  0.018  0.025  0.028  0.034
3.0     0.000  0.000  0.000  0.001  0.002  0.006  0.003  0.005  0.003  0.007
4.0     0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.001  0.001  0.002
5.0     0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
6.0     0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
7.0     0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000

Bonus   ...    60     70     80     90     100    110    120    130    140  \
nb_sin  ...
0.0     ...  0.784  0.769  0.763  0.760  0.741  0.722  0.715  0.691  0.659
1.0     ...  0.167  0.178  0.178  0.196  0.215  0.199  0.219  0.247  0.244
2.0     ...  0.032  0.040  0.042  0.033  0.033  0.060  0.051  0.050  0.070
3.0     ...  0.012  0.010  0.013  0.009  0.008  0.016  0.013  0.010  0.016
4.0     ...  0.003  0.001  0.003  0.003  0.002  0.003  0.002  0.002  0.005
5.0     ...  0.001  0.001  0.001  0.000  0.001  0.000  0.000  0.000  0.004
6.0     ...  0.001  0.001  0.001  0.000  0.000  0.000  0.000  0.000  0.000
7.0     ...  0.000  0.000  0.000  0.000  0.000  0.001  0.000  0.000  0.001

Bonus     150
nb_sin
0.0     0.619
1.0     0.260
2.0     0.073
3.0     0.027
4.0     0.010
5.0     0.007
6.0     0.001
7.0     0.003

[8 rows x 21 columns]
Poldur  -9.000000   -6.000000    0.000000    1.000000    2.000000   \
nb_sin
0.0            1.0         0.0       0.849       0.870       0.874
1.0            0.0         1.0       0.122       0.109       0.104
2.0            0.0         0.0       0.022       0.015       0.019
3.0            0.0         0.0       0.005       0.003       0.003
4.0            0.0         0.0       0.001       0.001       0.000
5.0            0.0         0.0       0.000       0.000       0.000
6.0            0.0         0.0       0.000       0.000       0.000
7.0            0.0         0.0       0.000       0.000       0.000
```

```
Poldur   3.000000    4.000000    5.000000    5.472278    6.000000   ...  \
nb_sin                                                              ...   
0.0          0.872       0.877       0.881         1.0       0.879  ...   
1.0          0.110       0.102       0.101         0.0       0.100  ...   
2.0          0.014       0.017       0.014         0.0       0.016  ...   
3.0          0.003       0.003       0.003         0.0       0.004  ...   
4.0          0.001       0.001       0.001         0.0       0.001  ...   
5.0          0.000       0.000       0.000         0.0       0.000  ...   
6.0          0.000       0.000       0.000         0.0       0.000  ...   
7.0          0.000       0.000       0.000         0.0       0.000  ...   

Poldur   10.000000   11.000000   12.000000   13.000000   14.000000  \
nb_sin                                                               
0.0          0.895       0.901       0.887       0.889       0.908   
1.0          0.089       0.088       0.098       0.099       0.079   
2.0          0.013       0.009       0.013       0.010       0.011   
3.0          0.003       0.002       0.001       0.001       0.002   
4.0          0.001       0.000       0.001       0.001       0.000   
5.0          0.000       0.000       0.000       0.000       0.000   
6.0          0.000       0.000       0.000       0.000       0.000   
7.0          0.000       0.000       0.000       0.000       0.000   

Poldur   15.000000   20.000000   22.000000   31.000000   55.000000  
nb_sin                                                              
0.0          0.893         0.0         1.0         0.5         1.0  
1.0          0.095         1.0         0.0         0.0         0.0  
2.0          0.010         0.0         0.0         0.5         0.0  
3.0          0.002         0.0         0.0         0.0         0.0  
4.0          0.001         0.0         0.0         0.0         0.0  
5.0          0.000         0.0         0.0         0.0         0.0  
6.0          0.000         0.0         0.0         0.0         0.0  
7.0          0.000         0.0         0.0         0.0         0.0  

[8 rows x 23 columns]
Value   1000.0   1005.0   1010.0   1015.0   1020.0   1025.0   1030.0   \
nb_sin                                                                  
0.0        1.0      0.8     0.75     0.75    0.867      1.0      1.0   
1.0        0.0      0.2     0.25     0.25    0.133      0.0      0.0   
2.0        0.0      0.0     0.00     0.00    0.000      0.0      0.0   
3.0        0.0      0.0     0.00     0.00    0.000      0.0      0.0   
4.0        0.0      0.0     0.00     0.00    0.000      0.0      0.0   
5.0        0.0      0.0     0.00     0.00    0.000      0.0      0.0   
6.0        0.0      0.0     0.00     0.00    0.000      0.0      0.0   
7.0        0.0      0.0     0.00     0.00    0.000      0.0      0.0   

Value   1035.0   1040.0   1045.0   ...  49940.0  49950.0  49955.0  49960.0  \
nb_sin                             ...                                       
0.0       0.812    0.846    0.857  ...      1.0    0.667    0.857      1.0   
1.0       0.125    0.154    0.143  ...      0.0    0.333    0.143      0.0   
2.0       0.062    0.000    0.000  ...      0.0    0.000    0.000      0.0   
3.0       0.000    0.000    0.000  ...      0.0    0.000    0.000      0.0   
4.0       0.000    0.000    0.000  ...      0.0    0.000    0.000      0.0   
5.0       0.000    0.000    0.000  ...      0.0    0.000    0.000      0.0   
6.0       0.000    0.000    0.000  ...      0.0    0.000    0.000      0.0   
7.0       0.000    0.000    0.000  ...      0.0    0.000    0.000      0.0   

Value   49970.0  49975.0  49980.0  49985.0  49990.0  49995.0  
nb_sin                                                        
0.0         1.0    0.667      1.0      1.0      0.6      0.5  
1.0         0.0    0.333      0.0      0.0      0.4      0.5  
2.0         0.0    0.000      0.0      0.0      0.0      0.0  
3.0         0.0    0.000      0.0      0.0      0.0      0.0  
4.0         0.0    0.000      0.0      0.0      0.0      0.0  
5.0         0.0    0.000      0.0      0.0      0.0      0.0  
6.0         0.0    0.000      0.0      0.0      0.0      0.0  
7.0         0.0    0.000      0.0      0.0      0.0      0.0  

[8 rows x 9385 columns]
Group2      L      M      N      O      P      Q      R      S      T      U
nb_sin                                                                      
0.0     0.903  0.833  0.866  0.910  0.906  0.867  0.823  0.922  0.915  0.895
1.0     0.086  0.136  0.111  0.077  0.083  0.111  0.138  0.069  0.076  0.095
2.0     0.009  0.024  0.016  0.011  0.008  0.017  0.028  0.008  0.009  0.009
3.0     0.001  0.006  0.004  0.001  0.002  0.003  0.007  0.000  0.001  0.002
4.0     0.000  0.001  0.002  0.000  0.000  0.001  0.002  0.000  0.000  0.000
5.0     0.000  0.000  0.000  0.000  0.000  0.000  0.001  0.000  0.000  0.000
6.0     0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
7.0     0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
```

In [208…

```
base.describe(include='all')  # stat variables (qualitatives et quantitatives)
```

Out[208]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	
count	9.999700e+04	99997.000000	99997	99997	99997	99997	99997.000000	99997.000000	99997.000000	99997.000000	99997.
unique	NaN	NaN	2	6	3	5	NaN	NaN	NaN	NaN	
top	NaN	NaN	Male	A	Medium	Employed	NaN	NaN	NaN	NaN	
freq	NaN	NaN	63432	27756	36251	31139	NaN	NaN	NaN	NaN	
mean	2.002003e+08	2009.500015	NaN	NaN	NaN	NaN	41.124144	10.692931	-6.931208	5.472278	16459.
std	6.216658e+04	0.500002	NaN	NaN	NaN	NaN	14.299563	4.687380	48.627095	4.595909	10496.
min	2.001149e+08	2009.000000	NaN	NaN	NaN	NaN	18.000000	1.000000	-50.000000	-9.000000	1000.
25%	2.001399e+08	2009.000000	NaN	NaN	NaN	NaN	30.000000	7.000000	-40.000000	1.000000	8370.
50%	2.002358e+08	2010.000000	NaN	NaN	NaN	NaN	40.000000	11.000000	-30.000000	4.000000	14570.
75%	2.002608e+08	2010.000000	NaN	NaN	NaN	NaN	51.000000	14.000000	10.000000	9.000000	22600.
max	2.002858e+08	2010.000000	NaN	NaN	NaN	NaN	75.000000	20.000000	150.000000	55.000000	49995.

In [209…

```
base.mean()
```

```
C:\Users\IDEAPAD5\AppData\Local\Temp\ipykernel_12872\3227421367.py:1: FutureWarning: Dropping of nuisance colum
ns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeEr
ror.  Select only valid columns before calling the reduction.
  base.mean()
```

Out[209]:

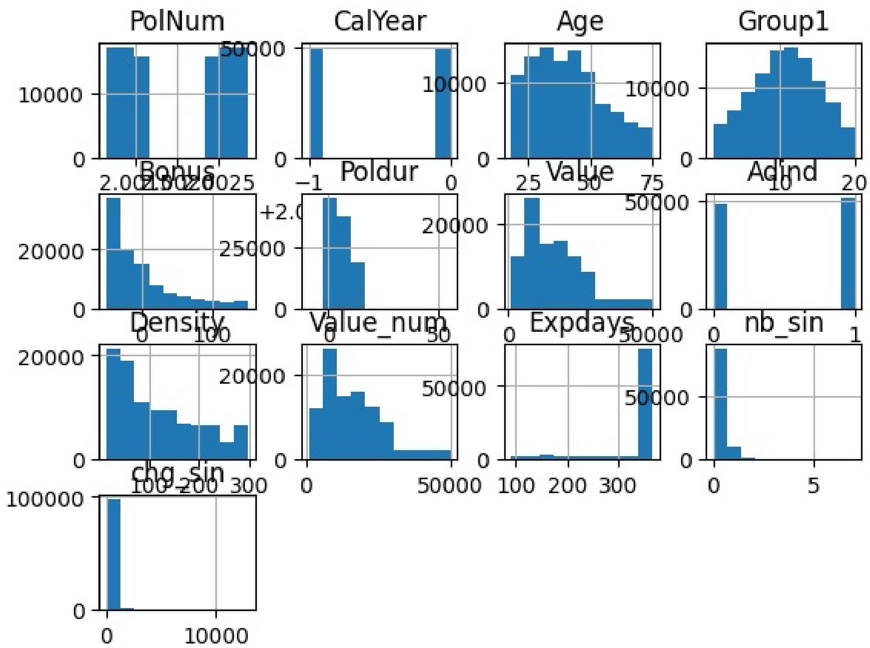
```
PolNum       2.002003e+08
CalYear      2.009500e+03
Age          4.112414e+01
Group1       1.069293e+01
Bonus       -6.931208e+00
Poldur       5.472278e+00
Value        1.645929e+04
Adind        5.122054e-01
Density      1.171560e+02
Value_num    1.645929e+04
Expdays      3.275814e+02
nb_sin       1.476544e-01
chg_sin      1.063249e+02
dtype: float64
```

In [210…

```
# Histo variables quant
base.hist()
```

Out[210]:

```
array([[<AxesSubplot:title={'center':'PolNum'}>,
        <AxesSubplot:title={'center':'CalYear'}>,
        <AxesSubplot:title={'center':'Age'}>,
        <AxesSubplot:title={'center':'Group1'}>],
       [<AxesSubplot:title={'center':'Bonus'}>,
        <AxesSubplot:title={'center':'Poldur'}>,
        <AxesSubplot:title={'center':'Value'}>,
        <AxesSubplot:title={'center':'Adind'}>],
       [<AxesSubplot:title={'center':'Density'}>,
        <AxesSubplot:title={'center':'Value_num'}>,
        <AxesSubplot:title={'center':'Expdays'}>,
        <AxesSubplot:title={'center':'nb_sin'}>],
       [<AxesSubplot:title={'center':'chg_sin'}>, <AxesSubplot:>,
        <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```



Au premier regard de la distribution on pourrait repérer qu’au niveau de l’âge notre population es asymétrique. Cela montre que le gros de nos observations possèdent moins de 50 ans (41 en moyenne).

Par rapport à la valeur du véhicule, on peut noter que la plupart du portefeuille es composé de véhicules pas chères.

Le Group1 « type de véhicule » montre une distribution plus normale, cela signifie une population plus homogène.

Pour analyser la chg_sin il nous faut isoler les valeur différentes à 0 car cela affecte directement les statistiques.

Analyse de la variable « Age »

In [211…
```
round(base.Age.mean(),2) # remplacer mean par min, max, std, var, median, sum, prod, idxmin, idxmax
```

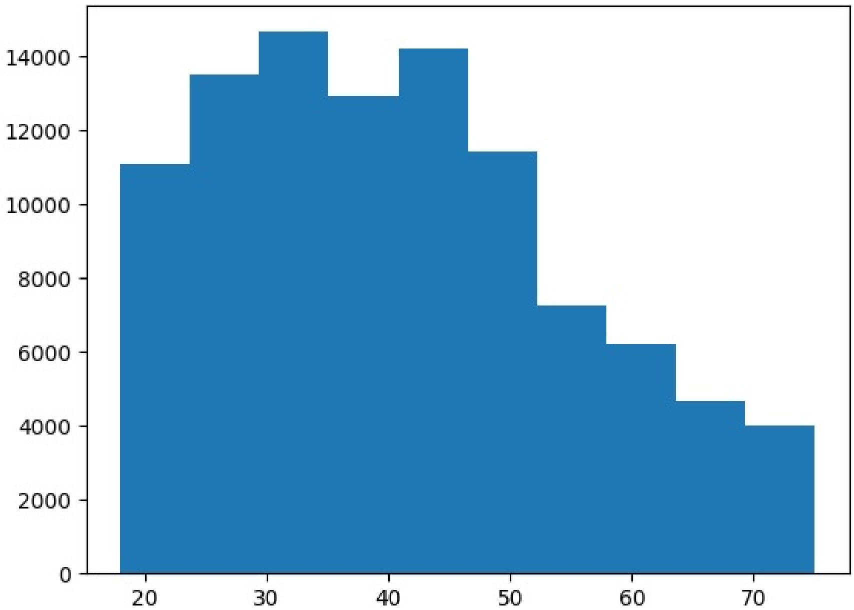
Out[211]:
```
41.12
```

In [212…
```
base.Age.describe()
```

Out[212]:
```
count    99997.000000
mean        41.124144
std         14.299563
min         18.000000
25%         30.000000
50%         40.000000
75%         51.000000
max         75.000000
Name: Age, dtype: float64
```

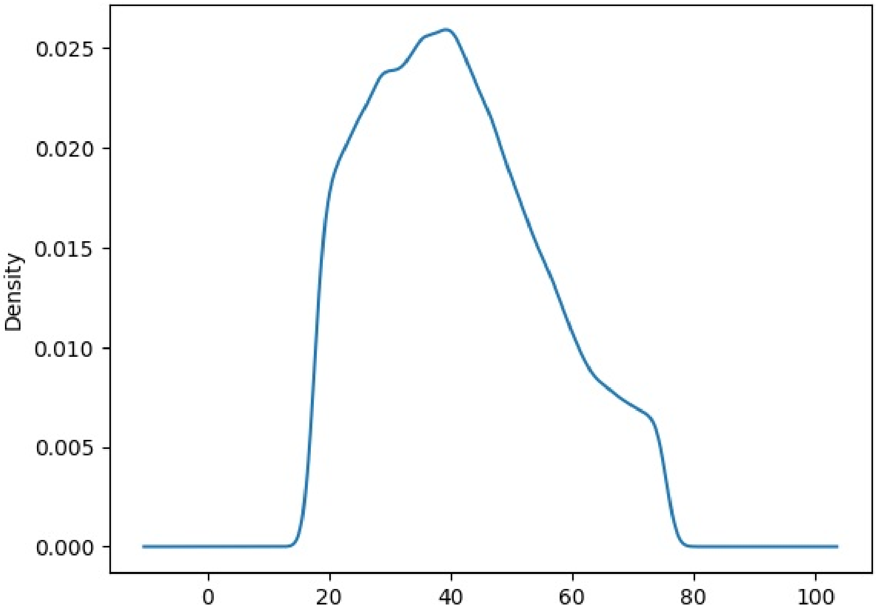
In [213…
```
plt.hist(base.Age) # Les intervalles sont de 10 par défaut
#base.Age.hist(bins=75-18+1) ou
#base.Age.hist(bins=base.Age.nunique()) #une barre par Age
```

Out[213]:
```
(array([11098., 13502., 14665., 12923., 14218., 11438.,  7247.,  6226.,
         4671.,  4009.]),
 array([18. , 23.7, 29.4, 35.1, 40.8, 46.5, 52.2, 57.9, 63.6, 69.3, 75. ]),
 <BarContainer object of 10 artists>)
```



In [214…
```
base.Age.plot(kind="kde") #preferable que le histograme la curve de densité,une estimation de la probabilité qu
```

Out[214]:
```
<AxesSubplot:ylabel='Density'>
```

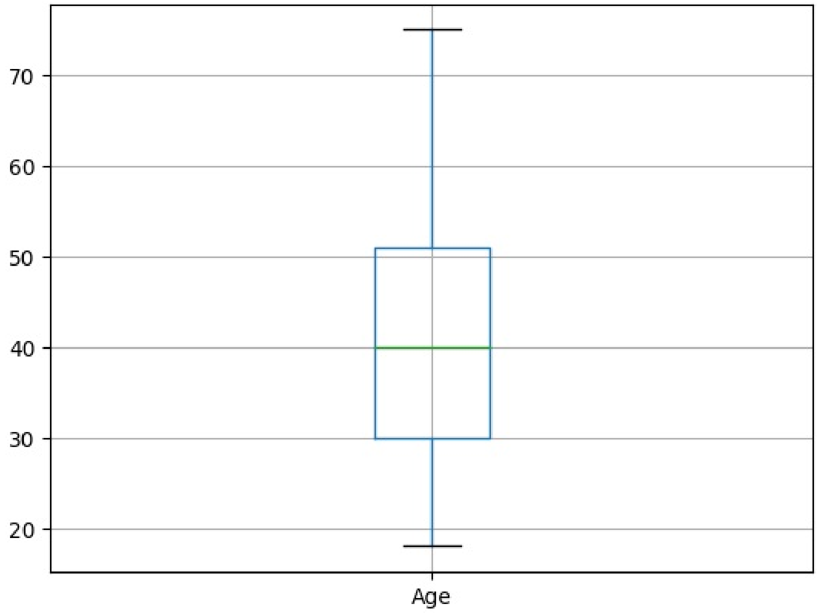


In [215…

```
base.boxplot(column = ["Age"]) #base.boxplot(["Age"])
```

Out[215]:

```
&lt;AxesSubplot:&gt;
```



In [216…

```
base.Age.quantile([0.25,0.5,0.75, 0.95]) # les quantiles du Boxplot
```

Out[216]:

```
0.25    30.0
0.50    40.0
0.75    51.0
0.95    68.0
Name: Age, dtype: float64
```

In [217…

```
freqAge = pd.crosstab(base.Age, "freq");freqAge.head() #tableau croissé Age et freq
```

Out[217]:

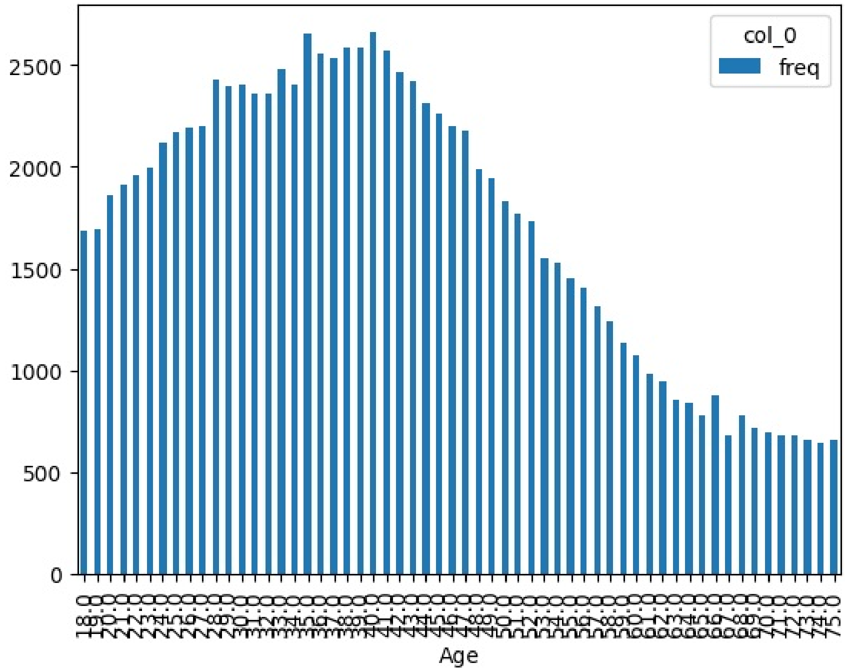
col_0	freq
Age	
18.0	1685
19.0	1694
20.0	1856
21.0	1912
22.0	1955

In [218…

```
freqAge.plot.bar() #une barre par age et sa frequence
```

Out[218]:

```
<AxesSubplot:xlabel='Age'>
```



In [219…

```
base.Age.quantile([0.05,0.1,0.25,0.5,0.75, 0.95])
```

Out[219]:

```
0.05    20.0
0.10    23.0
0.25    30.0
0.50    40.0
0.75    51.0
0.95    68.0
Name: Age, dtype: float64
```

Au niveau géneral notre portefeuille est constitué d'une population jeune, avec une age moyenne de 41. Le Q2 es trés proche à la moyenne, pour cela notre Boxplot devien presque symetrique.

Le 75% de notre population a moins de 51 ans.

Analyse de la variable « Gender »

In [220…

```
base.Gender.value_counts() #tdc
```

Out[220]:

```
Male      63432
Female    36565
Name: Gender, dtype: int64
```

In [221…

```
base.Gender.describe()
```

Out[221]:

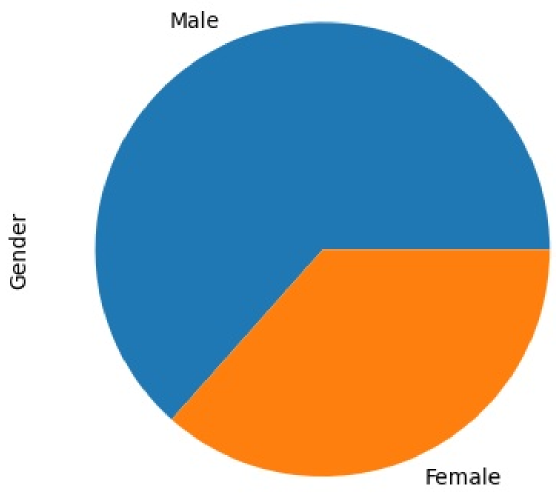
```
count      99997
unique         2
top         Male
freq       63432
Name: Gender, dtype: object
```

In [222…

```
base.Gender.value_counts().plot(kind="pie")
#base['Gender'].value_counts().plot.pie()
#plt.pie(base.Gender.value_counts(), labels=base.Gender.value_counts().index)
```

Out[222]:

```
<AxesSubplot:ylabel='Gender'>
```



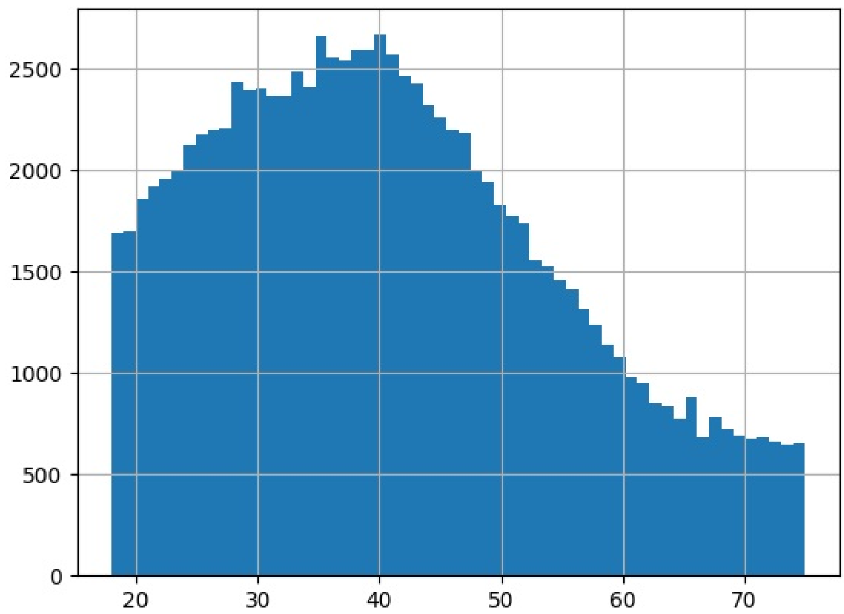
Répartition de l'âge par sexe

In [223…

```
#une barre par Age
base.Age.hist(bins=base.Age.nunique())
```

Out[223]:

```
<AxesSubplot:>
```



In [224…

```
round(base.groupby(['Gender']).Age.mean(),1) # via les propriétés de la dataframe
```

Out[224]:

```
Gender
Female    39.8
Male      41.9
Name: Age, dtype: float64
```

In [225…

```
g = base.groupby('Gender')
```

In [226…

```
g.get_group('Male')['Age'].mean() # via la scission
```

Out[226]:

```
41.8936656577122
```

In [227…

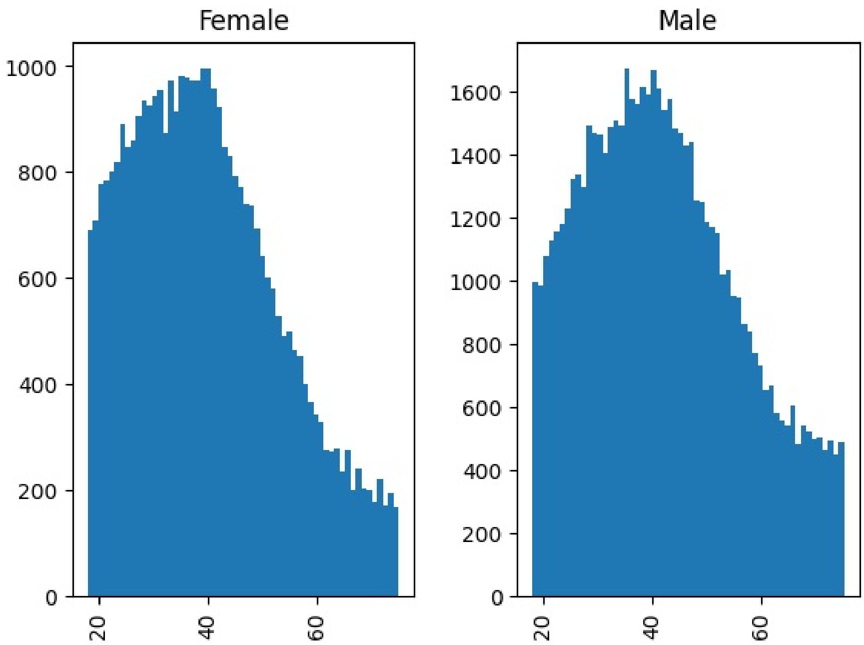
```
for groupe in g:
    print(groupe[0])
    print(round(np.mean(groupe[1]['Age']),1))
```

```
Female
39.8
Male
41.9
```

In [228…

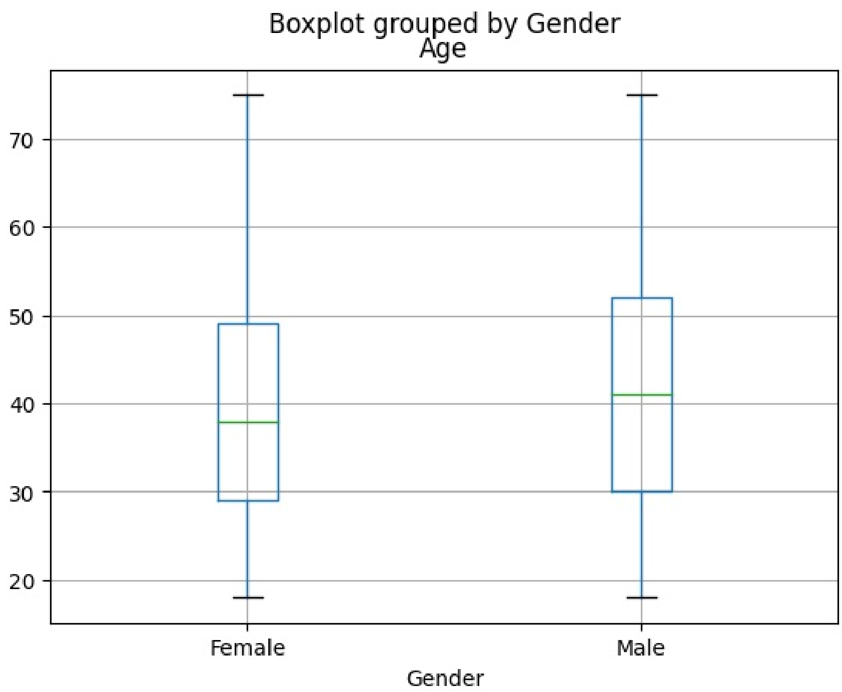
```
base.hist('Age',bins=len(np.unique(base.Age)),by = 'Gender') #age Par sexe
```

Out[228]:
```
array([<AxesSubplot:title={'center':'Female'}>,
       <AxesSubplot:title={'center':'Male'}>], dtype=object)
```



In [229…
```
base.boxplot(column='Age',by='Gender') # via la dataframe
```

Out[229]:
```
<AxesSubplot:title={'center':'Age'}, xlabel='Gender'>
```



La medianne de l'age chez les femmes est plus bas (39 ans) face 42 chez les hommes

25% des hommes ont moins de 30 ans et le 75% ont moins de 52 ans

25% des femmes ont moins de 29 ans et le 75% ont moins de 49 ans

Une population relativement plus jeune chez les femmes.

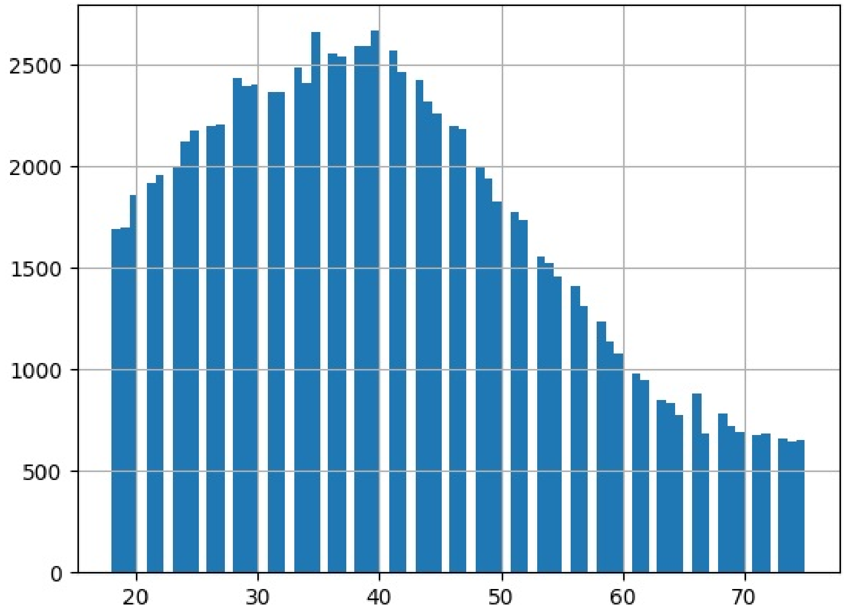
Tableau de contingence (ou stat des) et l'histogramme à travers une fonction

In [230…
```
def stat_des(var):
    if (base[var].dtypes in ("object","int64")):
        freq = pd.value_counts(base[var])
        print(freq.sort_index())
        modal = freq.index
        freqAbs = freq.values
        plt.bar(modal, freqAbs)
        print(base[var].describe())
    else:
        print(base[var].describe(include='all'))
        base[var].hist(bins=80)
        return "Analyse de " + var
```

In [231…
```
stat_des("Age") # Appel de la fonction avec la variable age
```

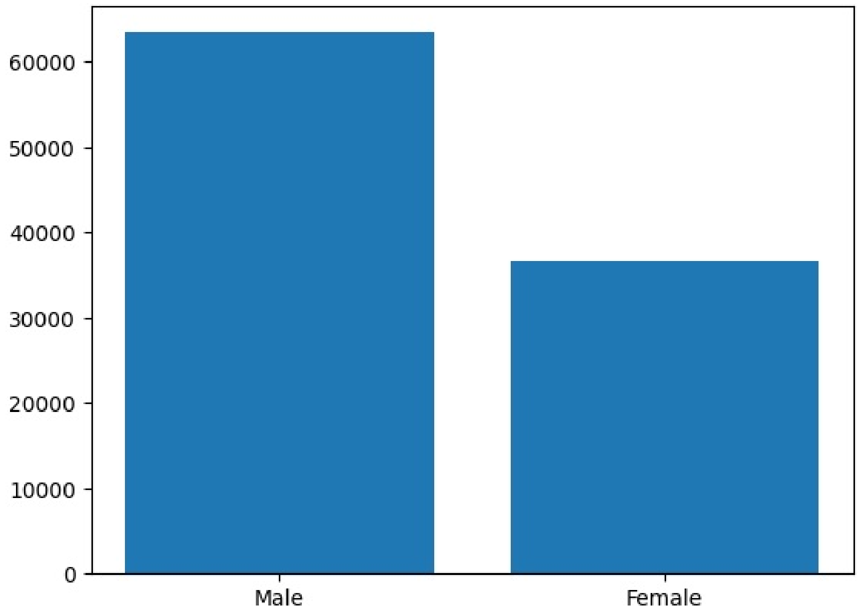
```
count    99997.000000
mean        41.124144
std         14.299563
min         18.000000
25%         30.000000
50%         40.000000
75%         51.000000
max         75.000000
Name: Age, dtype: float64
```

Out[231]:
```
'Analyse de Age'
```



In [232…
```
stat_des("Gender") # Appel de la fonction avec la variable Gender
```

```
Female    36565
Male      63432
Name: Gender, dtype: int64
count     99997
unique        2
top        Male
freq      63432
Name: Gender, dtype: object
```



Analyse des sinistres

In [233…
```
base.chg_sin.describe()
```

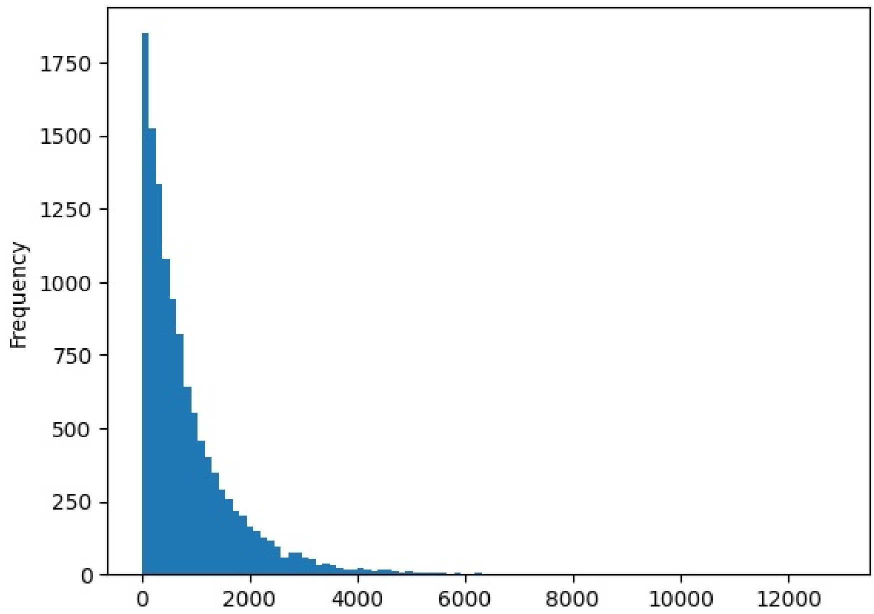
```
Out[233]: count    99997.000000
mean       106.324932
std        446.644212
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max      12878.370000
Name: chg_sin, dtype: float64
```

```
In [234… base[base.chg_sin != 0].chg_sin.describe() #les sinistres differents à 0
```

```
Out[234]: count    12255.000000
mean       867.578475
std        983.565248
min          0.180000
25%        228.035000
50%        562.840000
75%       1154.915000
max      12878.370000
Name: chg_sin, dtype: float64
```

```
In [235… base[base.chg_sin != 0].chg_sin.plot(kind="hist",bins=100)
```

```
Out[235]: <AxesSubplot:ylabel='Frequency'>
```

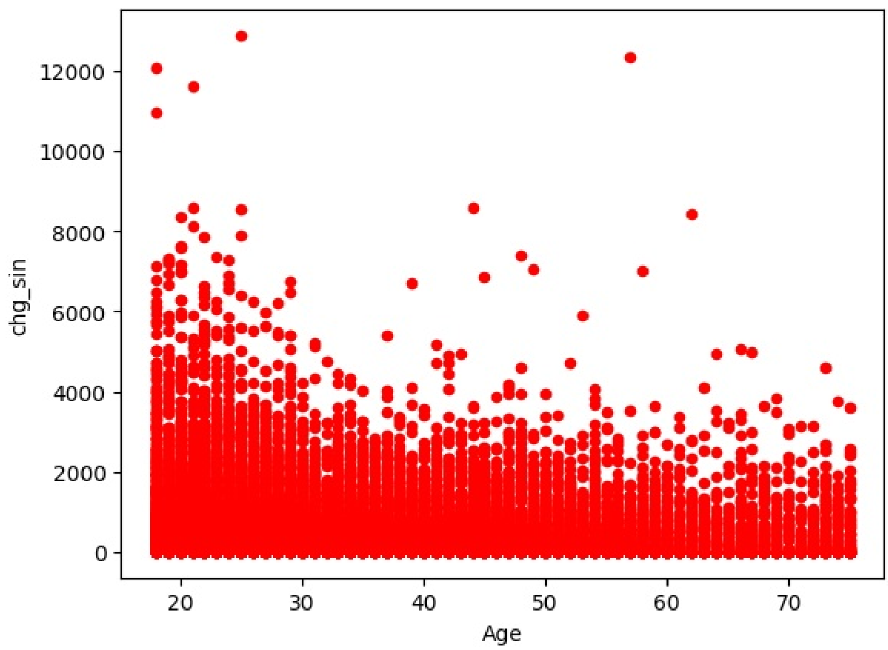


Le coût de sinistres décroît car on a beaucoup d'événements qui veulent très peu, et très peu qui veulent chère.

Charge de sinistre par âge

```
In [236… base.plot.scatter(x='Age',y='chg_sin',c='red')
#base.plot.scatter('Age','chg_sin',c='red')
```

```
Out[236]: <AxesSubplot:xlabel='Age', ylabel='chg_sin'>
```



Ce nuage de points montre globalement une décroissance. Peut être facile attirer la conclusion de que les jeunes (20 ans-45 ans) sont plus sinistrés en comparaison aux personnes âgées. Néanmoins, le 75% de notre portefeuille est composé des personnes de moins de 50 ans. Alors, forcément on aura plus de sinistres chez les jeunes. Il faudra donc, normaliser nos données pour faire une comparaison plus cohérente entre les groupes.

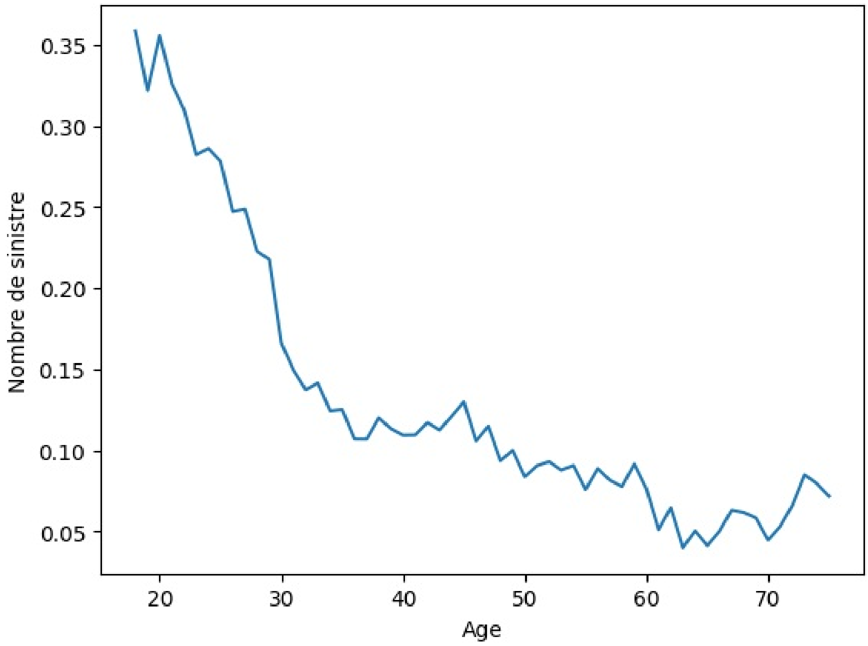
Nombre de sinistre moyen par âge

In [237…
```
round(base.groupby(['Age']).nb_sin.mean(),4) # Pas très lisible
```

Out[237]:
```
Age
18.0    0.3585
19.0    0.3217
20.0    0.3556
21.0    0.3258
22.0    0.3100
23.0    0.2821
24.0    0.2860
25.0    0.2780
26.0    0.2472
27.0    0.2486
28.0    0.2226
29.0    0.2175
30.0    0.1660
31.0    0.1492
32.0    0.1373
33.0    0.1414
34.0    0.1243
35.0    0.1251
36.0    0.1069
37.0    0.1069
38.0    0.1199
39.0    0.1133
40.0    0.1093
41.0    0.1094
42.0    0.1170
43.0    0.1124
44.0    0.1210
45.0    0.1298
46.0    0.1056
47.0    0.1148
48.0    0.0935
49.0    0.0999
50.0    0.0837
51.0    0.0904
52.0    0.0929
53.0    0.0877
54.0    0.0906
55.0    0.0758
56.0    0.0887
57.0    0.0814
58.0    0.0776
59.0    0.0915
60.0    0.0762
61.0    0.0510
62.0    0.0646
63.0    0.0399
64.0    0.0503
65.0    0.0413
66.0    0.0501
67.0    0.0630
68.0    0.0616
69.0    0.0583
70.0    0.0447
71.0    0.0531
72.0    0.0662
73.0    0.0848
74.0    0.0794
75.0    0.0718
Name: nb_sin, dtype: float64
```

In [238…
```
plt.plot(base.groupby(['Age']).nb_sin.mean())
plt.xlabel("Age")
plt.ylabel("Nombre de sinistre")
```

Out[238]:
```
Text(0, 0.5, 'Nombre de sinistre')
```



Le nombre de sinistre décroît à la mesure que l'âge augmente. Dans la tranche de 65 ans à 70 ans il y a une augmentation. Cela peut être expliqué par le détériore normal des conditions de santé.

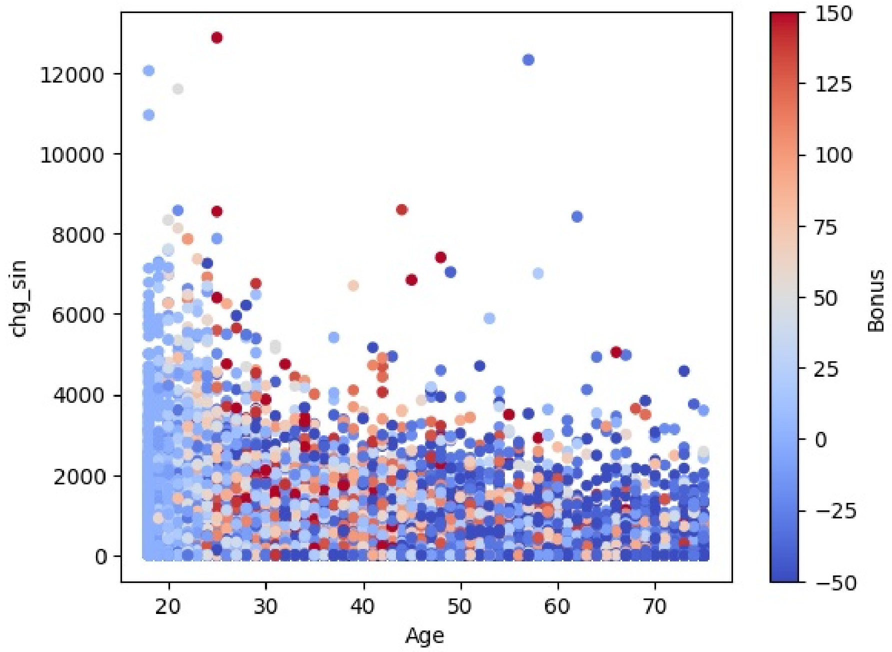
âge vs charge de sinistre/nombre de sinistre en indiquant le niveau de Bonus

In [258…

```
base.plot.scatter(x='Age',y='chg_sin', c='Bonus',cmap='coolwarm')
#base.plot.scatter('Age','chg_sin', c="Bonus",cmap="coolwarm")
```

Out[258]:

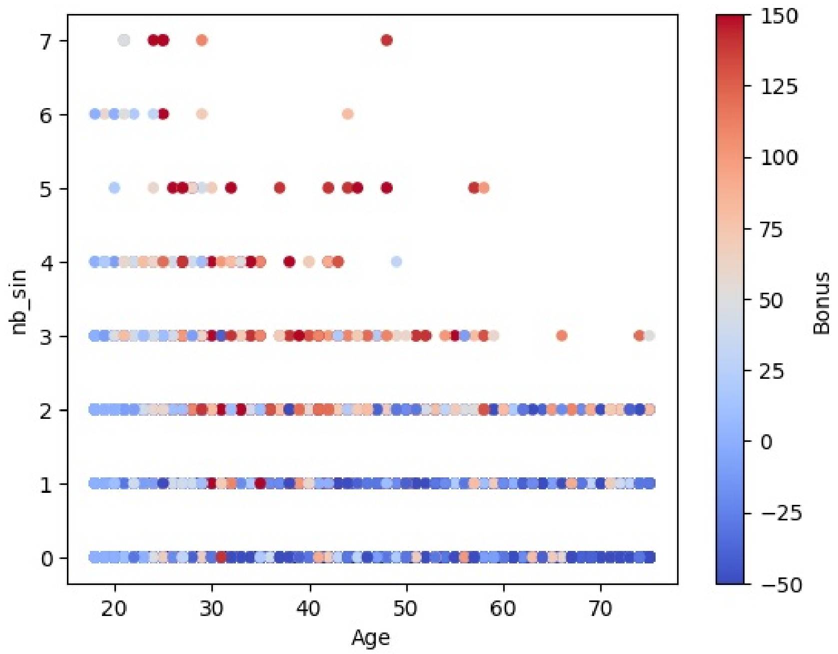
```
<AxesSubplot:xlabel='Age', ylabel='chg_sin'>
```



- La couleur pâle chez les jeunes est expliquée par l'ancienneté du permis de conduire. Permis nouveau alors plus probable d'avoir 0 sinistre.
- Chez les âgés de 25 à 42 ans, la couleur rouge foncé indique un niveau haut des coûts des sinistres. Ce que se traduit en un qualification « malus ».
- Comportement très hétérogène dans la tranche de l'âge 25 à 49 ans.
- Le coté où le bleu foncé est plus volumineux, indique que les plus âgées sont plus prudents

bonnus malus = tarif plus elévé

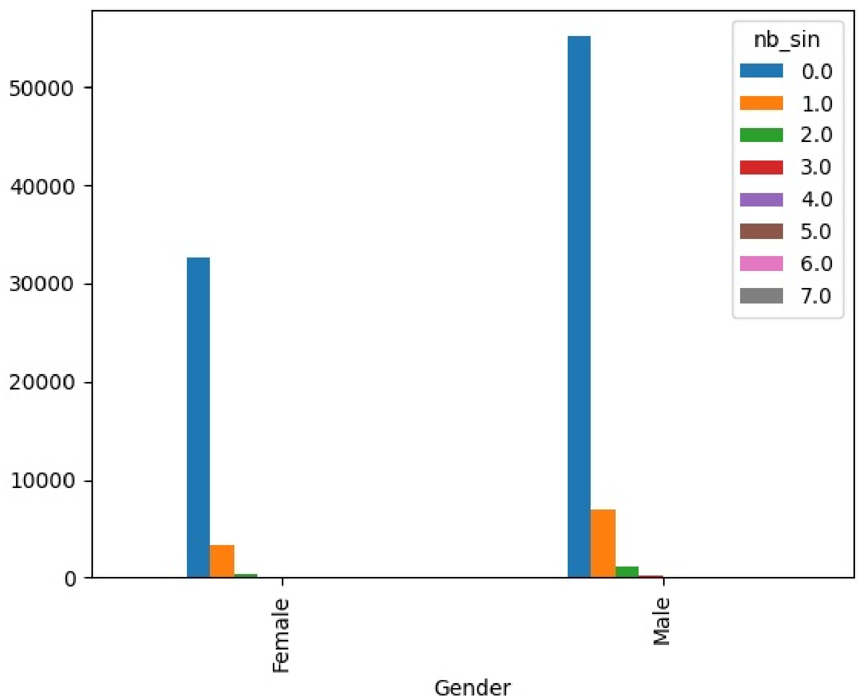
In [259…
```
base.plot.scatter(x='Age',y='nb_sin', c='Bonus',cmap='coolwarm')
```
Out[259]:
```
<AxesSubplot:xlabel='Age', ylabel='nb_sin'>
```



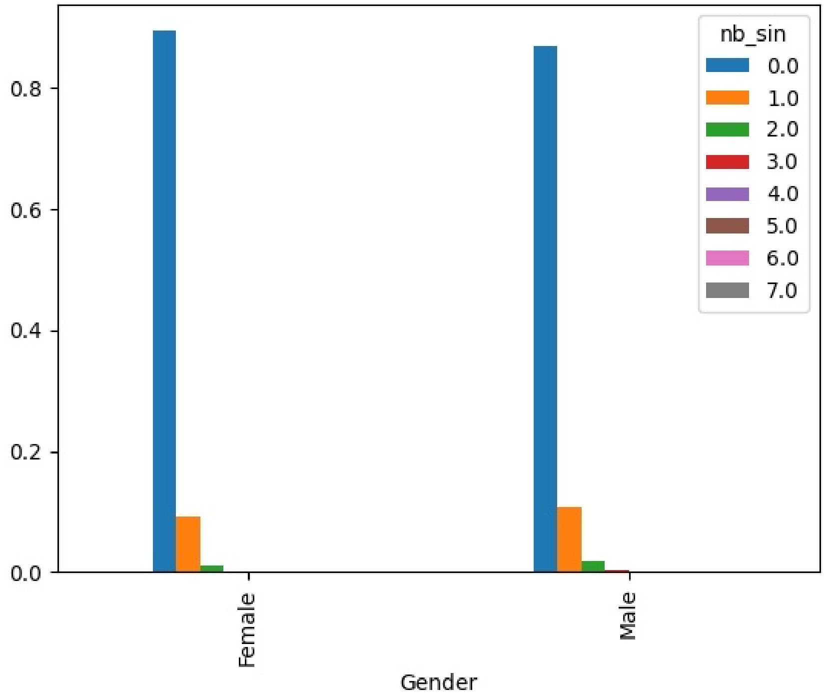
Une volumétrie faible pour un nombre de sinistres entre 6 et 7. Concentration de la volumétrie des sinistres entre 3 et 5 pour les âgés de 30 à 59 ans.

Nombre de sinistre moyen par sexe

In [239…
```
Freq_Sin_sexe = pd.crosstab(base.Gender,base.nb_sin)
Freq_Sin_sexe.plot.bar(stacked = False) #stacked=epiler
# la volumetrie des hommes c'est pas la même ()il ya beacoup plus hommes que des femmes. Alors dificil pour com
```
Out[239]:
```
<AxesSubplot:xlabel='Gender'>
```



In [240…
```
# Normaliser (une freqeuce relative)pour mieux comparer
Freq_Sin_sexe = pd.crosstab(base.Gender,base.nb_sin, normalize = "index") #Index=femmes et hommes
Freq_Sin_sexe.plot.bar(stacked = False)
# un regarde un comportement très similaire
```
Out[240]:
```
<AxesSubplot:xlabel='Gender'>
```



Nombre de sinistre moyen par Age et par sexe

In [241…

```
pd.crosstab(base['Age'], base['Gender'],
            values=base['nb_sin'],
            aggfunc=pd.Series.mean) # via crosstab
```

Out[241]:

Gender	Female	Male
Age		
18.0	0.253623	0.431156
19.0	0.194915	0.412779
20.0	0.215938	0.456401
21.0	0.182398	0.425532
22.0	0.197500	0.387879
23.0	0.182152	0.351443
24.0	0.187220	0.357783
25.0	0.193853	0.331822
26.0	0.193473	0.281648
27.0	0.173673	0.300926
28.0	0.163812	0.259383
29.0	0.159655	0.254087
30.0	0.123142	0.193703
31.0	0.118325	0.170107
32.0	0.129587	0.141801
33.0	0.137718	0.143804
34.0	0.117196	0.128600
35.0	0.119388	0.128435
36.0	0.092119	0.116117
37.0	0.101747	0.110045
38.0	0.119342	0.120273
39.0	0.104418	0.118793
40.0	0.095573	0.117506
41.0	0.101253	0.114286
42.0	0.100868	0.126623
43.0	0.117021	0.109981
44.0	0.130277	0.115825
45.0	0.128951	0.130286

45.0	[illegible]	[illegible]
46.0	0.101167	0.107919
47.0	0.109459	0.117606
48.0	0.070652	0.106858
49.0	0.092352	0.104167
50.0	0.068536	0.091906
51.0	0.104825	0.083048
52.0	0.089501	0.094618
53.0	0.081285	0.091087
54.0	0.087935	0.091876
55.0	0.074148	0.076681
56.0	0.079570	0.093220
57.0	0.084257	0.079954
58.0	0.095238	0.069212
59.0	0.104110	0.085603
60.0	0.055394	0.085948
61.0	0.052147	0.050382
62.0	0.054348	0.068862
63.0	0.047619	0.036269
64.0	0.032258	0.059353
65.0	0.038298	0.042593
66.0	0.040146	0.054545
67.0	0.075758	0.057851
68.0	0.046025	0.068519
69.0	0.069652	0.053846
70.0	0.055000	0.040486
71.0	0.062500	0.049801
72.0	0.077626	0.060738
73.0	0.111765	0.075510
74.0	0.092308	0.073826
75.0	0.107143	0.059548

In [242…

```
base.pivot_table(index=['Age'],columns=['Gender'],
                 values=['nb_sin'],
                 aggfunc=np.mean) # via pivot_table
```

Out[242]:

		nb_sin
Gender	**Female**	**Male**
Age		
18.0	0.253623	0.431156
19.0	0.194915	0.412779
20.0	0.215938	0.456401
21.0	0.182398	0.425532
22.0	0.197500	0.387879
23.0	0.182152	0.351443
24.0	0.187220	0.357783
25.0	0.193853	0.331822
26.0	0.193473	0.281648
27.0	0.173673	0.300926
28.0	0.163812	0.259383
29.0	0.159655	0.254087
30.0	0.123142	0.193703
31.0	0.118325	0.170107
32.0	0.129587	0.141801
33.0	0.137718	0.143804
34.0	0.117196	0.128600
35.0	0.119388	0.128435

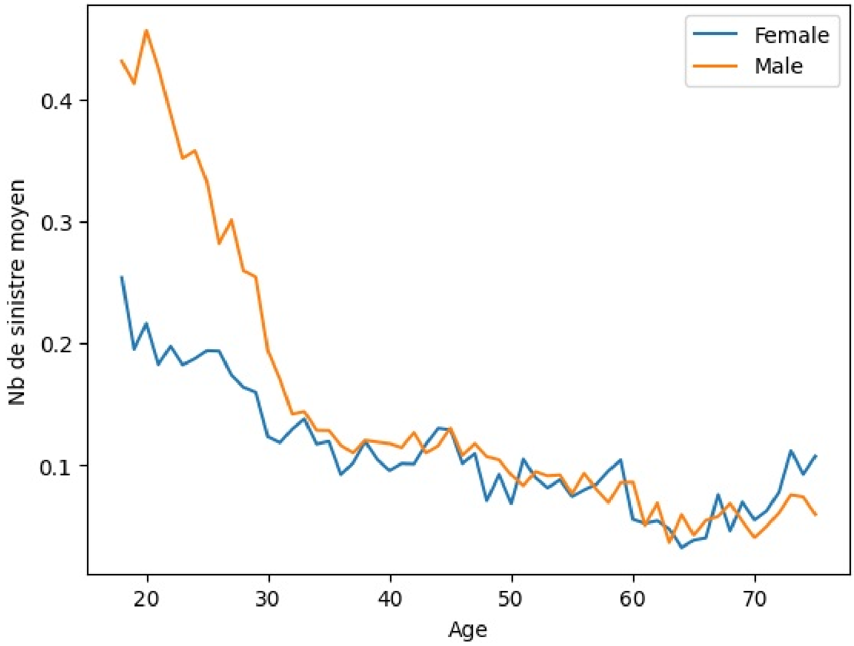
36.0	0.092119	0.116117
37.0	0.101747	0.110045
38.0	0.119342	0.120273
39.0	0.104418	0.118793
40.0	0.095573	0.117506
41.0	0.101253	0.114286
42.0	0.100868	0.126623
43.0	0.117021	0.109981
44.0	0.130277	0.115825
45.0	0.128951	0.130286
46.0	0.101167	0.107919
47.0	0.109459	0.117606
48.0	0.070652	0.106858
49.0	0.092352	0.104167
50.0	0.068536	0.091906
51.0	0.104825	0.083048
52.0	0.089501	0.094618
53.0	0.081285	0.091087
54.0	0.087935	0.091876
55.0	0.074148	0.076681
56.0	0.079570	0.093220
57.0	0.084257	0.079954
58.0	0.095238	0.069212
59.0	0.104110	0.085603
60.0	0.055394	0.085948
61.0	0.052147	0.050382
62.0	0.054348	0.068862
63.0	0.047619	0.036269
64.0	0.032258	0.059353
65.0	0.038298	0.042593
66.0	0.040146	0.054545
67.0	0.075758	0.057851
68.0	0.046025	0.068519
69.0	0.069652	0.053846
70.0	0.055000	0.040486
71.0	0.062500	0.049801
72.0	0.077626	0.060738
73.0	0.111765	0.075510
74.0	0.092308	0.073826
75.0	0.107143	0.059548

In [243…

```
NS_moy_Age = pd.crosstab(base['Age'], base['Gender'], values=base.nb_sin,
                         aggfunc=np.mean)
plt.plot(NS_moy_Age.index,NS_moy_Age.iloc[:,0], #Female
         label=NS_moy_Age.columns[0])
plt.plot(NS_moy_Age.index,NS_moy_Age.iloc[:,1], #Male
        label=NS_moy_Age.columns[1])
plt.legend()
plt.xlabel("Age")
plt.ylabel("Nb de sinistre moyen")
```

Out[243]:

```
Text(0, 0.5, 'Nb de sinistre moyen')
```



Charge sinistre Moyen par Age et par sexe

In [244…
```
base.pivot_table(index=['Age'],columns=['Gender'],values=['chg_sin'],
                 aggfunc=np.mean)
```

Out[244]:

	chg_sin	
Gender	**Female**	**Male**
Age		
18.0	223.673246	477.919618
19.0	187.163376	382.607677
20.0	193.610861	399.647950
21.0	162.002003	359.620390
22.0	183.579375	343.022156
23.0	167.944914	280.966723
24.0	158.015919	291.078615
25.0	134.004704	246.470401
26.0	137.401189	186.563243
27.0	137.947323	201.434722
28.0	120.096445	174.488023
29.0	109.611909	211.609475
30.0	73.672346	132.071068
31.0	66.227435	101.342342
32.0	67.929564	83.922897
33.0	78.945355	98.237276
34.0	75.235465	83.869109
35.0	70.949510	79.552139
36.0	48.593132	71.847963
37.0	54.517492	77.239360
38.0	63.429465	80.493453
39.0	60.408494	75.657643
40.0	57.954628	67.474323
41.0	56.364123	68.608795
42.0	55.303525	80.738351

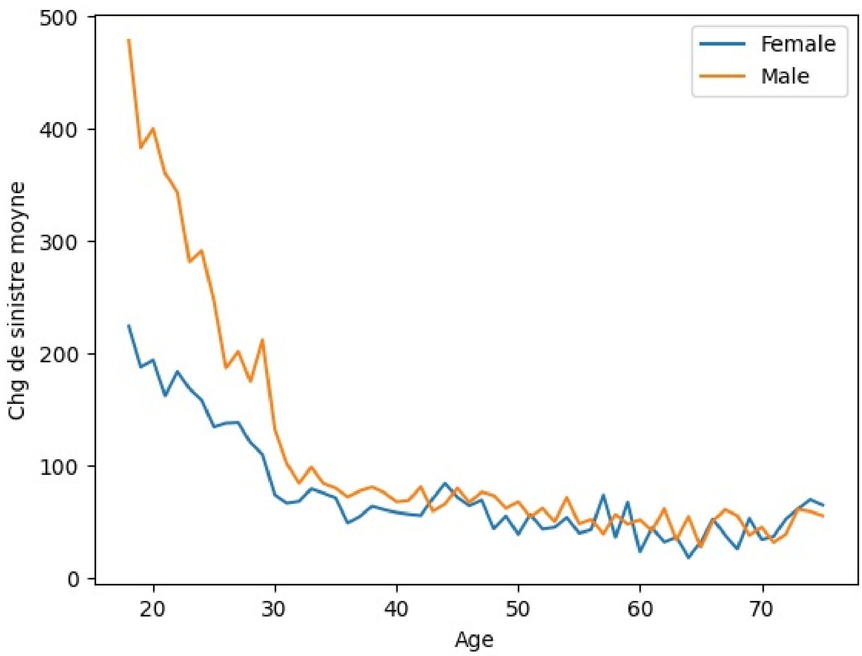
43.0	70.587435	59.142740
44.0	83.899397	65.566660
45.0	71.504564	79.575935
46.0	63.921440	67.033294
47.0	69.111757	76.171475
48.0	43.527079	73.039370
49.0	54.893810	61.586114
50.0	38.493411	67.470995
51.0	56.218952	54.041438
52.0	43.132478	61.645061
53.0	44.725009	49.965152
54.0	53.606421	71.375590
55.0	39.581082	47.898771
56.0	42.550645	51.745847
57.0	73.558160	38.766257
58.0	36.075313	56.197757
59.0	67.129425	47.433658
60.0	22.780875	51.522660
61.0	44.350399	41.551160
62.0	31.628913	61.295584
63.0	35.893846	33.867478
64.0	17.734659	54.380090
65.0	31.504340	27.171333
66.0	52.190620	50.183339
67.0	37.861919	60.471798
68.0	25.416695	55.224222
69.0	52.878905	37.623846
70.0	33.959050	44.778806
71.0	36.624091	31.255936
72.0	52.227580	38.822104
73.0	61.179588	60.804531
74.0	69.577231	58.838076
75.0	64.602500	55.117228

In [245…

```
CS_moy_Age = pd.crosstab(base['Age'], base['Gender'], values=base.chg_sin, aggfunc=np.mean)
plt.plot(CS_moy_Age.index,CS_moy_Age.iloc[:,0],label=CS_moy_Age.columns[0])
plt.plot(CS_moy_Age.index,CS_moy_Age.iloc[:,1],label=CS_moy_Age.columns[1])
plt.legend()
plt.xlabel("Age")
plt.ylabel("Chg de sinistre moyne")
```

Out[245]:

```
Text(0, 0.5, 'Chg de sinistre moyne')
```



Les deux graphiques mettent en evidence qu'il existe un comportement très différente entre les hommes et les femmes dans la tranche d'âge de 20 à 30 ans. La série se stabilise après le 35 ans en démontrant un comportement similaire des groupes.

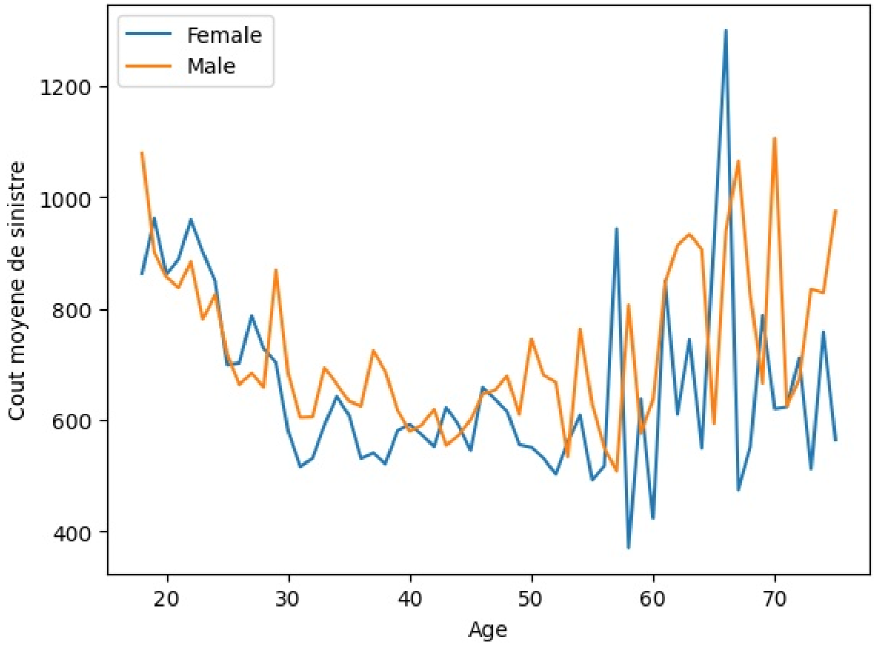
Cout moyene de sinistre

In [246…

```
CS_moy_Age = pd.crosstab(base['Age'], base['Gender'],
                         values=base.chg_sin/base.nb_sin, #Coût
                         aggfunc=np.mean)
plt.plot(CS_moy_Age.index,CS_moy_Age.iloc[:,0],label=CS_moy_Age.columns[0])
plt.plot(CS_moy_Age.index,CS_moy_Age.iloc[:,1],label=CS_moy_Age.columns[1])
plt.legend()
plt.xlabel("Age")
plt.ylabel("Cout moyene de sinistre")
```

Out[246]:

```
Text(0, 0.5, 'Cout moyene de sinistre')
```



- Une décroissance pour le plus jeunes et une grande volatilité après les 60 ans.
- La série se stabilise entre les 30 et 55 ans.
- Le coût des sinistres chez les femmes sont plus élevés après les 55 ans.

In [247…
```
g[['nb_sin','chg_sin']].agg([np.mean,np.std])
```

Out[247]:

	nb_sin		chg_sin	
	mean	**std**	**mean**	**std**
Gender				
Female	0.123725	0.387992	84.884305	386.428495
Male	0.161448	0.464936	118.684254	477.486012

In [248…
```
g.nb_sin.mean()
```

Out[248]:
```
Gender
Female    0.123725
Male      0.161448
Name: nb_sin, dtype: float64
```

In [249…
```
g.get_group('Male').nb_sin.mean()
```

Out[249]:
```
0.16144848026232816
```

In [250…
```
# Une Autre construction
agregGenderAge = base.groupby(['Gender', 'Age'])
agregGenderAge.count()
base.groupby(['Gender']).count()
base.groupby(['Gender']).size()
base.groupby(['Gender', 'Age']).mean()
base.groupby(['Gender']).nunique()
base.groupby(['Gender'],dropna=False).nunique()
base.groupby('Gender').agg(['mean', 'median']).chg_sin
```

```
C:\Users\IDEAPAD5\AppData\Local\Temp\ipykernel_12872\4083134700.py:9: FutureWarning: ['Type', 'Category', 'Occupation', 'Group2'] did not aggregate successfully. If any error is raised this will raise in a future version of pandas. Drop these columns/ops to avoid this warning.
  base.groupby('Gender').agg(['mean', 'median']).chg_sin
```

Out[250]:

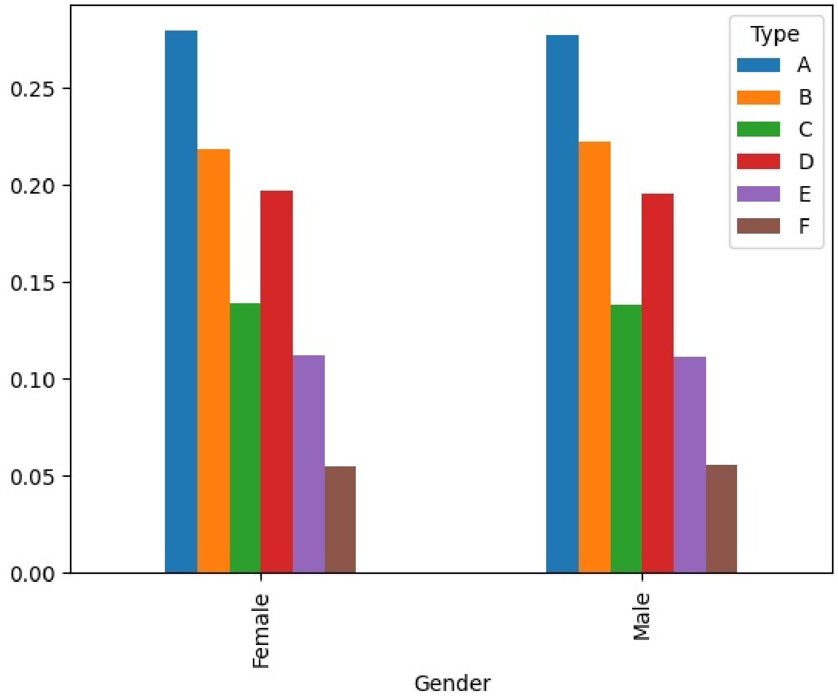
	mean	median
Gender		
Female	84.884305	0.0
Male	118.684254	0.0

Sexe par type vehicule

In [251…

```
freqGenderType = pd.crosstab(base.Gender, base.Type,normalize="index")
freqGenderType.plot.bar(stacked = False)
```

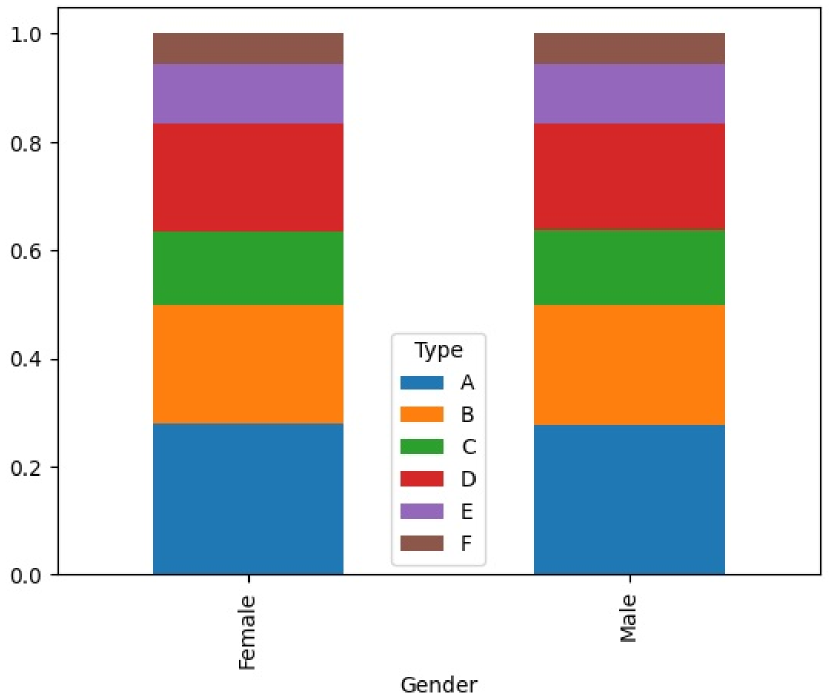
Out[251]: <AxesSubplot:xlabel='Gender'>



In [252…

```
freqGenderType = pd.crosstab(base.Gender, base.Type, normalize="index")
freqGenderType.plot.bar(stacked = True)
```

Out[252]: <AxesSubplot:xlabel='Gender'>



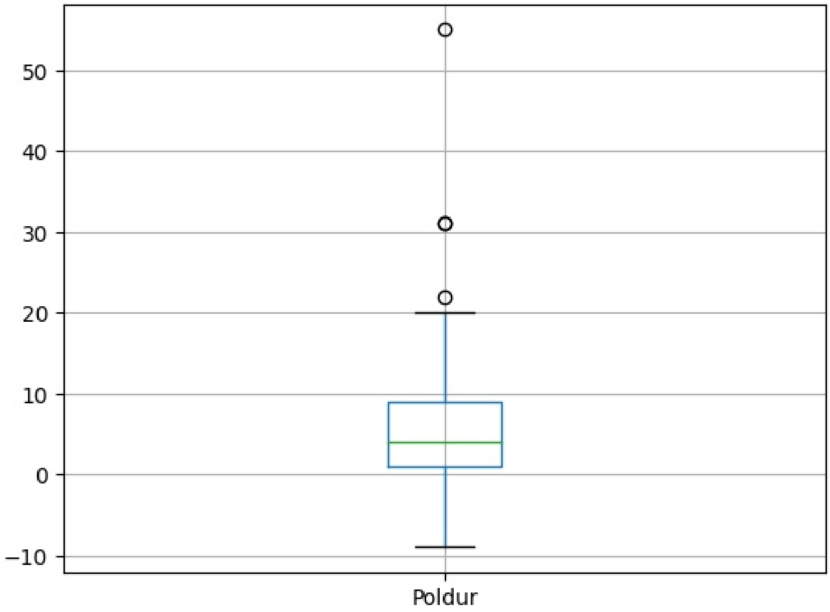
Composition très similaire du portefeuille par rapport au type de véhicule acheté.

age vs duration du contrat

In [268…
```
base.Poldur.mean()
```
Out[268]:
```
5.472278336700201
```

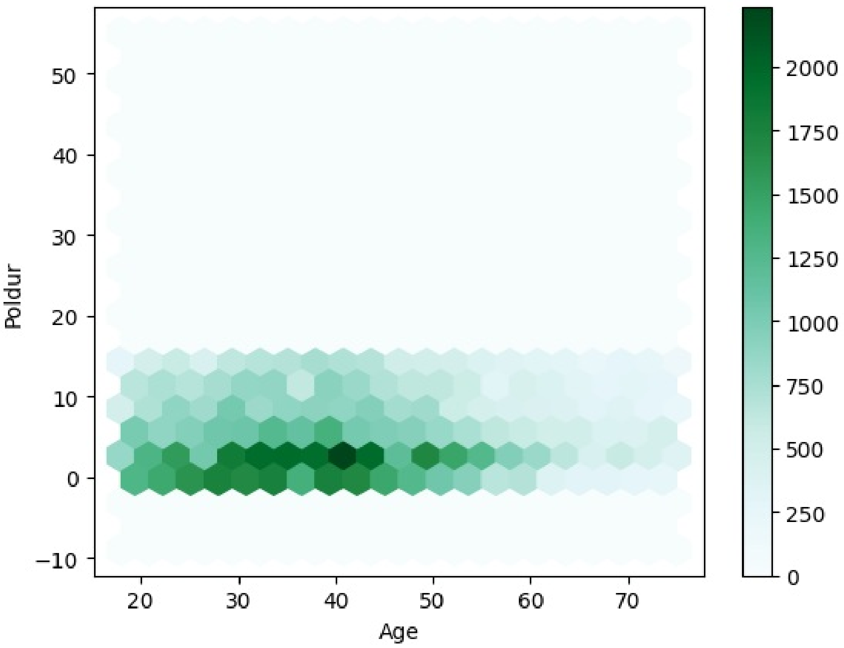
In [267…
```
base.groupby(['Age']).Poldur.mean()
```
Out[267]:
```
Age
18.0    5.291988
19.0    5.424439
20.0    5.335129
21.0    5.265167
22.0    5.414834
23.0    5.494990
24.0    5.335064
25.0    5.275921
26.0    5.289102
27.0    5.185909
28.0    5.391591
29.0    5.342380
30.0    5.175198
31.0    5.378390
32.0    5.355709
33.0    5.261886
34.0    5.153782
35.0    5.334213
36.0    5.327066
37.0    5.484227
38.0    5.458027
39.0    5.420564
40.0    5.249061
41.0    5.172897
42.0    5.255483
43.0    5.394378
44.0    5.354797
45.0    5.316349
46.0    5.787989
47.0    5.421222
48.0    5.431658
49.0    5.464952
50.0    5.477571
51.0    5.394573
52.0    5.363531
53.0    5.512903
54.0    5.470125
55.0    5.203308
56.0    5.287438
57.0    5.528158
58.0    5.357316
59.0    5.509683
60.0    5.657993
61.0    6.362895
62.0    6.038136
63.0    6.403756
64.0    6.486228
65.0    6.313548
66.0    6.299204
67.0    6.107038
68.0    6.278562
69.0    6.198336
70.0    6.502882
71.0    6.318584
72.0    6.536765
73.0    6.306061
74.0    6.510903
75.0    6.155725
Name: Poldur, dtype: float64
```

In [271…
```
base.boxplot(column='Poldur') # via la dataframe
```
Out[271]:
```
<AxesSubplot:>
```



In [257…
```
base.plot.hexbin(x='Age',y='Poldur',gridsize=20)
#base.plot.hexbin('Age','Poldur',gridsize=20)
```

Out[257]: <AxesSubplot:xlabel='Age', ylabel='Poldur'>



La grille à la carte de Kohonen met en évidence une faible volumétrie des personnes âgées. A l'appui du boxplot précèdent, on remarque que le 50% des assurés ont une duration du contrat moyenne inférieur à 5 ans. On remarque également une forte concentration chez les assurées de 41 ans.

Covariance

In [279…
```
# Via numpy
#np.cov(base.Age,base.nb_sin)
#Verifie si les variables varient dans la même direction
#on fixe l'attention sur les signes (+,-) pas la magnitude
#la varience ne donne pas une idée de causalité
```

In [275…
```
# Via le dataframe
base[["Age","nb_sin"]].cov()
```

Out[275]:

	Age	nb_sin
Age	204.477508	-1.020501
nb_sin	-1.020501	0.192497

L'analyse de la covariance entre l'âge et le nombre de sinistres est négative. Cela veut dire que ces deux variables vont dans directions différentes. Plus âgés, moins de sinistres.**Ici on évalue la direction pas la magnitude ni une causalité.**

Correlation

In [274…
```
base[["Age","nb_sin"]].corr()
```

Out[274]:

	Age	nb_sin
Age	1.000000	-0.162659
nb_sin	-0.162659	1.000000

Le coefficient de corrélation (rho), confirme le lien négative entre l'âge et le nombre de sinistre. En plus, il donne la magnitude (force de la relation). Dans ce cas-là, est une rélation pas très forte (rho ~> 0 alors grand dispertion) et négative. **Encore fois, on ne parle pas de causalité.**

In [278…
```
# Via numpy
#np.corrcoef(base.Age,base.nb_sin)
# Direction (+,-) et magnitude (Force de la correlation) de la relation de 2 variable
```

In [263…
```
base[['Bonus','nb_sin']].corr() # via la dataframe
```

Out[263]:

	Bonus	nb_sin
Bonus	1.000000	0.236209
nb_sin	0.236209	1.000000

Relation positive et cohérente. Bonus est lié au nombre de sinistres.

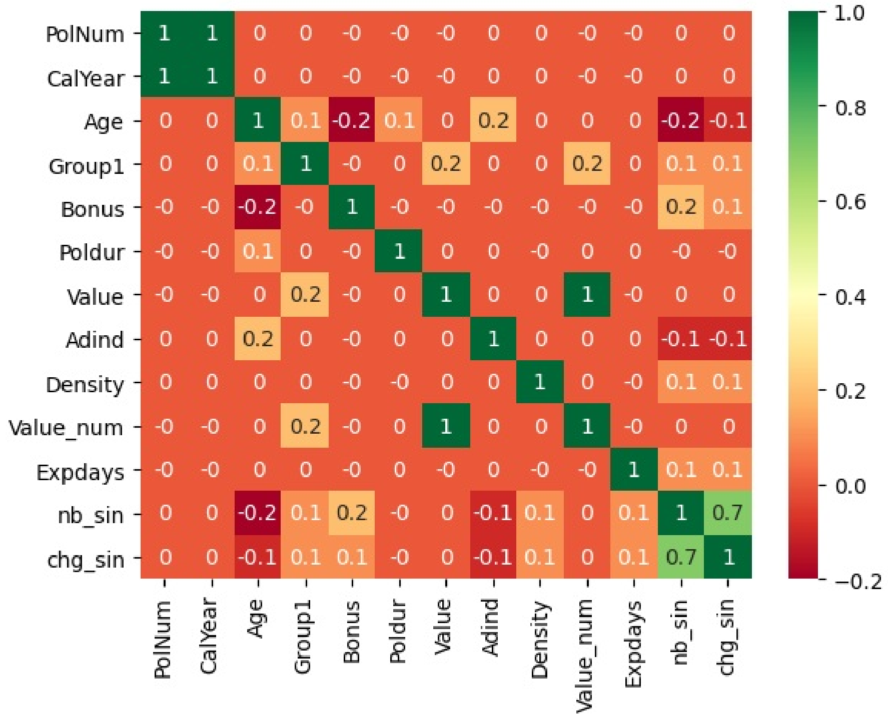
In [277…
```
#via numpy
#np.corrcoef(base.Bonus,base.nb_sin)
```

In [276…
```
#sous forme de tableau
#base.corr(method='kendall')
```

In [281…
```
import seaborn as sns
```

In [282…
```
sns.heatmap(round(base.corr(method='pearson'),1), annot=True, cmap="RdYlGn") #annot=True pour montrer les valeu
#interpretation. example : Bonus et Chg_sin = correlation positf soit le chrge sinistre augmente avec le bonus
#Il peut avoir variables correlées mais pas interesantes
```

Out[282]:
```
<AxesSubplot:>
```



Dans cette matrice de corrélation on utilise la méthode "Pearson" au lieu de "Kendall" ou "Spearman" (Par défaut Pearson).

- Bonus et Chg_sin = corrélation positive soit la charge sinistre augmente avec le bonus.
- Pas toutes les relations sont intéressantes à voir. Exemple value et Group1 "Type de véhicule". Ou le nombre de sinistres et charge de sinistres.

In [ ]:
```
# import Test
from scipy.stats import kendalltau, spearmanr, chi2_contingency, ttest_ind, bartlett
```

Test de Spearman :

Test non parametrique. Le coefficient de corrélation de Spearman (Rs) permet de préciser l'existence d'une liaison entre 2 variables quantitatives et également son intensité. Son carré, le coefficient de détermination (R^2) précise le pourcentage de valeurs expliquées par le modèle de régression défini par la droite de régression.

Hypothèse nulle : "H0 : Rs = 0 ..." Hypothèse alternative : "H1 : Rs est différent de 0 "

http://www.adscience.fr/uploads/ckfiles/files/html_files/StatEL/statel_coefficient_correlation_spearman.htm

In [269…
```
spearmanr(base.chg_sin,base.Bonus)
```

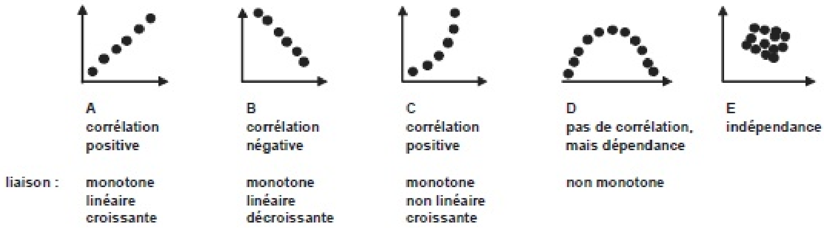
Out[269]:
```
SpearmanrResult(correlation=0.21187871477394854, pvalue=0.0)
```

In [456…
```
spearmanr(base.Age,base.nb_sin)
```

Out[456]:
```
SpearmanrResult(correlation=-0.1691356264655312, pvalue=0.0)
```

Test de Kendall : Plus \tau_a converge en valeur absolue vers 1 et plus la corrélation entre les deux variables est forte. A contrario, plus il tend vers 0 et plus l'orthogonalité entre les deux vecteurs est forte, ce qui implique l'absence de corrélation.

Le coefficient de corrélation de Kendall présente l'intérêt de pouvoir détecter les liaisons monotones contrairement à celui de Bravais-Pearson. La notion de liaison monotone est représentée dans la série de figure ci-dessous.



In [455…
```python
kendalltau(base.Age,base.nb_sin) # tau de Kendall
```
Out[455]:
```
KendalltauResult(correlation=-0.13844619020357793, pvalue=0.0)
```

Tous les tests avec une P-value 0 permettent rejeter H0 : rho = 0 (Pas de corrélation). Par contre H1: rho <> 0 est acceptée mais selon la magnitude des coefficients de corrélation, l'interaction des variables n'est pas assez forte.

Comparaison de 2 échantillons :

- Test du **chi2** d'indépendance pour analyser la corrélation entre l'âge puis le sexe et le nombre de sinistre

mauvaise interpretation de la X^2

https://openstax.org/books/introducci%C3%B3n-estad%C3%ADstica/pages/11-1-datos-sobre-la-distribucion-chi-cuadrado

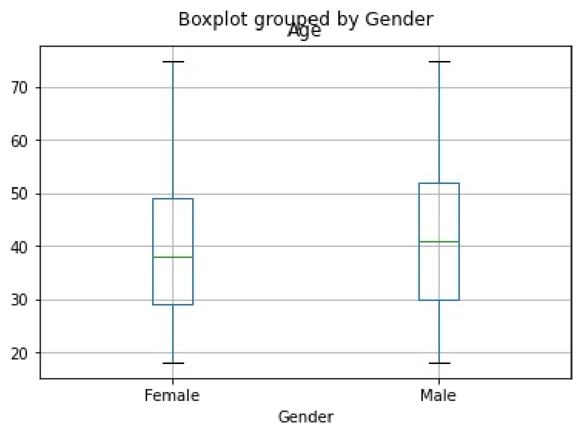
https://www.scribbr.com/statistics/chi-square-test-of-independence/

https://www.youtube.com/watch?v=R9PgZuzqEhY

In [504…
```python
mat = pd.crosstab(base['Gender'], base['nb_sin'])
chi2_contingency(mat)
# H0 : independance (pas de correlation)
# H1 : Dépendance (correlation)
#stat du test 176.266
#Pvalue 1.196.... significative alors on accépte Ho = independance alors il n'y pas relation entre le sexe et n
```
Out[504]:
```
(176.266407147742,
 1.1963588361150693e-34,
 7,
 array([[3.20838248e+04, 3.77106158e+03, 5.61655250e+02, 1.11160935e+02,
         2.44992850e+01, 6.21623649e+00, 4.02227067e+00, 2.55962679e+00],
        [5.56581752e+04, 6.54193842e+03, 9.74344750e+02, 1.92839065e+02,
         4.25007150e+01, 1.07837635e+01, 6.97772933e+00, 4.44037321e+00]]))
```

Test de Bartlett et de student pour comparer deux distributions

In [516…
```python
base.boxplot(column='Age',by='Gender') # via la dataframe
```
Out[516]:
```
<AxesSubplot:title={'center':'Age'}, xlabel='Gender'>
```



In [510…
```python
bartlett(g.get_group('Male').Age,g.get_group('Female').Age)
#Aprés avoir creer g comme un subgroup, ce test Permette comparer les variances(est que on a les meme variance
```

Out[510]:
```
BartlettResult(statistic=73.67383578951707, pvalue=9.215230817932262e-18)
```

In [511…
```
ttest_ind(g.get_group('Male').Age,g.get_group('Female').Age, equal_var=False)
#test studen est parametrique aussi car il suppose une dist gussienne pour tous
#test studen pour comparer les moyenne. Alors on rejet Ho = meme moyenne
```

Out[511]:
```
Ttest_indResult(statistic=22.71275488196123, pvalue=7.795477005400385e-114)
```

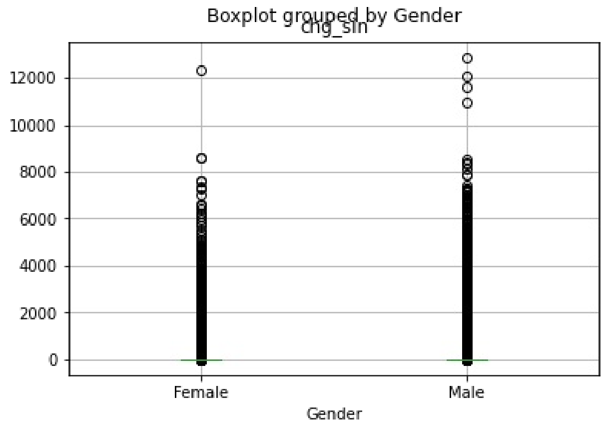
In [515…
```
mannwhitneyu(g.get_group('Male').Age,g.get_group('Female').Age)
# à privileger car il n'est pas parametrique
```

Out[515]:
```
MannwhitneyuResult(statistic=1255635533.5, pvalue=1.2754566027881444e-105)
```

la distrubution des ages est dif chez les homme et chez les femmes, on na pas la même demographie

In [517…
```
base.boxplot(column='chg_sin',by='Gender') # via la dataframe
```

Out[517]:
```
<AxesSubplot:title={'center':'chg_sin'}, xlabel='Gender'>
```



In [518…
```
bartlett(g.get_group('Male').chg_sin,g.get_group('Female').chg_sin)
```

Out[518]:
```
BartlettResult(statistic=1987.2309366230422, pvalue=0.0)
```

In [519…
```
ttest_ind(g.get_group('Male').chg_sin,g.get_group('Female').chg_sin, equal_var=False)
```

Out[519]:
```
Ttest_indResult(statistic=12.197962945052383, pvalue=3.3932028933677e-34)
```

In [520…
```
mannwhitneyu(g.get_group('Male').chg_sin,g.get_group('Female').chg_sin)
```

Out[520]:
```
MannwhitneyuResult(statistic=1189285312.5, pvalue=3.2168799376376035e-32)
```

29. Comparaison de x échantillons : Anova et kruskal

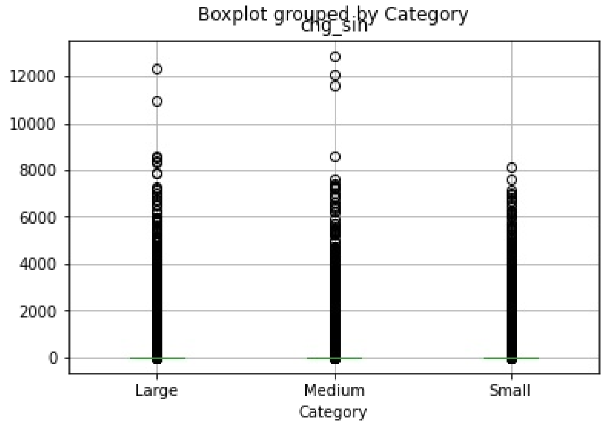
Est-ce que il y a une relation entre la categorie du vehicule et le charge sinistre ? L'idée c'est de cr´´er et comparer 3 çhantillons (Large,mediun,large) et voir si dans chaque échantillon la charge de sinistre est comparable ou c'est la même ou pas.

In [521…
```
base.Category.value_counts()
```

Out[521]:
```
Medium    36250
Large     32024
Small     31723
Name: Category, dtype: int64
```

In [254…
```
base.boxplot(column='chg_sin',by='Category') # via la dataframe
```

Out[254]:
```
<AxesSubplot:title={'center':'chg_sin'}, xlabel='Category'>
```



In [270…

```
gc = base.groupby('Category')
```

In [271…

```
# Anova version parametrique (hyposthe de normalité. Elle test la moyenne: est-ce que la charge sinistre moyenn
f_oneway(gc.get_group('Small').chg_sin,gc.get_group('Medium').chg_sin,gc.get_group('Large').chg_sin)
# pvalue=0.0 réjete Ho : la charge sinistre est different selon la categorie du vehicule
```

Out[271]:

```
F_onewayResult(statistic=7.854932644713429, pvalue=0.00038807355070197264)
```

In [272…

```
# version non parametrique
kruskal(gc.get_group('Small').chg_sin,gc.get_group('Medium').chg_sin,gc.get_group('Large').chg_sin)
#réjete H0
```

Out[272]:

```
KruskalResult(statistic=37.83296207867655, pvalue=6.090832656665146e-09)
```

car les échantillons sont differentes selon nos test. Alors, la charge de sinistre est correlée a la categorie du vehicule