Data Science pour l'Actuariat

Auteur: Ibrahima LY

PARTIE 1: Importation des bibliothèques et Préparation de l'environnement de travail

A- Importation des bibliothèques:

In [1]:
```
import os
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

B- Configuration du répertoire de travail:

In [2]:
```
#Affichage du répertoire courant (de travail actuel)
os.getcwd()
```

Out[2]:
```
'C:\\Users\\SCD UM\\Downloads\\Dossier Projet\\Notebook'
```

In [3]:
```
# Modification du répertoire courant (de travail)
os.chdir("C:\\Users\\SCD UM\\Downloads\\Dossier Projet\\BDD")
```

Chapitre 1 : Manipulation et prétraitement de données

Section 1 : Analyse et traitement du format de la base de données

A. Gestion des anomalies et QDD

Importation des données

In [4]:
```
base_sin = pd.read_csv("data_sin.csv", sep=";", decimal=",")
base_ptf = pd.read_excel("data_ptf.xlsx", sheet_name = "PTF")
base_expo = pd.read_excel("data_ptf.xlsx", sheet_name = "Expo")
```

In [5]:
```
base_sin.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13300 entries, 0 to 13299
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   nb_sin    13300 non-null  int64  
 1   chg_sin   13300 non-null  float64
 2   PolNum    13300 non-null  int64  
dtypes: float64(1), int64(2)
memory usage: 311.8 KB
```

In [6]:
```
base_ptf.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 [7]:

```python
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
```

1. Présentation des données et compréhension des données

In [8]:

```python
base_ptf2 = base_ptf.copy()
```

In [9]:

```python
base_ptf2.head()
```

Out[9]:

	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

2. Gestion des doublons

Nombre de doublons dans la colonne PolNum

In [10]:

```python
doublons = len(base_ptf2.PolNum) - base_ptf2.PolNum.nunique()
print("Nombre de doublons dans la colonne PolNum:", doublons)
```

```
Nombre de doublons dans la colonne PolNum: 27
```

OU

In [11]:

```python
sum(base_ptf2.duplicated(subset = "PolNum"))
```

Out[11]:

```
27
```

In [12]:

```python
# Indique les doublons avec 'True' à partir de la deuxième occurrence.
base_ptf2.duplicated()
```

Out[12]:

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

In [13]:

```python
# Compte les doublons dans le DataFrame.
```

```
sum(base_ptf2.duplicated())
```

Out[13]:

```
22
```

Supprimer les doublons, en gardant la première occurrence de chaque valeur de clé primaire

In [14]:

```
base_ptf2.drop_duplicates(subset="PolNum", keep='first', inplace=True)
```

In [15]:

```
base_ptf2.shape
```

Out[15]:

```
(100000, 15)
```

In [16]:

```
sum(base_ptf2.duplicated(subset = "PolNum"))
```

Out[16]:

```
0
```

In [17]:

```
sum(base_ptf2.duplicated())
```

Out[17]:

```
0
```

3. Gestion de données manquantes

Afficher le nombre de valeurs manquantes par colonne

In [18]:

```
missing_values = base_ptf2.isna().sum().sort_values(ascending=False)
print(" Nombre de valeurs manquantes par colonne:")
print(missing_values)
```

```
 Nombre de valeurs manquantes par colonne:
SubGroup2     88406
Value           785
Gender            5
PolNum            0
CalYear           0
Type              0
Category          0
Occupation        0
Age               0
Group1            0
Bonus             0
Poldur            0
Adind             0
Group2            0
Density           0
dtype: int64
```

In [19]:

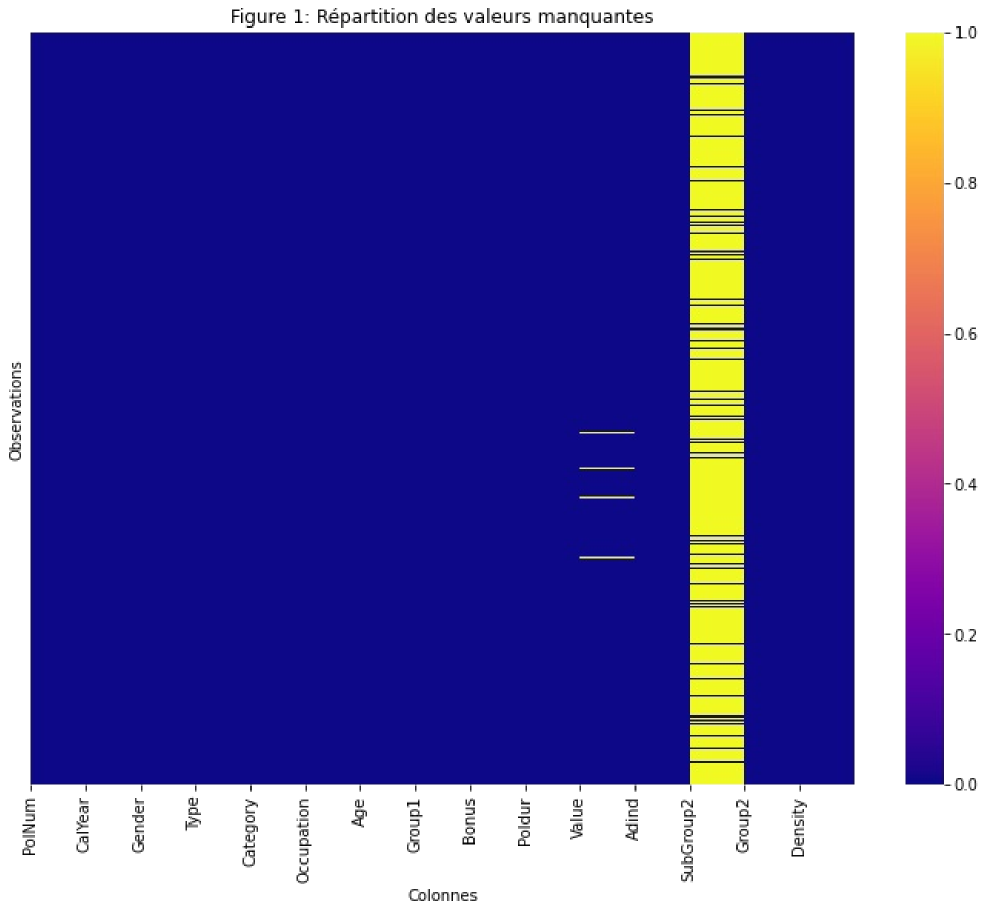
```
# Calcule le nombre de valeurs manquantes pour chaque colonne
missing_data = base_ptf2.isna().sum().sort_values(ascending=False)

# Calcule le pourcentage de valeurs manquantes pour chaque colonne
missing_percentage = (missing_data / len(base_ptf2) * 100).round(2)
missing_percentage
```

Out[19]:

```
SubGroup2     88.41
Value          0.78
Gender         0.00
PolNum         0.00
CalYear        0.00
Type           0.00
Category       0.00
Occupation     0.00
Age            0.00
Group1         0.00
Bonus          0.00
Poldur         0.00
Adind          0.00
Group2         0.00
Density        0.00
dtype: float64
```

In [20]:

```
plt.figure(figsize=(12,9))
sns.heatmap(base_ptf2.isna(), cmap="plasma", cbar=True, yticklabels=False)

plt.title("Figure 1: Répartition des valeurs manquantes")
plt.xticks(ticks=range(len(base_ptf2.columns)), labels=base_ptf2.columns, rotation=90)
plt.xlabel('Colonnes')
plt.ylabel('Observations')
plt.savefig("missing_values_heatmap.png", dpi=300)
plt.show()
```



In [21]:
```
unique_values_subgroup2 = base_ptf2["SubGroup2"].unique()
print(unique_values_subgroup2)
```

```
['P20' nan 'R34' 'Q63' 'R36' 'T19' 'Q34' 'Q22' 'L25' 'P12' 'Q41' 'R25'
 'L40' 'P19' 'L84' 'L99' 'U13' 'Q21' 'R35' 'Q12' 'Q62' 'R38' 'O19' 'R8'
 'Q20' 'L6' 'S2' 'N11' 'R17' 'M13' 'L71' 'Q60' 'Q46' 'P29' 'R14' 'R44'
 'R30' 'M15' 'O12' 'R28' 'R22' 'U8' 'Q45' 'U4' 'O1' 'Q36' 'U21' 'L104'
 'M1' 'M11' 'P27' 'N6' 'T10' 'L56' 'L125' 'T18' 'L102' 'M4' 'Q48' 'L52'
 'P14' 'L68' 'L4' 'L28' 'Q54' 'U17' 'L2' 'U2' 'M22' 'S29' 'Q18' 'L55'
 'R29' 'N22' 'O26' 'Q52' 'T26' 'T2' 'N26' 'N2' 'M14' 'N8' 'S10' 'L93'
 'O18' 'L87' 'L133' 'S30' 'R20' 'Q1' 'Q10' 'L64' 'Q6' 'L9' 'R23' 'T34'
 'L51' 'T13' 'L47' 'N25' 'R31' 'P17' 'Q66' 'Q32' 'S17' 'N12' 'L86' 'L44'
 'L92' 'P23' 'Q5' 'Q4' 'L16' 'R46' 'L128' 'T7' 'R37' 'S28' 'L124' 'L127'
 'L100' 'Q30' 'M18' 'O16' 'S25' 'R47' 'O6' 'O28' 'L31' 'P6' 'U16' 'T16'
 'U10' 'R13' 'Q19' 'R6' 'R41' 'M7' 'S15' 'L12' 'M12' 'P31' 'U9' 'Q26' 'Q3'
 'T6' 'R45' 'L106' 'Q47' 'M3' 'P34' 'L18' 'L82' 'T20' 'L94' 'R4' 'P36'
 'U5' 'L74' 'Q23' 'N23' 'N19' 'Q7' 'L7' 'L107' 'R43' 'P22' 'Q24' 'L96'
 'R26' 'T29' 'R16' 'Q61' 'M2' 'R10' 'T4' 'S3' 'O7' 'Q35' 'Q33' 'L3' 'Q2'
 'N24' 'S41' 'Q25' 'T28' 'M9' 'R48' 'T15' 'L123' 'N20' 'R21' 'L26' 'R24'
 'Q39' 'L69' 'L13' 'L79' 'L20' 'L105' 'Q9' 'T30' 'U11' 'L59' 'R15' 'L95'
 'R2' 'L46' 'L34' 'N16' 'M21' 'S8' 'Q14' 'Q37' 'Q13' 'Q56' 'M17' 'L53'
 'R49' 'L33' 'L23' 'O22' 'L134' 'S34' 'S23' 'L116' 'R39' 'R5' 'O15' 'L76'
 'L120' 'T33' 'P8' 'Q29' 'L11' 'L132' 'Q16' 'R27' 'L89' 'Q64' 'U19' 'L70'
 'Q51' 'P24' 'L30' 'O23' 'P21' 'Q28' 'L129' 'N21' 'L75' 'R32' 'L109' 'Q44'
 'Q57' 'L115' 'L24' 'L83' 'L21' 'P32' 'T27' 'Q38' 'M6' 'L45' 'L48' 'M19'
 'L39' 'S20' 'Q58' 'O2' 'O30' 'N13' 'L122' 'N17' 'S21' 'P7' 'S12' 'L61'
 'P30' 'L118' 'R19' 'Q49' 'S39' 'O32' 'N7' 'M20' 'Q55' 'O10' 'O9' 'N5'
 'U15' 'U12' 'M10' 'L114' 'S7' 'T25' 'O11' 'N10' 'R18' 'L131' 'O37' 'L10'
 'R1' 'U1' 'L29' 'P16' 'L8' 'T1' 'U18' 'L110' 'L101' 'L91' 'R9' 'T23'
 'S37' 'L90' 'O3' 'T22' 'O35' 'O8' 'O29' 'N9' 'Q59' 'Q31' 'Q43' 'L5' 'L35'
 'T8' 'L67' 'Q50' 'S4' 'S27' 'S32' 'L41' 'S19' 'R7' 'L14' 'P35' 'U14'
 'R50' 'L85' 'N4' 'Q65' 'P3' 'P13' 'O34' 'P15' 'R12' 'S1' 'Q11' 'R33' 'S9'
 'L27' 'L80' 'L73' 'R42' 'T21' 'R11' 'Q42' 'L19' 'S22' 'L54' 'P26' 'O38'
 'N3' 'Q53' 'L43' 'O24' 'O27' 'O17' 'L117' 'M8' 'L72' 'L108' 'L57' 'S35'
 'T9' 'L17' 'Q8' 'N14' 'L63' 'Q15' 'U20' 'R40' 'L126' 'S18' 'N1' 'T17'
 'S40' 'P11' 'P28' 'L103' 'L130' 'S33' 'O36' 'L135' 'U7' 'O13' 'O5' 'R3'
 'L112' 'Q40' 'L88' 'S13' 'S6' 'P33' 'L37' 'T24' 'P18' 'M16' 'L1' 'T14'
 'L66' 'L32' 'M5' 'L98' 'U6' 'N15' 'O39' 'P2' 'S14' 'L119' 'S5' 'L36' 'U3'
 'S11' 'O25' 'O21' 'L81' 'L97' 'P10' 'P4' 'L38' 'Q27' 'O14' 'L111' 'L49'
 'S31' 'N18' 'O33' 'T32' 'O20' 'L65' 'S38' 'P9' 'L15' 'T12' 'P5' 'Q17'
 'L60' 'L62' 'P25' 'L42' 'L121' 'S16' 'T3' 'O4' 'L78' 'L58' 'L50' 'T11'
 'S36' 'P1' 'L77' 'O31' 'T5' 'L113' 'S26' 'S24' 'T31' 'L136' 'L22']
```

In [22]:
```
unique_values_subgroup2 = base_ptf2["SubGroup2"].value_counts()
print(unique_values_subgroup2)
```

```
M17    59
Q29    56
Q36    54
Q34    53
M22    53
       ..
S38     8
S35     8
O13     8
S8      8
L22     7
Name: SubGroup2, Length: 471, dtype: int64
```

Suppression de la Colonne SubGroup2

In [23]:
```
base_ptf.drop(columns=['SubGroup2'], inplace=True)
```

In [24]:
```
base_ptf.columns
```

Out[24]:
```
Index(['PolNum', 'CalYear', 'Gender', 'Type', 'Category', 'Occupation', 'Age',
       'Group1', 'Bonus', 'Poldur', 'Value', 'Adind', 'Group2', 'Density'],
      dtype='object')
```

In [25]:
```
uniqueValue = base_ptf2["Value"].unique()
print(uniqueValue)
```

```
[8590 27445 11290 ... 48025 47190 36615]
```

In [26]:
```
base_ptf2["Value"].value_counts()
```

Out[26]:
```
9385     56
8015     54
8245     51
9665     50
9175     50
         ..
46285     1
39430     1
43095     1
45820     1
36615     1
Name: Value, Length: 9383, dtype: int64
```

Identifiaction des valeur non numérique dans la colonne "Value"

In [27]:
```
non_numeric_values = base_ptf[~base_ptf['Value'].apply(lambda x: str(x).replace('.', '', 1).isdigit())]['Value'
print("Valeurs non numériques uniques dans la colonne 'Value':", non_numeric_values.unique())
print()
print("Indices des valeurs non numériques dans la colonne 'Value':", non_numeric_values.index)
```

```
Valeurs non numériques uniques dans la colonne 'Value': ['??' nan]

Indices des valeurs non numériques dans la colonne 'Value': Int64Index([  103,   182,   229,   541,   687,  106
2,  1130,  1137,  1296,
             1309,
            ...
            98824, 99164, 99191, 99209, 99308, 99596, 99639, 99644, 99703,
            99986],
           dtype='int64', length=786)
```

In [28]:
```
base_ptf[base_ptf.loc[:,"Value"]=="??"]
```

Out[28]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density
103	200114974	2009	Female	A	Medium	Employed	18	7	0	4	??	0	U	103.949399

In [29]:
```
base_ptf.Value.replace({'??': np.nan},inplace=True)
```

In [30]:
```
base_ptf['Value'] = pd.to_numeric(base_ptf["Value"])
```

In [31]:
```
median_value = base_ptf['Value'].median()
base_ptf['Value'].fillna(median_value, inplace=True)
```

Suppression des NaN dans "Gender"

In [32]:
```
base_ptf.dropna(subset=['Gender'], inplace=True)
```

In [33]:
```
base_ptf.isna().sum()
```

Out[33]:
```
PolNum        0
CalYear       0
Gender        0
Type          0
Category      0
Occupation    0
Age           0
Group1        0
Bonus         0
Poldur        0
Value         0
Adind         0
Group2        0
Density       0
dtype: int64
```

4. Analyse et Correction des Données Incohérentes

In [34]:
```
base_ptf.columns
```

Out[34]:
```
Index(['PolNum', 'CalYear', 'Gender', 'Type', 'Category', 'Occupation', 'Age',
       'Group1', 'Bonus', 'Poldur', 'Value', 'Adind', 'Group2', 'Density'],
      dtype='object')
```

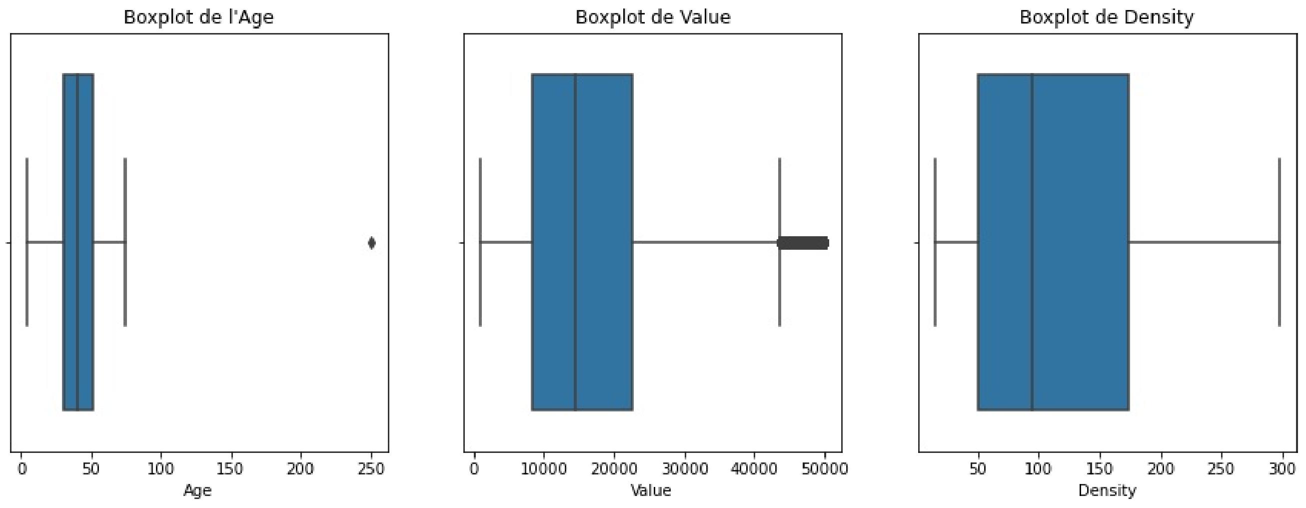
In [35]:
```
base_ptf[["Age","Value","Density"]].describe()
```

Out[35]:

	Age	Value	Density
count	100022.000000	100022.000000	100022.000000
mean	41.123463	16440.235398	117.160264
std	14.315898	10466.567424	79.500672
min	4.000000	1000.000000	14.377142
25%	30.000000	8410.000000	50.625783
50%	40.000000	14610.000000	94.364623
75%	51.000000	22515.000000	174.644525
max	250.000000	49995.000000	297.385170

Visualisation des valeurs Abérentes: Boîtes à moustaches (Boxplot)

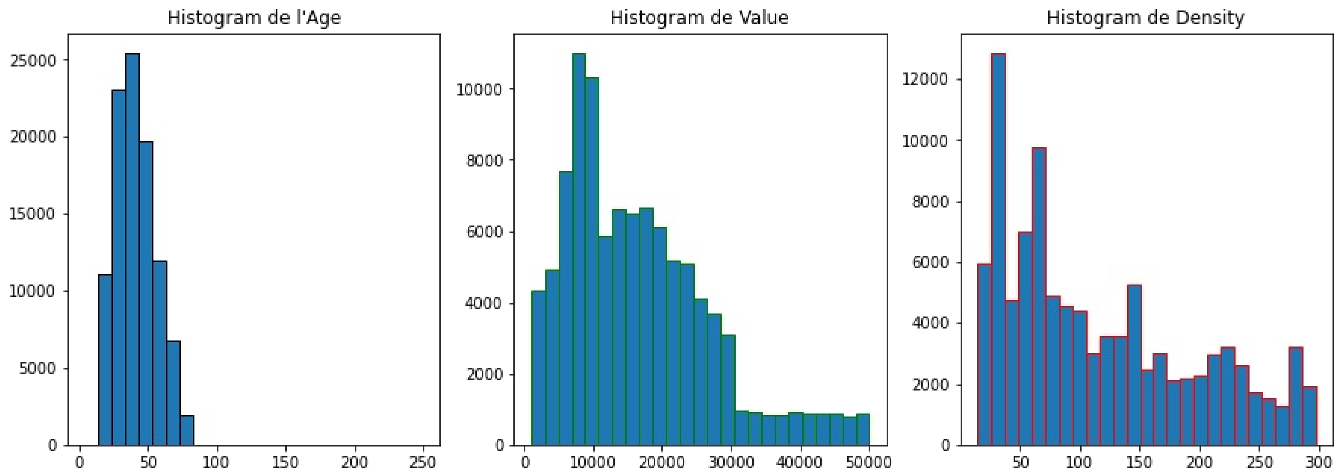
In [36]:
```
# Boîtes à moustaches pour Age, Value, Density
plt.figure(figsize=[15,5])

plt.subplot(1,3,1)
sns.boxplot(x=base_ptf["Age"])
plt.title("Boxplot de l'Age")

plt.subplot(1,3,2)
sns.boxplot(x=base_ptf["Value"])
plt.title("Boxplot de Value")

plt.subplot(1,3,3)
sns.boxplot(x=base_ptf["Density"])
plt.title("Boxplot de Density")
plt.savefig('boxplots.png')
plt.show()
```



Visualisation des valeurs Abérentes: Histogrammes

In [37]:
```
plt.figure(figsize=[15,5])
```

```python
plt.subplot(1,3,1)
plt.hist(base_ptf["Age"], bins=25, edgecolor='black')
plt.title("Histogram de l'Age")

plt.subplot(1,3,2)
plt.hist(base_ptf["Value"], bins=25, edgecolor='Green')
plt.title("Histogram de Value")

plt.subplot(1,3,3)
plt.hist(base_ptf["Density"], bins=25, edgecolor='red')
plt.title("Histogram de Density")
plt.savefig('histograms.png')
plt.show()
```



Variables catégorielles/ dichotomiques

- Variable 'Genre': **genre du conducteur**

In [38]:
```python
base_ptf['Gender'].value_counts()
```

Out[38]:
```
Male      63437
Female    36574
H             5
F             5
h             1
Name: Gender, dtype: int64
```

In [39]:
```python
base_ptf.Gender.unique()
```

Out[39]:
```
array(['Male', 'Female', 'H', 'F', 'h'], dtype=object)
```

- Variable 'Type': **type de vehicule**

In [40]:
```python
base_ptf['Type'].value_counts()
```

Out[40]:
```
A    27760
B    22090
D    19597
C    13864
E    11170
F     5541
Name: Type, dtype: int64
```

- Variable 'Catégorie': **categorie du vehicule**

In [41]:
```python
base_ptf['Category'].value_counts()
```

Out[41]:
```
Medium    36253
Large     32022
Small     31718
???          29
Name: Category, dtype: int64
```

- Variable 'Occupation': **profession**

In [42]:
```python
base_ptf['Occupation'].value_counts()
```

Out[42]:
```
Employed         31149
Self-employed    20372
Housewife        20012
Unemployed       15322
Retired          13167
Name: Occupation, dtype: int64
```

- Variable 'Group2': **Region d'habitation**

In [43]:
```
base_ptf['Group2'].value_counts()
```

Out[43]:
```
L    23730
Q    22389
R    15081
M     7596
U     5365
P     5259
O     5216
T     5197
N     5195
S     4994
Name: Group2, dtype: int64
```

- Variable 'Adind': **Indicateur d'une garantie dommages**

In [44]:
```
base_ptf['Adind'].value_counts()
```

Out[44]:
```
1    51225
0    48797
Name: Adind, dtype: int64
```

Correction des Données Incohérentes

- Variable 'Age': **age du conducteur**

In [45]:
```
#selection des lignes cohérentes (supression des anomalies)
base_ptf = base_ptf[(base_ptf.Age>=18)&(base_ptf.Age<=80)];base_ptf
```

Out[45]:

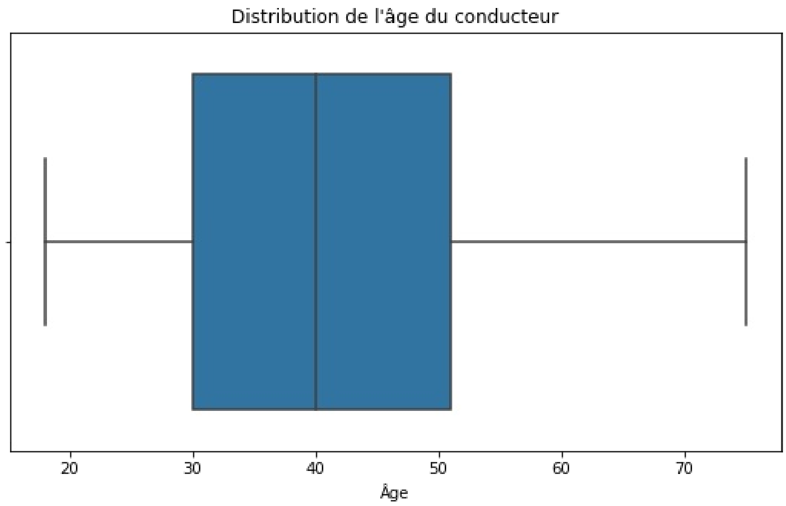
	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density
0	200114871	2009	Male	C	Small	Self-employed	27	3	-20	0	8590.0	0	P	43.843798
1	200114872	2009	Female	E	Large	Unemployed	60	20	-30	0	27445.0	0	L	66.066684
2	200114873	2009	Female	D	Medium	Housewife	62	13	-30	9	11290.0	1	R	276.335565
3	200114874	2009	Female	B	Large	Employed	27	16	50	3	26985.0	0	T	30.462442
4	200114875	2009	Male	F	Large	Housewife	37	16	80	3	39705.0	1	R	285.621744
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
100022	200285801	2010	Male	F	Medium	Housewife	45	11	30	0	19700.0	0	L	76.052726
100023	200285802	2010	Male	E	Medium	Retired	53	8	-30	6	10980.0	1	U	61.794759
100024	200285803	2010	Male	C	Large	Employed	47	10	-10	9	21980.0	0	L	45.669823
100025	200285804	2010	Female	D	Large	Retired	46	7	-50	1	28925.0	1	U	54.931812
100026	200285805	2010	Female	C	Medium	Retired	67	17	-50	9	14525.0	1	L	73.252499

100019 rows × 14 columns

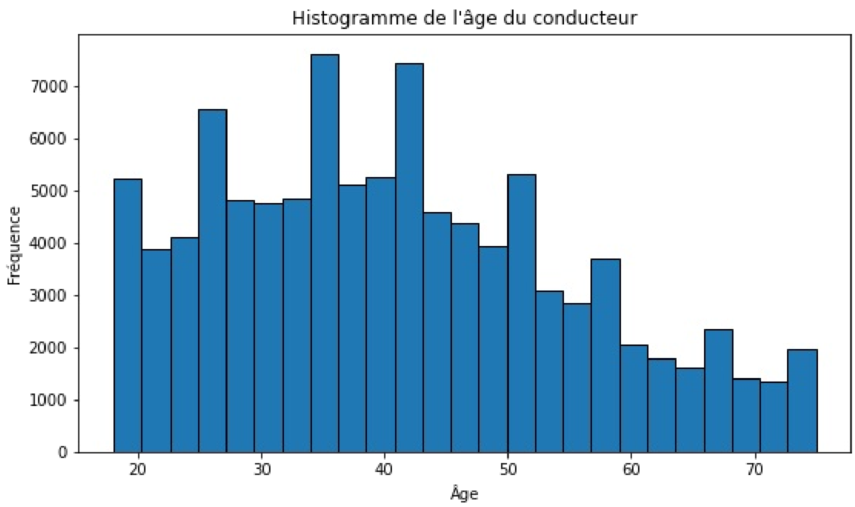
In [46]:
```
base_ptf['Age'].describe()
```

Out[46]:
```
count    100019.000000
mean         41.122057
std          14.300049
min          18.000000
25%          30.000000
50%          40.000000
75%          51.000000
max          75.000000
Name: Age, dtype: float64
```

In [47]:
```
plt.figure(figsize=[9,5])
sns.boxplot(x='Age', data=base_ptf)
plt.title('Distribution de l\'âge du conducteur')
plt.xlabel('Âge')
plt.savefig('Distribution.png')
plt.show()
```




```
In [48]: plt.figure(figsize=[9,5])
         plt.hist(base_ptf['Age'], bins=25, edgecolor='black')
         plt.title('Histogramme de l\'âge du conducteur')
         plt.xlabel('Âge')
         plt.ylabel('Fréquence')
         plt.savefig('Histogramme.png')
         plt.show()
```



- Vatiable 'Gender': **Genre du conducteur**

```
In [49]: base_ptf['Gender'].value_counts()
```

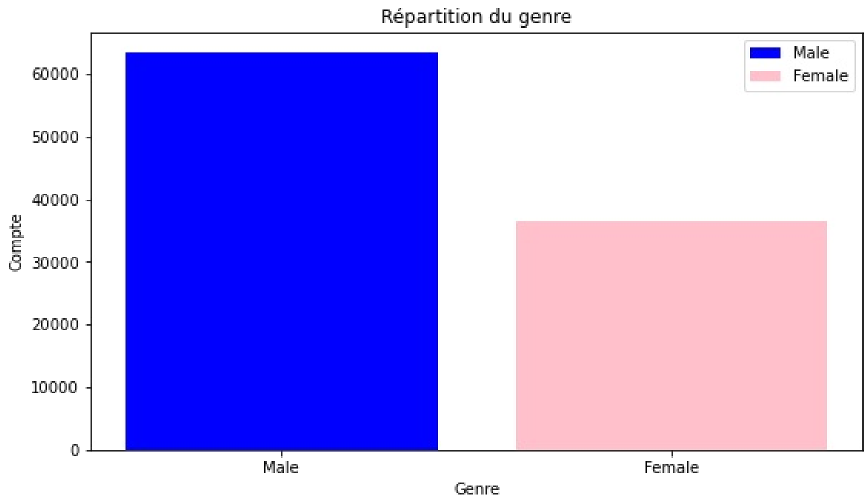
```
Out[49]: Male      63435
         Female    36573
         H             5
         F             5
         h             1
         Name: Gender, dtype: int64
```

```
In [50]: base_ptf['Gender'].replace({"H": "Male", "h": "Male", "F": "Female"}, inplace=True)
```

```
In [51]: base_ptf['Gender'].value_counts()
```

```
Out[51]: Male      63441
         Female    36578
         Name: Gender, dtype: int64
```

```
In [52]: gender_counts = base_ptf['Gender'].value_counts()
         gender_counts = gender_counts[['Male', 'Female']]
         colors = ['blue', 'pink']
         labels = ['Male', 'Female']

         plt.figure(figsize=[9,5])

         for i, (gender, count) in enumerate(gender_counts.items()):
             plt.bar(gender, count, color=colors[i], label=labels[i])

         plt.title('Répartition du genre')
         plt.xlabel('Genre')
         plt.ylabel('Compte')
         plt.legend()
         plt.savefig('gender.png')
         plt.show()
```



- Variable **Catégorie du véhicule :**

In [53]:
```
base_ptf['Category'].value_counts()
```

Out[53]:
```
Medium    36253
Large     32020
Small     31717
???          29
Name: Category, dtype: int64
```

In [54]:
```
# Obtenir les catégories valides (c'est-à-dire, exclure "???")
categories_valides = base_ptf['Category'][base_ptf['Category'] != "???"]
# Calculer les proportions des catégories valides
proportions = categories_valides.value_counts(normalize=True);proportions
```

Out[54]:
```
Medium    0.362566
Large     0.320232
Small     0.317202
Name: Category, dtype: float64
```

Identifier les lignes aberrantes: Trouver les lignes où la catégorie est "???"

In [55]:
```
# Identifier les lignes ayant la valeur aberrante "???"
lignes_manquantes = base_ptf['Category'] == "???";lignes_manquantes
```

Out[55]:
```
0         False
1         False
2         False
3         False
4         False
          ...
100022    False
100023    False
100024    False
100025    False
100026    False
Name: Category, Length: 100019, dtype: bool
```

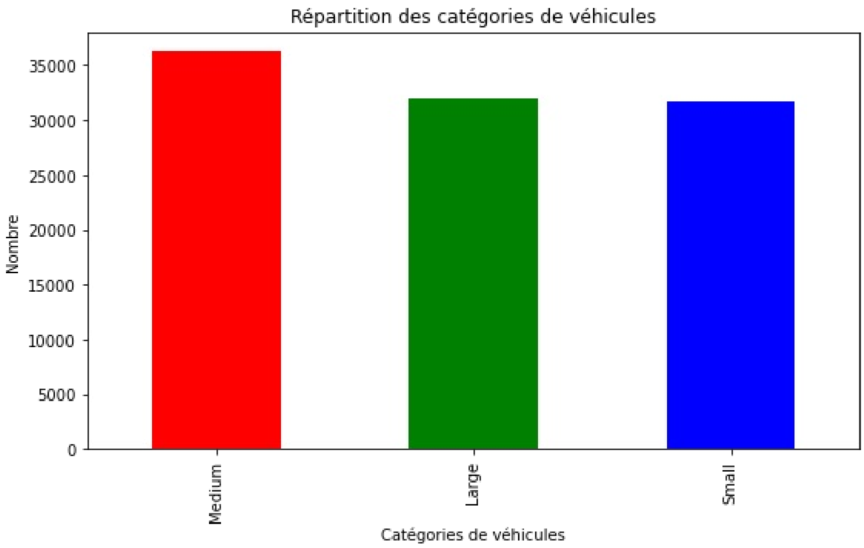
Choisir aléatoirement des catégories en fonction des proportions

In [56]:
```
valeurs_imputees = np.random.choice(proportions.index, size=lignes_manquantes.sum(), p=proportions.values)
# Remplacer les valeurs aberrantes par les valeurs imputées
base_ptf.loc[lignes_manquantes, 'Category'] = valeurs_imputees
```

In [57]:
```
base_ptf['Category'].value_counts()
```

Out[57]:
```
Medium    36263
Large     32032
Small     31724
Name: Category, dtype: int64
```

In [58]:
```
colors = ['red', 'green', 'blue']
plt.figure(figsize=[9,5])
base_ptf['Category'].value_counts().plot(kind='bar', color=colors)
plt.title('Répartition des catégories de véhicules')
plt.xlabel('Catégories de véhicules')
plt.ylabel('Nombre')
plt.savefig('category.png')
plt.show()
```



Base Sinistre:

In [59]:
```
base_sin.head(5)
```

Out[59]:

	nb_sin	chg_sin	PolNum
0	1	0.0	200114978
1	1	0.0	200114994
2	2	0.0	200115001
3	1	0.0	200115011
4	2	0.0	200115015

In [60]:
```
#1. Vérification des Doublons
if base_sin['PolNum'].duplicated().any():
    print("Il y a des doublons dans la colonne PolNum.")
else:
    print("Pas de doublons dans la colonne PolNum.")
```

```
Il y a des doublons dans la colonne PolNum.
```

In [61]:
```
len(base_sin) - base_sin.PolNum.nunique()
```

Out[61]: 99

In [62]:
```
base_sin[base_sin.duplicated('PolNum',keep=False)].sort_values('PolNum')
```

Out[62]:

	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 [63]:
```
base_sin = base_sin.groupby(['PolNum']).sum();base_sin
```

Out[63]:

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 [64]:
```
#Vérification des Valeurs NaN
if base_sin.isnull().any().any():
    print("Il y a des valeurs NaN dans la base.")
else:
    print("Pas de valeurs NaN dans la base.")
```

```
Pas de valeurs NaN dans la base.
```

base Exposition

In [65]:
```
# Vérification des Doublons:
if base_expo['PolNum'].duplicated().any():
    print("Il y a des doublons dans la colonne PolNum.")
else:
    print("Pas de doublons dans la colonne PolNum.")
```

```
Il y a des doublons dans la colonne PolNum.
```

In [66]:
```
sum(base_expo.duplicated())
```

Out[66]:
```
21
```

In [67]:
```
base_expo = base_expo[~base_expo.duplicated()];base_expo
```

Out[67]:

	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

100000 rows × 2 columns

In [68]:
```
sum(base_expo.duplicated())
```

Out[68]:
```
0
```

In [69]:
```
base_expo.shape
```

Out[69]:
```
(100000, 2)
```

In [70]:
```
# Vérification des Valeurs NaN:
if base_expo.isnull().any().any():
    print("Il y a des valeurs NaN dans la base.")
else:
    print("Pas de valeurs NaN dans la base.")
```

```
Pas de valeurs NaN dans la base.
```

C. Jointure des bases

In [71]:
```python
# Fusion/jointure première étape
base_version1 = pd.merge(base_ptf, base_expo, on=['PolNum'], how='inner')

# Fusion/jointure deuxième étape
base_Finale = pd.merge(base_version1, base_sin, on="PolNum", how='inner')
```

In [72]:
```python
base_Finale
```

Out[72]:

	PolNum	CalYear	Gender	Type	Category	Occupation	Age	Group1	Bonus	Poldur	Value	Adind	Group2	Density	Expday
0	200114878	2009	Female	A	Large	Housewife	41	10	100	0	24940.0	1	R	272.966995	36
1	200114880	2009	Male	B	Large	Unemployed	25	3	40	12	48945.0	0	M	190.051565	13
2	200114890	2009	Female	F	Small	Employed	29	17	30	7	1525.0	0	R	225.043089	36
3	200114894	2009	Male	A	Medium	Self-employed	47	17	20	12	18480.0	1	M	129.419475	36
4	200114895	2009	Female	C	Small	Employed	47	11	-10	7	8690.0	0	R	290.132719	36
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
12272	200285737	2010	Male	E	Medium	Unemployed	25	9	60	1	10265.0	0	U	94.657516	36
12273	200285745	2010	Male	A	Large	Housewife	54	10	30	1	21610.0	0	R	250.841326	36
12274	200285756	2010	Male	A	Small	Employed	22	2	-10	11	6910.0	1	R	295.797092	36
12275	200285769	2010	Male	A	Medium	Employed	51	13	-30	0	11955.0	1	P	24.826528	19
12276	200285791	2010	Male	D	Medium	Self-employed	21	15	50	1	12100.0	1	R	259.004060	36

12277 rows × 17 columns

In [73]:
```python
# Vérification des Valeurs NaN:
if base_Finale.isnull().any().any():
    print("Il y a des valeurs NaN dans la base.")
else:
    print("Pas de valeurs NaN dans la base.")
```

```
Pas de valeurs NaN dans la base.
```

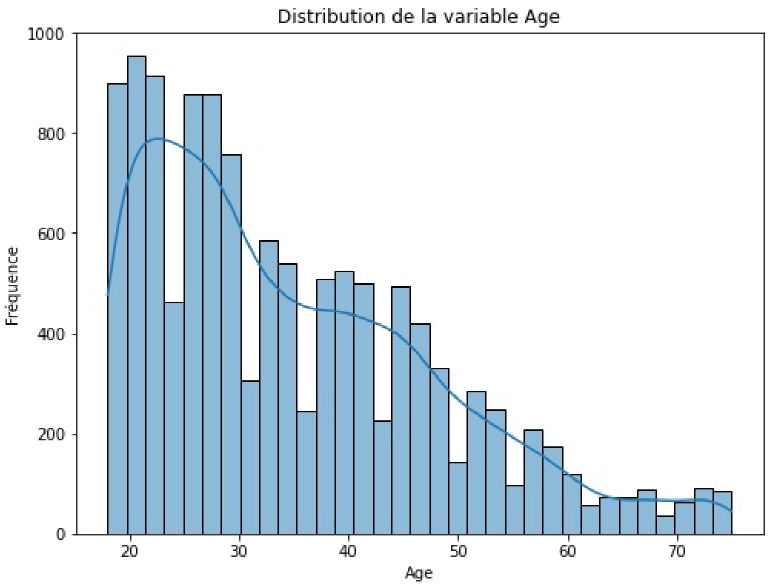
Section 2 : Analyse Statistiques des Données

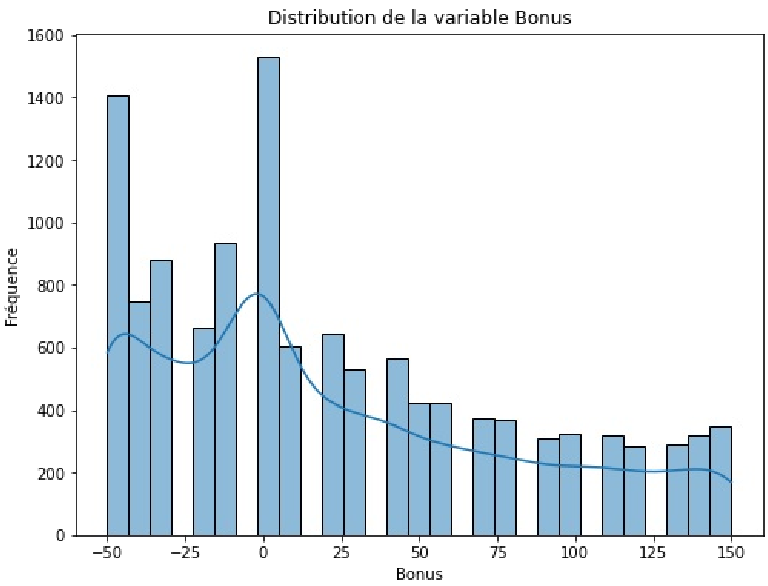
Variables Numeriques: 'Age', 'Bonus', 'Value'

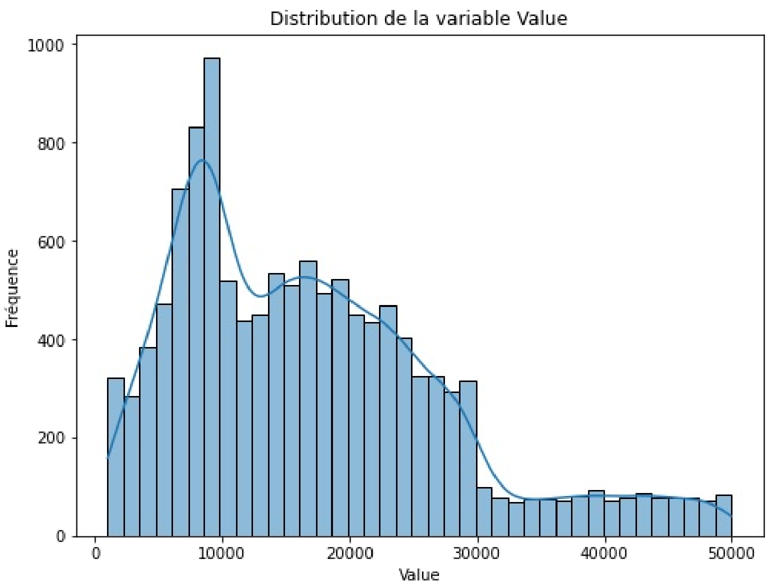
In [74]:
```python
quantitative_variables = ['Age', 'Bonus', 'Value']
description = base_Finale[quantitative_variables].describe().T
description['Skewness'] = base_Finale[quantitative_variables].skew()
print(description)
```

```
         count          mean           std     min     25%      50%      75%  \
Age    12277.0     34.972062     13.298235    18.0    24.0     32.0     44.0
Bonus  12277.0     22.283131     57.690639   -50.0   -20.0      0.0     60.0
Value  12277.0  17088.835628  10721.573527  1005.0  8625.0  15410.0  23225.0

           max  Skewness
Age       75.0  0.846536
Bonus    150.0  0.654603
Value  49995.0  0.919374
```

In [75]:
```python
quantitative_variables = ['Age', 'Bonus', 'Value']

for var in quantitative_variables:
    plt.figure(figsize=(8, 6))
    sns.histplot(base_Finale[var], kde=True)
    plt.title(f'Distribution de la variable {var}')
    plt.xlabel(var)
    plt.ylabel('Fréquence')
    filename = f'Frequence_{var}.png'
    plt.savefig(filename)
    plt.show()
```

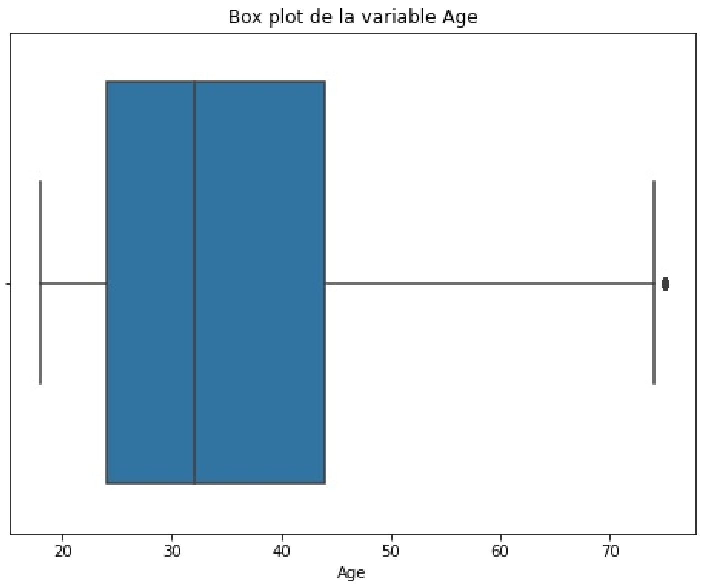


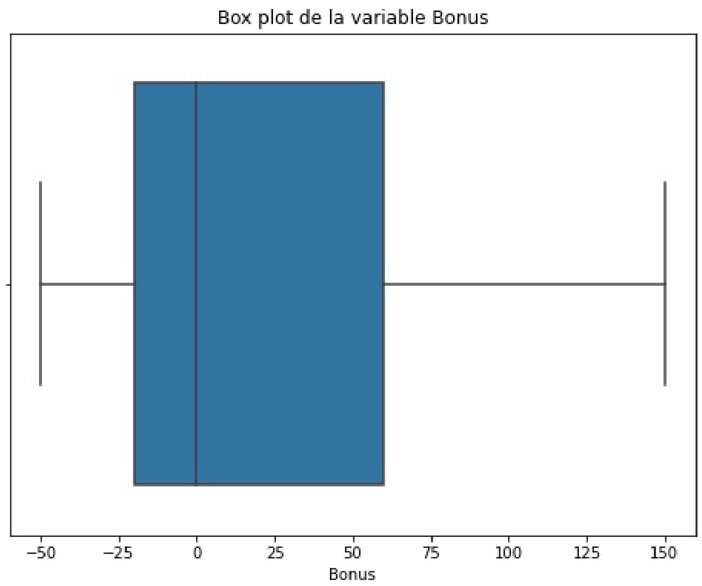


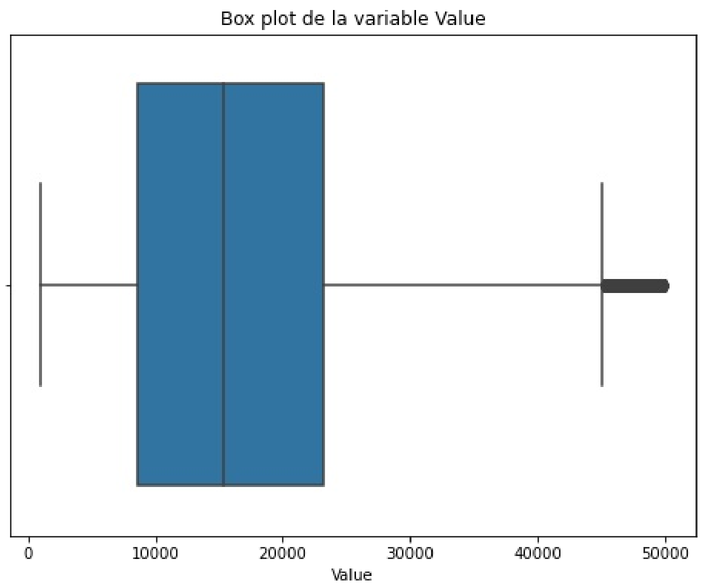


In [76]:

```
quantitative_variables = ['Age', 'Bonus', 'Value']

for var in quantitative_variables:
    plt.figure(figsize=(8,6))
    sns.boxplot(x=base_Finale[var])
    plt.title(f'Box plot de la variable {var}')
    filename = f'BoxPlot_{var}.png'
    plt.savefig(filename)
    plt.show()
```







In [77]:

```
categorical_variables = ['Gender', 'Type', 'Category', 'Occupation']

for var in categorical_variables:
    frequency_table = base_Finale[var].value_counts(normalize=True).reset_index()
    frequency_table.columns = [var, 'Fréquence']
    frequency_table['Pourcentage'] = frequency_table['Fréquence'] * 100
    print(f"Table de fréquence pour {var} :")
    print(frequency_table)
```

```
Table de fréquence pour Gender :
   Gender  Fréquence  Pourcentage
0    Male   0.680541    68.054085
1  Female   0.319459    31.945915
Table de fréquence pour Type :
  Type  Fréquence  Pourcentage
0    A   0.244604    24.460373
1    D   0.212104    21.210393
2    B   0.207624    20.762401
3    C   0.139936    13.993647
4    E   0.128370    12.837012
5    F   0.067362     6.736173
Table de fréquence pour Category :
  Category  Fréquence  Pourcentage
0   Medium   0.358557    35.855665
1    Large   0.343406    34.340637
2    Small   0.298037    29.803698
Table de fréquence pour Occupation :
      Occupation  Fréquence  Pourcentage
0       Employed   0.350737    35.073715
1      Housewife   0.238087    23.808748
2     Unemployed   0.203307    20.330700
3  Self-employed   0.170074    17.007412
4        Retired   0.037794     3.779425
```

In [78]:

```
from scipy.stats import shapiro

for var in quantitative_variables:
    stat, p = shapiro(base_Finale[var])
    print(f'Test de Shapiro-Wilk pour {var}: Statistique = {stat}, p-valeur = {p}')
```

```
Test de Shapiro-Wilk pour Age: Statistique = 0.9258148074150085, p-valeur = 0.0
Test de Shapiro-Wilk pour Bonus: Statistique = 0.919397234916687, p-valeur = 0.0
Test de Shapiro-Wilk pour Value: Statistique = 0.9326664805412292, p-valeur = 0.0
```

In [79]:

```
# Sélection des variables pertinentes
variables_to_analyze = ['Age', 'Bonus', 'Value', 'Type', 'Category', 'Occupation', 'Group1', 'Group2', 'Density
correlation_data = base_Finale[variables_to_analyze]

# Calcul de la matrice de corrélation
correlation_matrix = correlation_data.corr()
```

In [80]:

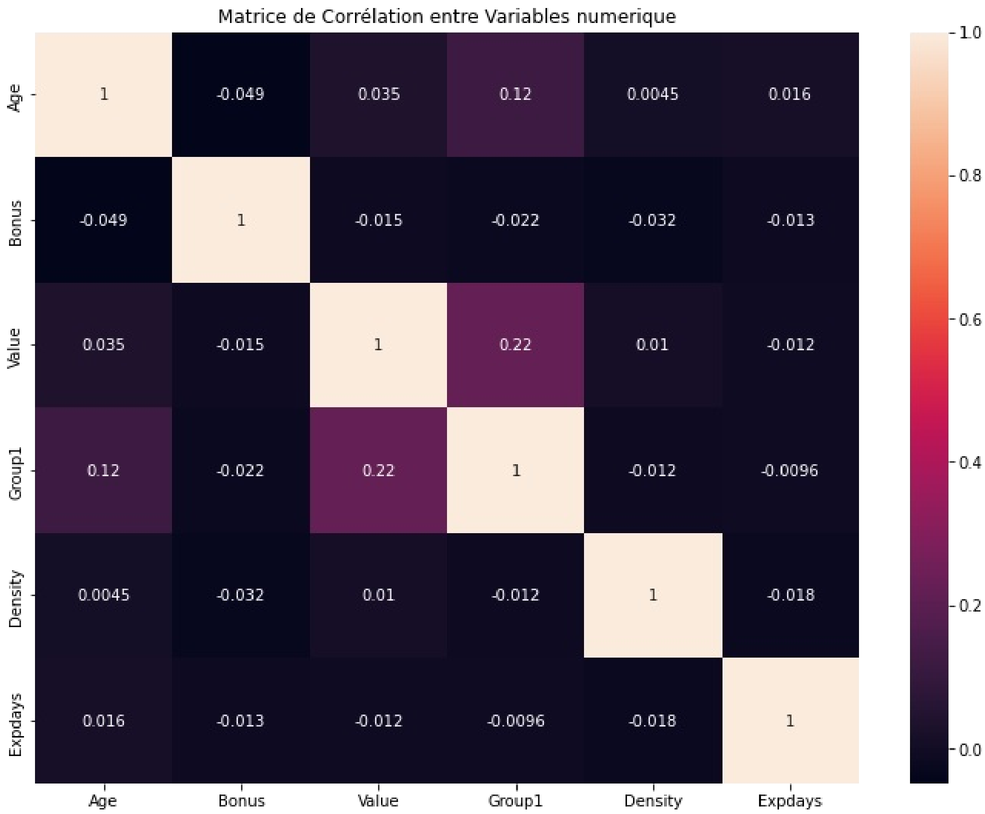
```
correlation_matrix
```

Out[80]:

	Age	Bonus	Value	Group1	Density	Expdays
Age	1.000000	-0.048884	0.035300	0.120692	0.004450	0.016496
Bonus	-0.048884	1.000000	-0.015083	-0.022038	-0.031734	-0.013351
Value	0.035300	-0.015083	1.000000	0.223247	0.010053	-0.011643
Group1	0.120692	-0.022038	0.223247	1.000000	-0.012441	-0.009625
Density	0.004450	-0.031734	0.010053	-0.012441	1.000000	-0.017617
Expdays	0.016496	-0.013351	-0.011643	-0.009625	-0.017617	1.000000

In [81]:

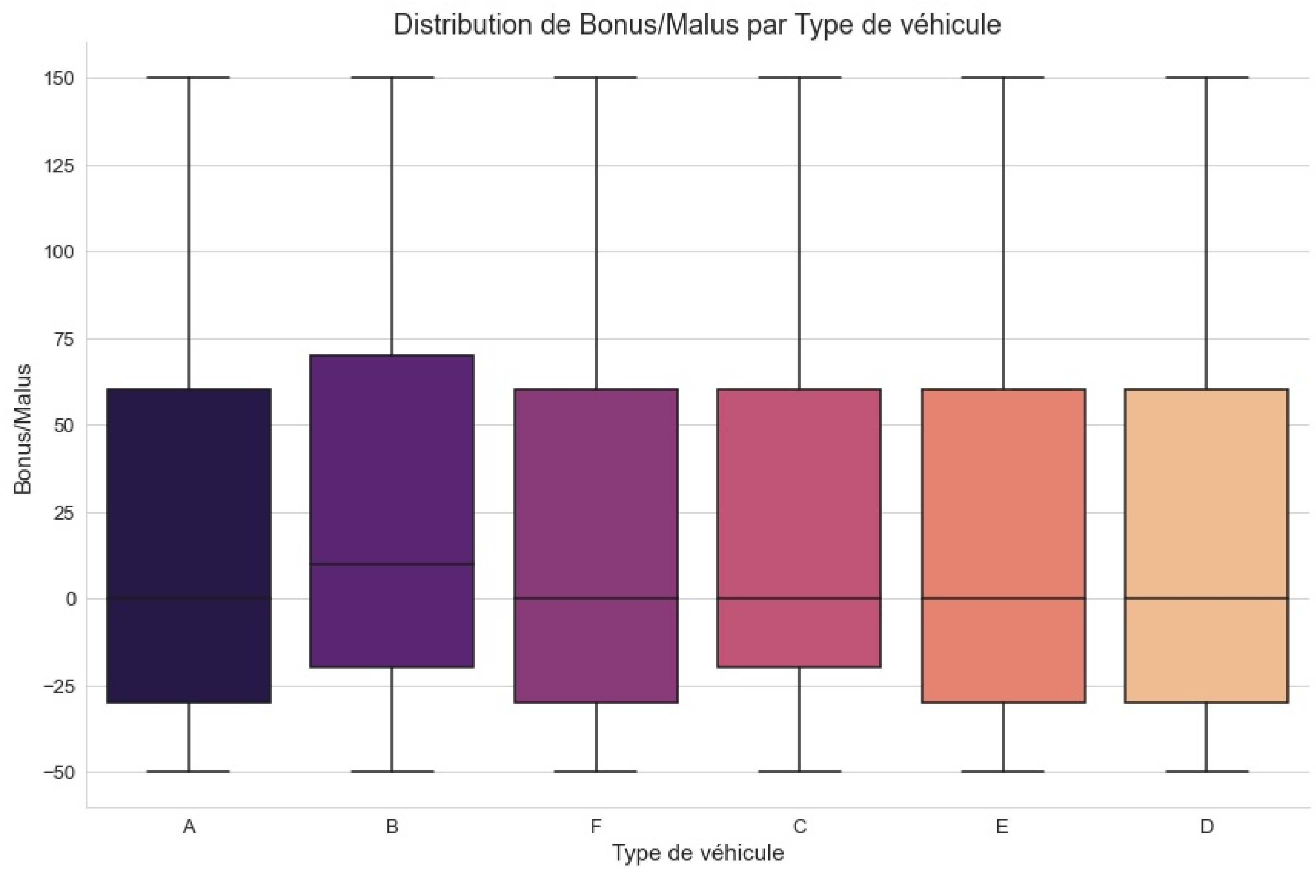
```
# Affichage de la carte thermique
plt.figure(figsize=(12, 9))
sns.heatmap(correlation_matrix, annot=True)
plt.title('Matrice de Corrélation entre Variables numerique')
plt.savefig('matrice_de_corelation.png')
plt.show()
```



In [82]:
```python
# Réalisation du test ANOVA
from scipy import stats
f_value, p_value = stats.f_oneway(base_Finale['Bonus'][base_Finale['Type'] == 'A'],
                                  base_Finale['Bonus'][base_Finale['Type'] == 'B'],
                                  base_Finale['Bonus'][base_Finale['Type'] == 'C'],
                                  )

print("F-value:", f_value)
print("P-value:", p_value)
```

```
F-value: 3.1213512185596772
P-value: 0.04415666986654134
```

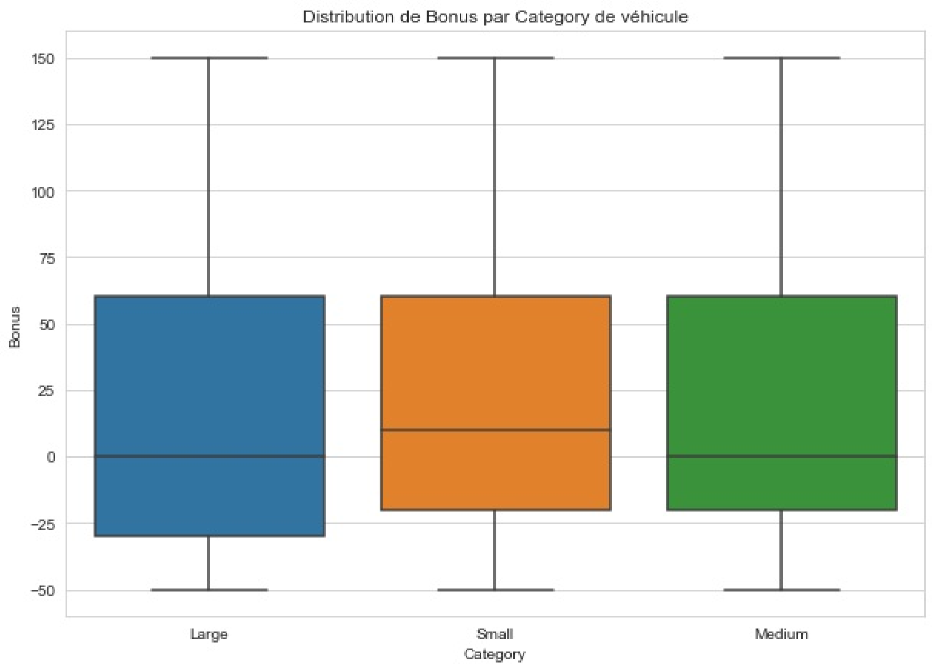
In [83]:
```python
# Configuration de style
sns.set_style("whitegrid")

# Création du boxplot
plt.figure(figsize=(12, 8))
sns.boxplot(x='Type', y='Bonus', data=base_Finale, palette="magma")

# Titre et étiquettes
plt.title("Distribution de Bonus/Malus par Type de véhicule", fontsize=18)
plt.xlabel('Type de véhicule', fontsize=15)
plt.ylabel('Bonus/Malus', fontsize=15)
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)

# Éliminer la bordure supérieure et droite pour une meilleure esthétique
sns.despine()

# Sauvegarde en haute qualité
plt.tight_layout()
plt.savefig('distribution_bonus_type_amélioré.png', dpi=300)

# Afficher le graphique
plt.show()
```



In [84]:
```
f_value_category, p_value_category = stats.f_oneway(base_Finale['Bonus'][base_Finale['Category'] == 'Large'],
                                                     base_Finale['Bonus'][base_Finale['Category'] == 'Small'],
                                                     base_Finale['Bonus'][base_Finale['Category'] == 'Medium'])
print("F-value for Category:", f_value_category)
print("P-value for Category:", p_value_category)
```

```
F-value for Category: 1.0924169178774714
P-value for Category: 0.33543747567569154
```

In [85]:
```
plt.figure(figsize=(10, 7))
sns.boxplot(x='Category', y='Bonus', data=base_Finale)
plt.title("Distribution de Bonus par Category de véhicule")
plt.savefig('distribution_bonus_Category.png')
plt.show()
```

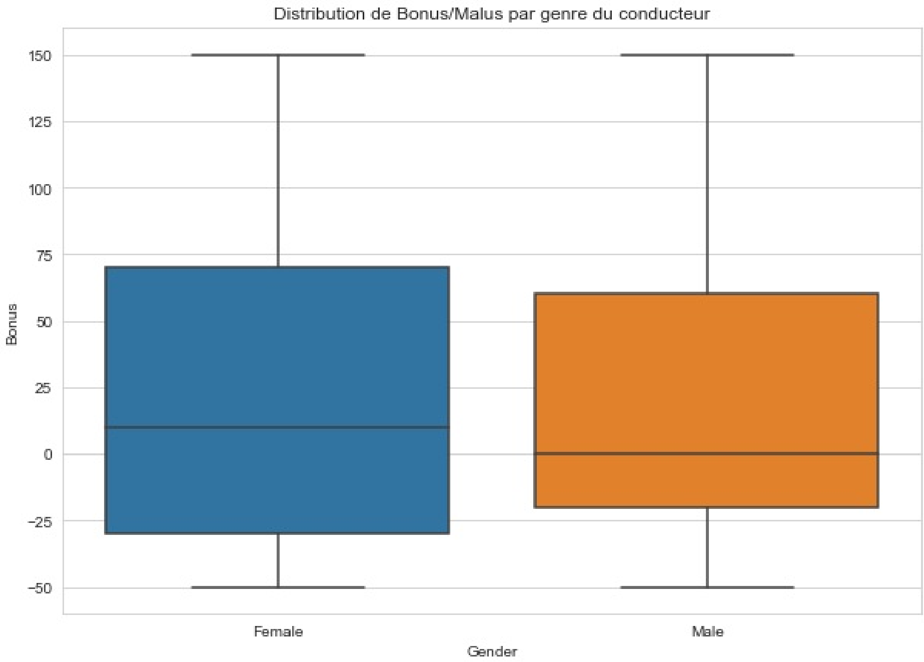


In [86]:
```
f_value_gender, p_value_gender = stats.f_oneway(base_Finale['Bonus'][base_Finale['Gender'] == 'Female'],
                                                 base_Finale['Bonus'][base_Finale['Gender'] == 'Male'])
print("F-value for Gender:", f_value_gender)
print("P-value for Gender:", p_value_gender)
```

```
F-value for Gender: 6.985574116655018
P-value for Gender: 0.008227363945318618
```

In [87]:
```
plt.figure(figsize=(10, 7))
```

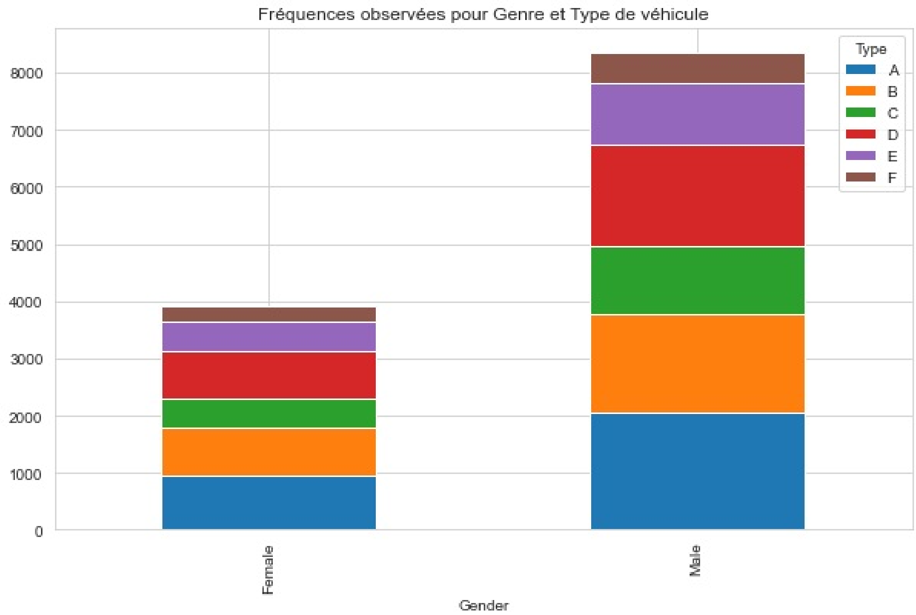
```
sns.boxplot(x='Gender', y='Bonus', data=base_Finale)
plt.title("Distribution de Bonus/Malus par genre du conducteur")
plt.savefig('distribution_bonus_Gender.png')
plt.show()
```



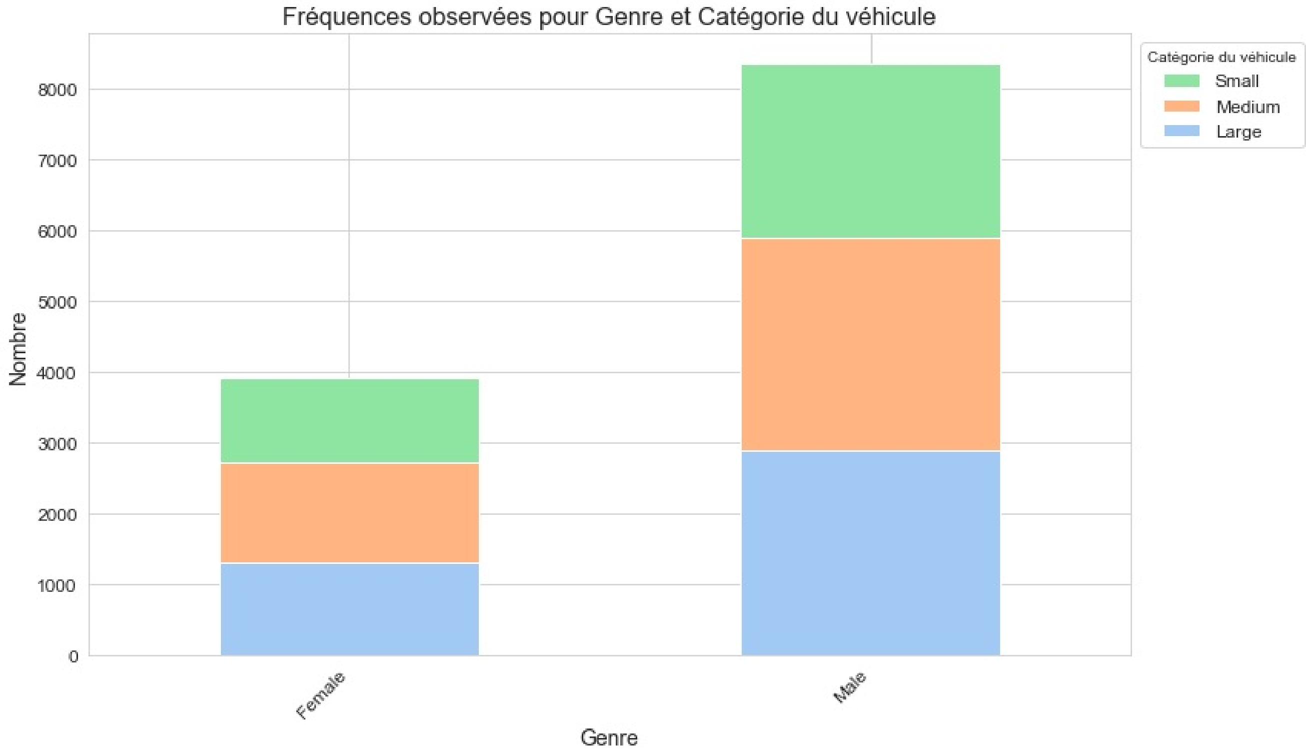
In [88]:
```
from scipy.stats import chi2_contingency

# Test du Chi-deux entre Gender et Type
table_contingence_genre_type = pd.crosstab(base_Finale['Gender'], base_Finale['Type'])
chi2, p, dof, expected = chi2_contingency(table_contingence_genre_type)
print("Chi2 entre Genre et Type :", chi2)
print("Valeur p entre Genre et Type :", p)

# Test du Chi-deux entre Gender et Category
table_contingence_genre_categorie = pd.crosstab(base_Finale['Gender'], base_Finale['Category'])
chi2, p, dof, expected = chi2_contingency(table_contingence_genre_categorie)
print("Chi2 entre Genre et Catégorie :", chi2)
print("Valeur p entre Genre et Catégorie :", p)

# Test du Chi-deux entre Type et Category
table_contingence_type_categorie = pd.crosstab(base_Finale['Type'], base_Finale['Category'])
chi2, p, dof, expected = chi2_contingency(table_contingence_type_categorie)
print("Chi2 entre Type et Catégorie :", chi2)
print("Valeur p entre Type et Catégorie :", p)
```

```
Chi2 entre Genre et Type : 1.3085901108608664
Valeur p entre Genre et Type : 0.9340452853832044
Chi2 entre Genre et Catégorie : 2.0591603983578555
Valeur p entre Genre et Catégorie : 0.35715686384688083
Chi2 entre Type et Catégorie : 9.446110844528883
Valeur p entre Type et Catégorie : 0.49035263694057285
```

In [89]:
```
#Genre et Type
table_contingence_genre_type.plot.bar(stacked=True, figsize=(10, 6))
plt.title("Fréquences observées pour Genre et Type de véhicule")
plt.savefig('frequences_genre_type.png')
plt.show()
```

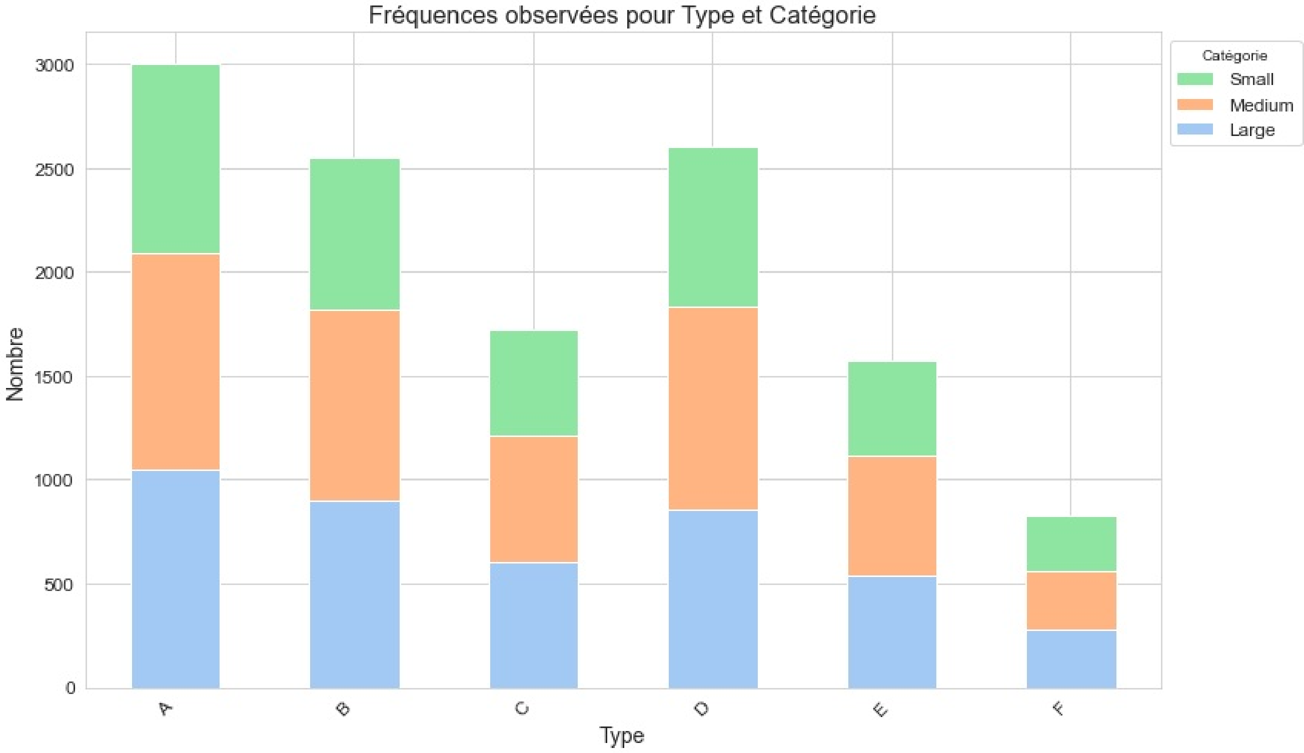


In [90]:
```
# Configuration de style
sns.set_style("whitegrid")

fig, ax = plt.subplots(figsize=(12, 7))
table_contingence_genre_categorie.plot.bar(stacked=True, ax=ax, color=sns.color_palette("pastel"))

plt.title("Fréquences observées pour Genre et Catégorie du véhicule", fontsize=16)
plt.xlabel('Genre', fontsize=14)
plt.ylabel('Nombre', fontsize=14)

plt.xticks(rotation=45, ha='right', fontsize=12)
plt.yticks(fontsize=12)

handles, labels = ax.get_legend_handles_labels()
ax.legend(handles[::-1], labels[::-1], title='Catégorie du véhicule', fontsize=12, loc='upper left', bbox_to_an

plt.tight_layout()
plt.savefig('frequences_genre_category_améliorés.png', dpi=300)
plt.show()
```



In [91]:
```
sns.set_style("whitegrid")

fig, ax = plt.subplots(figsize=(12, 7))
table_contingence_type_categorie.plot.bar(stacked=True, ax=ax, color=sns.color_palette("pastel"))

plt.title("Fréquences observées pour Type et Catégorie", fontsize=16)
plt.xlabel('Type', fontsize=14)
plt.ylabel('Nombre', fontsize=14)

plt.xticks(rotation=45, ha='right', fontsize=12)
```

```
plt.yticks(fontsize=12)

handles, labels = ax.get_legend_handles_labels()
ax.legend(handles[::-1], labels[::-1], title='Catégorie', fontsize=12, loc='upper left', bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.savefig('frequences_type_category_améliorés.png', dpi=300)
plt.show()
```



Section 3 : Analyse de Régression Linéaire

In [92]:
```
# Encodage à chaud pour les variables catégorielles
variables_categorielles = ['Gender', 'Type', 'Category', 'Occupation', 'Group2']
base_Finale_encoded = pd.get_dummies(base_Finale, columns=variables_categorielles)
```

In [93]:
```
import statsmodels.api as sm

# Définir la variable dépendante et les variables indépendantes
y = base_Finale_encoded['nb_sin']
X = base_Finale_encoded.drop(columns=['nb_sin', 'chg_sin', 'PolNum'])
# Ajouter une constante (intercept) à notre modèle
X = sm.add_constant(X)

# Créer le modèle
model = sm.OLS(y, X).fit()

# Afficher le résumé du modèle
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 nb_sin   R-squared:                       0.109
Model:                            OLS   Adj. R-squared:                  0.107
Method:                 Least Squares   F-statistic:                     49.98
Date:                Wed, 06 Sep 2023   Prob (F-statistic):          2.91e-279
Time:                        15:31:18   Log-Likelihood:                -9306.7
No. Observations:               12277   AIC:                         1.868e+04
Df Residuals:                   12246   BIC:                         1.891e+04
Df Model:                          30
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                      -18.3588      8.170     -2.247      0.025     -34.372      -2.345
CalYear                      0.0214      0.009      2.290      0.022       0.003       0.040
Age                         -0.0053      0.000    -13.001      0.000      -0.006      -0.005
Group1                       0.0148      0.001     13.979      0.000       0.013       0.017
Bonus                        0.0019   8.13e-05     22.998      0.000       0.002       0.002
Poldur                      -0.0066      0.001     -6.327      0.000      -0.009      -0.005
Value                    -9.242e-07   8.67e-07     -1.066      0.287   -2.62e-06    7.76e-07
Adind                       -0.0277      0.010     -2.846      0.004      -0.047      -0.009
Density                      0.0014      0.000      8.965      0.000       0.001       0.002
Expdays                      0.0008    8.3e-05      9.197      0.000       0.001       0.001
Gender_Female               -9.2160      4.085     -2.256      0.024     -17.223      -1.209
Gender_Male                 -9.1428      4.085     -2.238      0.025     -17.150      -1.136
Type_A                      -3.0963      1.362     -2.274      0.023      -5.765      -0.427
Type_B                      -3.0842      1.362     -2.265      0.024      -5.753      -0.415
Type_C                      -3.0946      1.362     -2.273      0.023      -5.764      -0.426
Type_D                      -3.0317      1.362     -2.227      0.026      -5.701      -0.363
Type_E                      -3.0064      1.362     -2.208      0.027      -5.675      -0.337
Type_F                      -3.0455      1.362     -2.237      0.025      -5.715      -0.377
Category_Large              -6.1082      2.723     -2.243      0.025     -11.446      -0.770
Category_Medium             -6.1214      2.723     -2.248      0.025     -11.459      -0.783
Category_Small              -6.1292      2.723     -2.251      0.024     -11.467      -0.791
Occupation_Employed         -3.6993      1.634     -2.264      0.024      -6.902      -0.497
Occupation_Housewife        -3.6431      1.634     -2.230      0.026      -6.846      -0.440
Occupation_Retired          -3.6606      1.634     -2.240      0.025      -6.864      -0.458
Occupation_Self-employed    -3.7045      1.634     -2.267      0.023      -6.907      -0.502
Occupation_Unemployed       -3.6513      1.634     -2.235      0.025      -6.854      -0.448
Group2_L                    -1.8503      0.817     -2.264      0.024      -3.452      -0.248
Group2_M                    -1.8423      0.817     -2.255      0.024      -3.443      -0.241
Group2_N                    -1.7543      0.817     -2.148      0.032      -3.355      -0.153
Group2_O                    -1.7715      0.817     -2.167      0.030      -3.374      -0.169
Group2_P                    -1.7967      0.818     -2.197      0.028      -3.400      -0.194
Group2_Q                    -1.9005      0.817     -2.326      0.020      -3.502      -0.299
Group2_R                    -1.9161      0.817     -2.345      0.019      -3.517      -0.315
Group2_S                    -1.8116      0.818     -2.215      0.027      -3.415      -0.209
Group2_T                    -1.8227      0.817     -2.230      0.026      -3.425      -0.220
Group2_U                    -1.8929      0.817     -2.316      0.021      -3.495      -0.291
==============================================================================
Omnibus:                     8806.047   Durbin-Watson:                   1.943
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           180950.704
Skew:                           3.246   Prob(JB):                         0.00
Kurtosis:                      20.652   Cond. No.                     1.18e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.63e-20. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [94]:

```
import statsmodels.api as sm

# Définir la variable dépendante et les variables indépendantes
y = base_Finale_encoded['chg_sin']
X = base_Finale_encoded.drop(columns=['nb_sin', 'chg_sin', 'PolNum'])

# Ajouter une constante (intercept) à notre modèle
X = sm.add_constant(X)

# Créer le modèle
model = sm.OLS(y, X).fit()

# Afficher le résumé du modèle
print(model.summary())
```

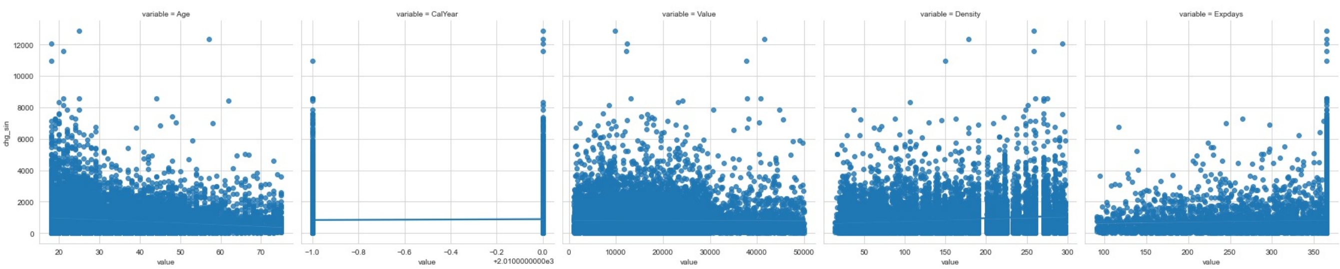
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                chg_sin   R-squared:                       0.080
Model:                            OLS   Adj. R-squared:                  0.078
Method:                 Least Squares   F-statistic:                     35.64
Date:                Wed, 06 Sep 2023   Prob (F-statistic):          1.55e-196
Time:                        15:31:19   Log-Likelihood:            -1.0151e+05
No. Observations:               12277   AIC:                         2.031e+05
Df Residuals:                   12246   BIC:                         2.033e+05
Df Model:                          30
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                    -5.092e+04   1.49e+04     -3.413      0.001   -8.02e+04   -2.17e+04
CalYear                     58.6598     17.076      3.435      0.001      25.189      92.131
Age                        -11.6510      0.747    -15.593      0.000     -13.116     -10.186
Group1                       9.9569      1.935      5.146      0.000       6.165      13.749
Bonus                       -0.0422      0.149     -0.284      0.777      -0.333       0.249
Poldur                      -4.9728      1.898     -2.621      0.009      -8.693      -1.253
Value                        0.0014      0.002      0.891      0.373      -0.002       0.005
Adind                     -133.1886     17.769     -7.495      0.000    -168.019     -98.358
Density                      2.3251      0.285      8.147      0.000       1.766       2.885
Expdays                      0.7093      0.152      4.679      0.000       0.412       1.006
Gender_Female            -2.551e+04   7459.456     -3.420      0.001   -4.01e+04   -1.09e+04
Gender_Male              -2.541e+04   7459.503     -3.406      0.001      -4e+04   -1.08e+04
Type_A                   -8374.0393   2486.629     -3.368      0.001   -1.32e+04   -3499.855
Type_B                   -8398.7350   2486.560     -3.378      0.001   -1.33e+04   -3524.684
Type_C                   -8485.7771   2486.474     -3.413      0.001   -1.34e+04   -3611.895
Type_D                   -8540.1674   2486.628     -3.434      0.001   -1.34e+04   -3665.984
Type_E                   -8543.6278   2486.607     -3.436      0.001   -1.34e+04   -3669.485
Type_F                   -8577.3006   2486.579     -3.449      0.001   -1.35e+04   -3703.214
Category_Large             -1.7e+04   4973.054     -3.418      0.001   -2.67e+04   -7248.494
Category_Medium          -1.699e+04   4973.046     -3.417      0.001   -2.67e+04   -7244.740
Category_Small           -1.693e+04   4972.959     -3.405      0.001   -2.67e+04   -7182.705
Occupation_Employed      -1.035e+04   2983.780     -3.468      0.001   -1.62e+04   -4498.371
Occupation_Housewife     -1.042e+04   2983.894     -3.492      0.000   -1.63e+04   -4571.875
Occupation_Retired       -9766.8594   2983.996     -3.273      0.001   -1.56e+04   -3917.756
Occupation_Self-employed -1.028e+04   2983.798     -3.444      0.001   -1.61e+04   -4426.989
Occupation_Unemployed    -1.011e+04   2983.968     -3.388      0.001    -1.6e+04   -4260.207
Group2_L                 -5097.0173   1492.412     -3.415      0.001   -8022.380   -2171.655
Group2_M                 -5099.6916   1491.721     -3.419      0.001   -8023.699   -2175.684
Group2_N                 -4961.6085   1491.646     -3.326      0.001   -7885.469   -2037.748
Group2_O                 -5017.5464   1492.614     -3.362      0.001   -7943.305   -2091.787
Group2_P                 -5000.0960   1493.478     -3.348      0.001   -7927.549   -2072.643
Group2_Q                 -5213.0415   1491.905     -3.494      0.000   -8137.410   -2288.673
Group2_R                 -5250.0121   1491.895     -3.519      0.000   -8174.361   -2325.663
Group2_S                 -5041.3653   1493.441     -3.376      0.001   -7968.746   -2113.985
Group2_T                 -5108.5606   1492.771     -3.422      0.001   -8034.627   -2182.495
Group2_U                 -5130.7079   1492.341     -3.438      0.001   -8055.932   -2205.484
==============================================================================
Omnibus:                     7582.785   Durbin-Watson:                   1.980
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           119395.994
Skew:                           2.705   Prob(JB):                         0.00
Kurtosis:                      17.288   Cond. No.                     1.18e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.63e-20. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [95]:

```python
cols_to_plot = ['Age', 'CalYear', 'Value', 'Density', 'Expdays']
cols_to_plot.append('chg_sin')

melted_df = pd.melt(base_Finale[cols_to_plot], id_vars=['chg_sin'], var_name='variable', value_name='value')

sns.lmplot(x='value', y='chg_sin', col='variable', data=melted_df, sharex=False)
```

Out[95]:

```
<seaborn.axisgrid.FacetGrid at 0x20a2af2f880>
```



In [96]:

```python
import statsmodels.api as sm

# Définir la variable dépendante et les variables indépendantes
y = base_Finale_encoded['chg_sin']
X = base_Finale_encoded.drop(columns=['nb_sin', 'chg_sin', 'PolNum', 'Bonus', 'Value'])
```

```python
# Ajouter une constante (intercept) à notre modèle
X = sm.add_constant(X)

# Créer le modèle
model = sm.OLS(y, X).fit()

# Afficher le résumé du modèle
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                chg_sin   R-squared:                       0.080
Model:                            OLS   Adj. R-squared:                  0.078
Method:                 Least Squares   F-statistic:                     38.15
Date:                Wed, 06 Sep 2023   Prob (F-statistic):          6.22e-198
Time:                        15:31:35   Log-Likelihood:            -1.0151e+05
No. Observations:               12277   AIC:                         2.031e+05
Df Residuals:                   12248   BIC:                         2.033e+05
Df Model:                          28
Covariance Type:            nonrobust
===========================================================================================
                              coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------------
const                   -5.091e+04   1.49e+04     -3.413      0.001   -8.01e+04   -2.17e+04
CalYear                    58.6557     17.072      3.436      0.001      25.192      92.120
Age                       -11.6373      0.747    -15.583      0.000     -13.101     -10.173
Group1                     10.1348      1.925      5.264      0.000       6.361      13.908
Poldur                     -4.9878      1.897     -2.629      0.009      -8.707      -1.269
Adind                    -133.3688     17.762     -7.509      0.000    -168.185     -98.552
Density                     2.3237      0.285      8.143      0.000       1.764       2.883
Expdays                     0.7103      0.152      4.686      0.000       0.413       1.007
Gender_Female           -2.551e+04   7457.747     -3.420      0.001   -4.01e+04   -1.09e+04
Gender_Male              -2.54e+04   7457.802     -3.406      0.001       -4e+04   -1.08e+04
Type_A                  -8372.1270   2486.064     -3.368      0.001   -1.32e+04   -3499.050
Type_B                  -8396.8091   2485.984     -3.378      0.001   -1.33e+04   -3523.889
Type_C                  -8483.7355   2485.903     -3.413      0.001   -1.34e+04   -3610.973
Type_D                  -8537.8859   2486.060     -3.434      0.001   -1.34e+04   -3664.816
Type_E                  -8541.6009   2486.040     -3.436      0.001   -1.34e+04   -3668.570
Type_F                  -8575.1963   2486.016     -3.449      0.001   -1.34e+04   -3702.213
Category_Large          -1.698e+04   4971.746     -3.414      0.001   -2.67e+04   -7229.745
Category_Medium         -1.699e+04   4971.932     -3.417      0.001   -2.67e+04   -7245.572
Category_Small          -1.694e+04   4971.904     -3.407      0.001   -2.67e+04   -7195.143
Occupation_Employed     -1.034e+04   2983.086     -3.468      0.001   -1.62e+04   -4497.474
Occupation_Housewife    -1.042e+04   2983.208     -3.492      0.000   -1.63e+04   -4570.648
Occupation_Retired      -9764.0735   2983.343     -3.273      0.001   -1.56e+04   -3916.251
Occupation_Self-employed -1.027e+04   2983.109     -3.444      0.001   -1.61e+04   -4426.077
Occupation_Unemployed   -1.011e+04   2983.280     -3.388      0.001    -1.6e+04   -4259.143
Group2_L                -5096.0831   1492.068     -3.415      0.001   -8020.771   -2171.395
Group2_M                -5098.0216   1491.388     -3.418      0.001   -8021.378   -2174.665
Group2_N                -4960.1284   1491.315     -3.326      0.001   -7883.341   -2036.916
Group2_O                -5016.5022   1492.275     -3.362      0.001   -7941.596   -2091.409
Group2_P                -4998.2471   1493.136     -3.347      0.001   -7925.030   -2071.464
Group2_Q                -5211.3459   1491.558     -3.494      0.000   -8135.036   -2287.656
Group2_R                -5248.2602   1491.559     -3.519      0.000   -8171.951   -2324.570
Group2_S                -5040.9544   1493.101     -3.376      0.001   -7967.667   -2114.242
Group2_T                -5108.7905   1492.416     -3.423      0.001   -8034.161   -2183.420
Group2_U                -5129.0213   1491.995     -3.438      0.001   -8053.567   -2204.475
==============================================================================
Omnibus:                     7584.319   Durbin-Watson:                   1.980
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           119488.229
Skew:                           2.706   Prob(JB):                         0.00
Kurtosis:                      17.293   Cond. No.                     1.00e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 5.12e-22. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Section 4 : Diagnostics du Modèle MCO

- Normalité des Résidus

In [97]:
```python
from scipy.stats import shapiro

residuals = model.resid
statistic, p_value = shapiro(residuals)

print('Statistic:', statistic)
print('p-value:', p_value)
```

```
Statistic: 0.7989834547042847
p-value: 0.0
```

In [98]:
```python
from statsmodels.stats.outliers_influence import OLSInfluence

influence = OLSInfluence(model)
```

```python
leverage = influence.hat_matrix_diag
high_leverage_points = np.where(leverage > 0.05)
print(high_leverage_points)
```

```
(array([], dtype=int64),)
```

In [99]:
```python
large_residuals = np.where(influence.resid_studentized_external > 2)
print(large_residuals)
```

```
(array([    8,    28,    29,    90,   155,   163,   223,   231,   293,
          294,   300,   398,   411,   415,   478,   483,   507,   508,
          515,   546,   614,   634,   637,   655,   683,   698,   707,
          740,   742,   786,   830,   855,   883,   895,   907,   908,
          943,   966,  1014,  1044,  1053,  1055,  1069,  1098,  1102,
         1112,  1130,  1138,  1156,  1190,  1218,  1260,  1266,  1277,
         1291,  1332,  1343,  1377,  1405,  1409,  1432,  1439,  1481,
         1487,  1497,  1540,  1609,  1618,  1658,  1680,  1682,  1719,
         1724,  1737,  1743,  1776,  1830,  1851,  1862,  1864,  1869,
         1872,  1918,  1943,  1950,  1953,  2019,  2043,  2050,  2062,
         2085,  2115,  2124,  2137,  2143,  2201,  2214,  2230,  2250,
         2251,  2261,  2291,  2292,  2352,  2357,  2368,  2382,  2387,
         2388,  2437,  2440,  2455,  2459,  2465,  2474,  2498,  2503,
         2534,  2535,  2552,  2579,  2622,  2657,  2707,  2737,  2745,
         2761,  2801,  2820,  2824,  2826,  2827,  2835,  2837,  2923,
         2925,  3010,  3062,  3104,  3118,  3176,  3248,  3254,  3280,
         3318,  3327,  3378,  3379,  3395,  3397,  3402,  3410,  3460,
         3467,  3469,  3551,  3580,  3587,  3606,  3609,  3625,  3633,
         3669,  3678,  3701,  3702,  3757,  3776,  3843,  3892,  3922,
         3994,  4064,  4082,  4111,  4143,  4151,  4152,  4156,  4220,
         4236,  4294,  4326,  4335,  4344,  4355,  4379,  4437,  4469,
         4500,  4504,  4518,  4521,  4530,  4560,  4565,  4589,  4616,
         4678,  4708,  4712,  4737,  4739,  4753,  4758,  4779,  4829,
         4905,  4976,  4990,  5004,  5009,  5076,  5087,  5116,  5156,
         5244,  5266,  5285,  5294,  5296,  5322,  5437,  5489,  5492,
         5554,  5583,  5584,  5606,  5665,  5667,  5677,  5704,  5750,
         5794,  5801,  5803,  5857,  5878,  5959,  5965,  5990,  5992,
         6030,  6060,  6063,  6092,  6118,  6119,  6120,  6133,  6157,
         6160,  6174,  6181,  6194,  6203,  6204,  6232,  6304,  6327,
         6385,  6421,  6425,  6453,  6454,  6514,  6516,  6554,  6565,
         6584,  6592,  6625,  6637,  6668,  6678,  6680,  6694,  6709,
         6724,  6739,  6741,  6803,  6809,  6839,  6840,  6856,  6887,
         6888,  6927,  6929,  6953,  7009,  7040,  7135,  7182,  7215,
         7223,  7224,  7225,  7226,  7235,  7258,  7263,  7316,  7334,
         7370,  7378,  7389,  7396,  7410,  7429,  7434,  7443,  7486,
         7501,  7548,  7580,  7857,  7863,  7872,  7877,  7901,  7963,
         7966,  8023,  8104,  8129,  8131,  8165,  8187,  8224,  8256,
         8287,  8289,  8293,  8305,  8320,  8322,  8323,  8362,  8379,
         8395,  8397,  8401,  8440,  8444,  8453,  8470,  8484,  8528,
         8533,  8586,  8596,  8653,  8654,  8664,  8675,  8727,  8736,
         8871,  8872,  8884,  8902,  8932,  8941,  8952,  8969,  8971,
         9023,  9048,  9057,  9080,  9088,  9095,  9100,  9121,  9122,
         9153,  9197,  9268,  9292,  9303,  9325,  9341,  9342,  9365,
         9383,  9418,  9436,  9438,  9459,  9476,  9486,  9500,  9539,
         9552,  9559,  9695,  9712,  9734,  9765,  9813,  9849,  9859,
         9869,  9897,  9907,  9911,  9915,  9937,  9944, 10011, 10024,
        10082, 10090, 10110, 10157, 10187, 10201, 10242, 10262, 10271,
        10272, 10274, 10293, 10301, 10318, 10394, 10434, 10459, 10526,
        10586, 10612, 10628, 10642, 10664, 10671, 10714, 10743, 10750,
        10772, 10848, 10856, 10863, 10887, 10896, 10916, 10953, 10974,
        10999, 11101, 11119, 11123, 11127, 11145, 11158, 11234, 11257,
        11294, 11316, 11359, 11367, 11374, 11397, 11401, 11429, 11439,
        11456, 11500, 11515, 11527, 11540, 11573, 11589, 11609, 11612,
        11670, 11676, 11689, 11713, 11721, 11732, 11753, 11760, 11781,
        11792, 11795, 11822, 11837, 11842, 11852, 11891, 11892, 11902,
        11944, 11966, 11971, 11985, 12051, 12074, 12085, 12154, 12217,
        12221, 12231, 12253, 12267, 12268, 12273, 12276], dtype=int64),)
```

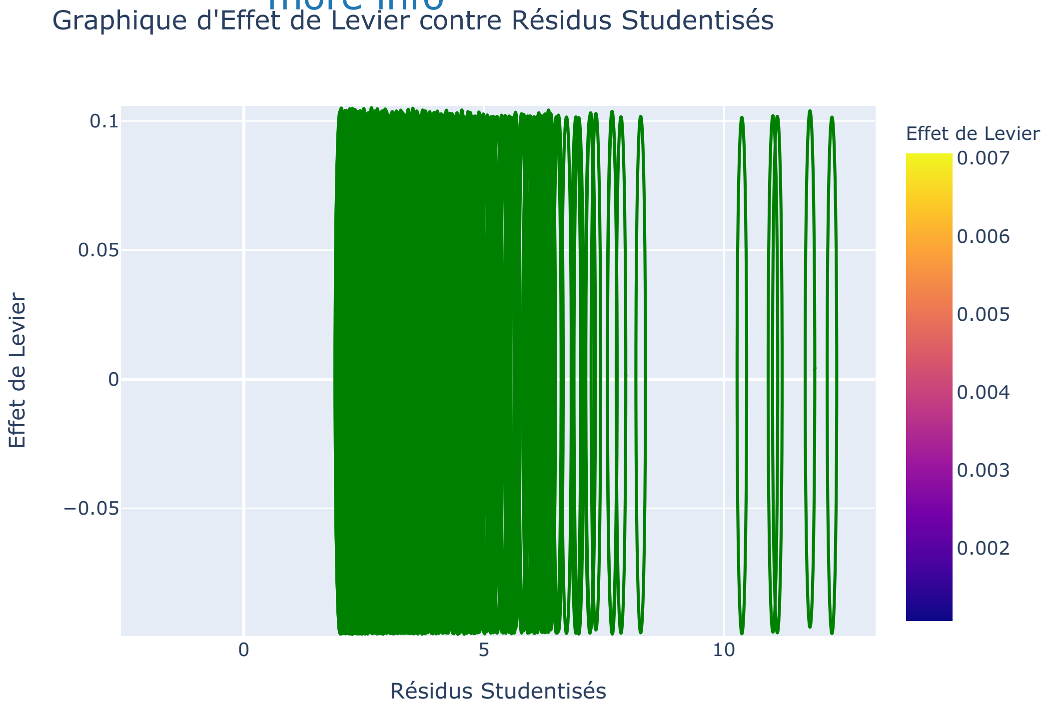
In [100…
```python
import statsmodels.api as sm

# Supposons que X et y soient vos données
X = sm.add_constant(X)  # ajouter une constante si nécessaire
model = sm.OLS(y, X).fit()

from statsmodels.graphics.regressionplots import plot_leverage_resid2

# Ensuite, le reste de votre code devrait fonctionner
sns.set_style("whitegrid")

fig, ax = plt.subplots(figsize=(15, 10))
plot_leverage_resid2(model, ax=ax, color='blue')

# Mettez en évidence les points avec de grands résidus
for i in large_residuals[0]:
    ax.scatter(influence.resid_studentized_external[i], influence.hat_matrix_diag[i], s=200, linewidth=2, facec

# Titre, étiquettes et mise en forme
plt.xlabel('Résidus Studentisés', fontsize=16)
plt.ylabel('Effet de Levier', fontsize=16)
plt.title('Graphique d\'Effet de Levier contre Résidus Studentisés', fontsize=18)
```

```
# Sauvegarde en haute qualité
plt.tight_layout()
plt.savefig('leverage_vs_residuals_améliorés.png', dpi=300)

plt.show()
```

In [101…

```
import plotly.express as px

df = pd.DataFrame({
    'Résidus Studentisés': influence.resid_studentized_external,
    'Effet de Levier': influence.hat_matrix_diag
})

fig = px.scatter(df, x='Résidus Studentisés', y='Effet de Levier',
                 color='Effet de Levier',
                 title="Graphique d'Effet de Levier contre Résidus Studentisés",
                 labels={'Résidus Studentisés': 'Résidus Studentisés', 'Effet de Levier': 'Effet de Levier'})

for i in large_residuals[0]:
    fig.add_shape(type='circle',
                  xref='x', yref='y',
                  x0=influence.resid_studentized_external[i] - 0.1, y0=influence.hat_matrix_diag[i] - 0.1,
                  x1=influence.resid_studentized_external[i] + 0.1, y1=influence.hat_matrix_diag[i] + 0.1,
                  line=dict(color='Green'))

fig.write_html('graphique.html')
fig.show()
```

more info



Homoscédasticité

In [102…

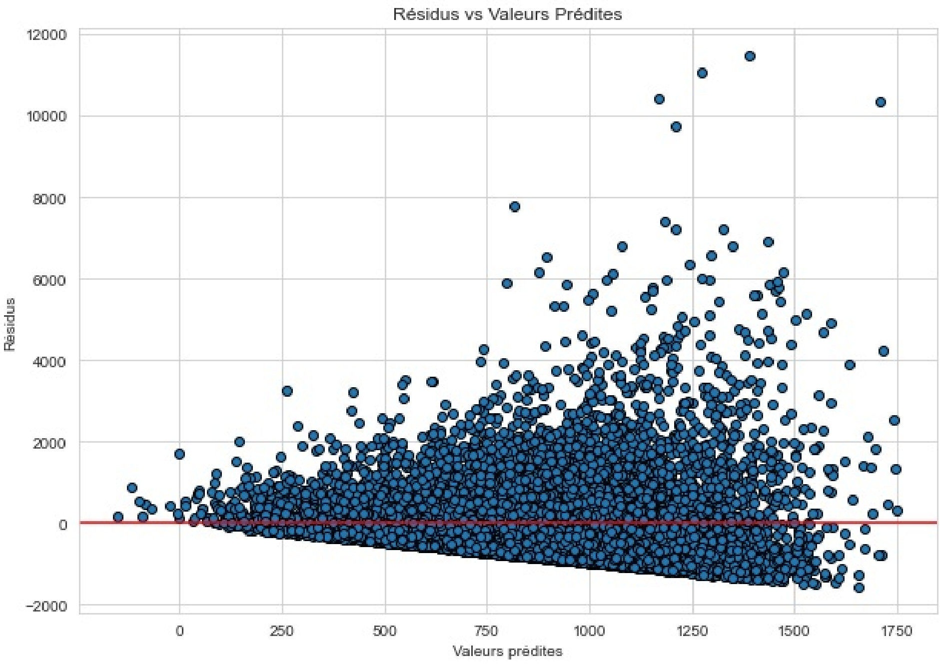
```python
from statsmodels.compat import lzip
import statsmodels.stats.api as sms

name = ['Lagrange multiplier statistic', 'p-value',
        'f-value', 'f p-value']
test = sms.het_breuschpagan(residuals, model.model.exog)
lzip(name, test)
```

Out[102]:

```
[('Lagrange multiplier statistic', 360.8236415595496),
 ('p-value', 8.78720670482319e-57),
 ('f-value', 13.245404005224515),
 ('f p-value', 1.6717467427272165e-60)]
```

In [105…

```python
# Configuration du style

sns.set_style("whitegrid")

ppredicted_values = model.fittedvalues
residuals = model.resid

# Tracer les résidus par rapport aux valeurs prédites
plt.figure(figsize=(10, 7))
plt.scatter(predicted_values, residuals, edgecolors='k')
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel('Valeurs prédites')
plt.ylabel('Résidus')
plt.title('Résidus vs Valeurs Prédites')
plt.savefig('Résidus.png')
plt.show()
```



Multicolinéarité

In [106…

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif_data = pd.DataFrame()
vif_data["feature"] = model.model.exog_names
vif_data["VIF"] = [variance_inflation_factor(model.model.exog, i)
                   for i in range(len(model.model.exog_names))]

print(vif_data)
```

```
                  feature       VIF
0                   const  0.000000
1                 CalYear  1.001936
2                     Age  1.358096
3                  Group1  1.059287
4                  Poldur  1.003789
5                   Adind  1.073191
6                 Density  7.938060
7                 Expdays  1.001994
8           Gender_Female       inf
9             Gender_Male       inf
10                 Type_A       inf
11                 Type_B       inf
12                 Type_C       inf
13                 Type_D       inf
14                 Type_E       inf
15                 Type_F       inf
16         Category_Large       inf
17        Category_Medium       inf
18         Category_Small       inf
19    Occupation_Employed       inf
20   Occupation_Housewife       inf
21     Occupation_Retired       inf
22  Occupation_Self-employed    inf
23  Occupation_Unemployed       inf
24               Group2_L       inf
25               Group2_M       inf
26               Group2_N       inf
27               Group2_O       inf
28               Group2_P       inf
29               Group2_Q       inf
30               Group2_R       inf
31               Group2_S       inf
32               Group2_T       inf
33               Group2_U       inf
```

Influence des Observations Individuelles

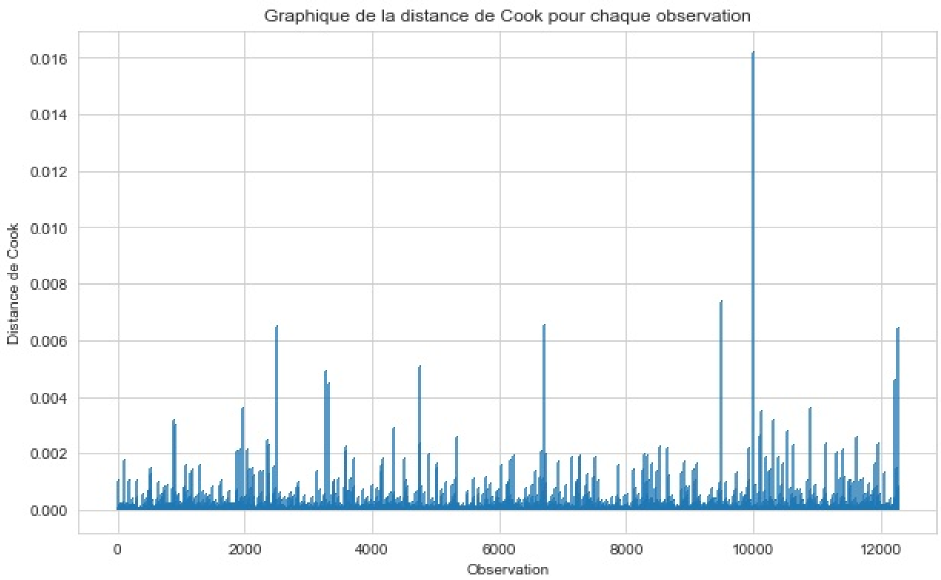
In [107…

```python
from statsmodels.stats.outliers_influence import OLSInfluence

influence = OLSInfluence(model)
influence_summary = influence.summary_frame()
print(influence_summary[['cooks_d']])
```

```
          cooks_d
0      1.668221e-07
1      1.065514e-04
2      7.610876e-07
3      4.896620e-07
4      4.657627e-05
...             ...
12272  7.109444e-05
12273  4.136303e-04
12274  7.653962e-07
12275  1.673475e-04
12276  6.452234e-03

[12277 rows x 1 columns]
```

In [108…

```python
from statsmodels.stats.outliers_influence import OLSInfluence
influence = OLSInfluence(model)
influence_summary = influence.summary_frame()
cooks_d = influence_summary['cooks_d']

plt.figure(figsize=(10, 6))
plt.stem(cooks_d, linefmt='-', markerfmt=',', basefmt=" ")
plt.title('Graphique de la distance de Cook pour chaque observation')
plt.xlabel('Observation')
plt.ylabel('Distance de Cook')
plt.savefig('cooks_distance.png')
plt.show()
```



Lignes d'Ajustement

Validation Croisée

In [109…

```python
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

# Sélection des variables indépendantes et dépendantes
X = base_Finale_encoded.drop(columns=['nb_sin', 'chg_sin', 'PolNum', 'Bonus', 'Value'])
y = base_Finale_encoded['chg_sin']

# Création du modèle de régression linéaire
model = LinearRegression()

# Effectuer la validation croisée 10-fold
scores = cross_val_score(model, X, y, cv=10)

# Afficher les scores pour chaque fold
print('Scores pour chaque fold:', scores)

# Afficher la moyenne des scores
print('Score moyen:', scores.mean())
```

```
Scores pour chaque fold: [0.07871992 0.0333325  0.0597756  0.05878873 0.09398965 0.09108667
 0.09304505 0.10102653 0.06259332 0.07430033]
Score moyen: 0.07466583027059928
```

PCA

In [110…

```python
from sklearn.decomposition import PCA

# Sélectionnez vos variables indépendantes, y compris 'Bonus' et 'Value'
X = base_Finale[['CalYear', 'Age', 'Group1', 'Poldur', 'Adind', 'Density', 'Expdays', 'Bonus', 'Value']]

# Création de l'objet PCA
```

```python
pca = PCA()

# Ajuster et transformer les données
X_pca = pca.fit_transform(X)

# Afficher la variance expliquée par chaque composante
print("Variance expliquée par chaque composante:", pca.explained_variance_ratio_)
```

```
Variance expliquée par chaque composante: [9.99880056e-01 6.16356204e-05 2.90075779e-05 2.74170962e-05
 1.53495022e-06 1.76381541e-07 1.68287454e-07 2.17599404e-09
 2.00150125e-09]
```

In [111…

```python
from sklearn.preprocessing import StandardScaler

# Standardiser les données
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Appliquer l'ACP sur les données standardisées
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# Afficher la variance expliquée
print("Variance expliquée par chaque composante:", pca.explained_variance_ratio_)
```

```
Variance expliquée par chaque composante: [0.15001555 0.12840262 0.11521321 0.11392985 0.11084003 0.10958058
 0.10581939 0.0865174  0.07968136]
```

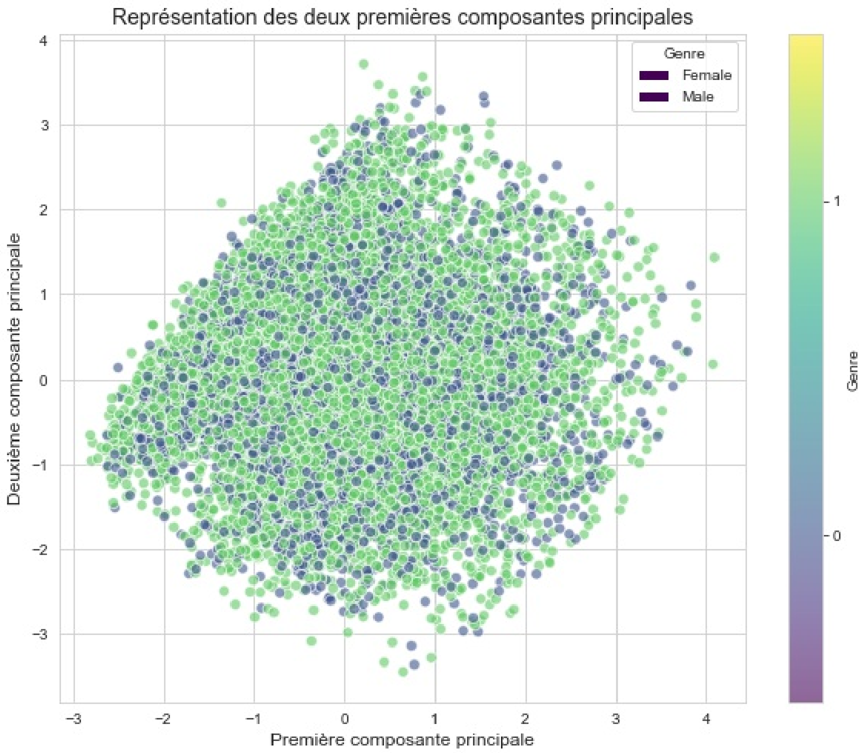
In [112…

```python
# Afficher les coefficients de la première composante principale
print("Composante principale 1:", pca.components_[0])
```

```
Composante principale 1: [-0.00135699  0.58405785  0.51006604  0.0497943   0.47932903  0.01358403
  0.01135581 -0.11006307  0.3924845 ]
```

In [113…

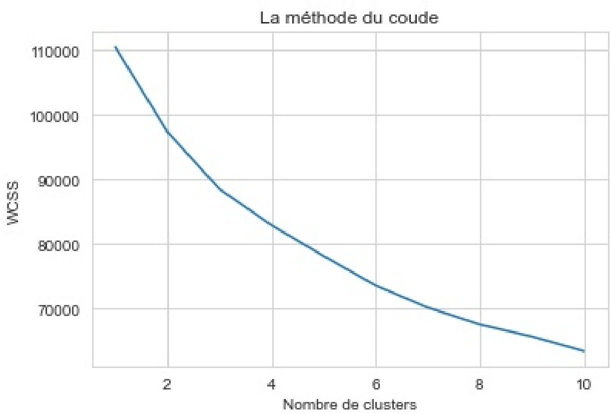
```python
import matplotlib.cm as cm
import matplotlib.patches as mpatches

base_Finale['Gender'] = base_Finale['Gender'].replace({'Female': 0, 'Male': 1})
colors = base_Finale['Gender']

plt.figure(figsize=(10, 8))

# Tracer le nuage de points
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=colors, cmap='viridis', s=50, alpha=0.6, edgecolors='w')

plt.xlabel('Première composante principale', fontsize=12)
plt.ylabel('Deuxième composante principale', fontsize=12)
plt.title('Représentation des deux premières composantes principales', fontsize=14)

cmap = cm.get_cmap('viridis')

legend_elements = [mpatches.Patch(facecolor=cmap(0), edgecolor='w', label='Female'),
                   mpatches.Patch(facecolor=cmap(1), edgecolor='w', label='Male')]

plt.legend(handles=legend_elements, loc='upper right', title='Genre')

# Ajouter une barre de couleurs
plt.colorbar(scatter, label='Genre', ticks=[0,1])
plt.clim(-0.5, 1.5)  # Ajuste les limites de la barre de couleurs

plt.savefig('pca_plot.png')
plt.show()
```



Segmentation des Polices d'Assurance par Clustering (K-means)

- Déterminer le Nombre Idéal de Clusters

In [114…
```
from sklearn.cluster import KMeans

wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X_pca)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('La méthode du coude')
plt.xlabel('Nombre de clusters')
plt.ylabel('WCSS')
plt.show()
```



In [115…
```
from scipy.spatial.distance import cdist

# Calculez la somme des carrés pour chaque nombre de clusters
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X_pca)
    wcss.append(kmeans.inertia_)

# Trouvez la ligne reliant les points extrêmes
p1 = [1, wcss[0]]
p2 = [10, wcss[-1]]
line_eq = lambda x: ((p2[1] - p1[1]) / (p2[0] - p1[0])) * (x - p1[0]) + p1[1]

# Calculez la distance de chaque point à cette ligne
distances = [np.abs((p2[1] - p1[1]) * x - (p2[0] - p1[0]) * y + p2[0] * p1[1] - p2[1] * p1[0]) /
             np.sqrt((p2[1] - p1[1])**2 + (p2[0] - p1[0])**2) for x, y in enumerate(wcss, 1)]

# Trouvez le point avec la distance maximale
```

```
optimal_clusters = np.argmax(distances) + 1

print("Nombre optimal de clusters:", optimal_clusters)
```

Nombre optimal de clusters: 4

- Ajuster le Modèle K-means

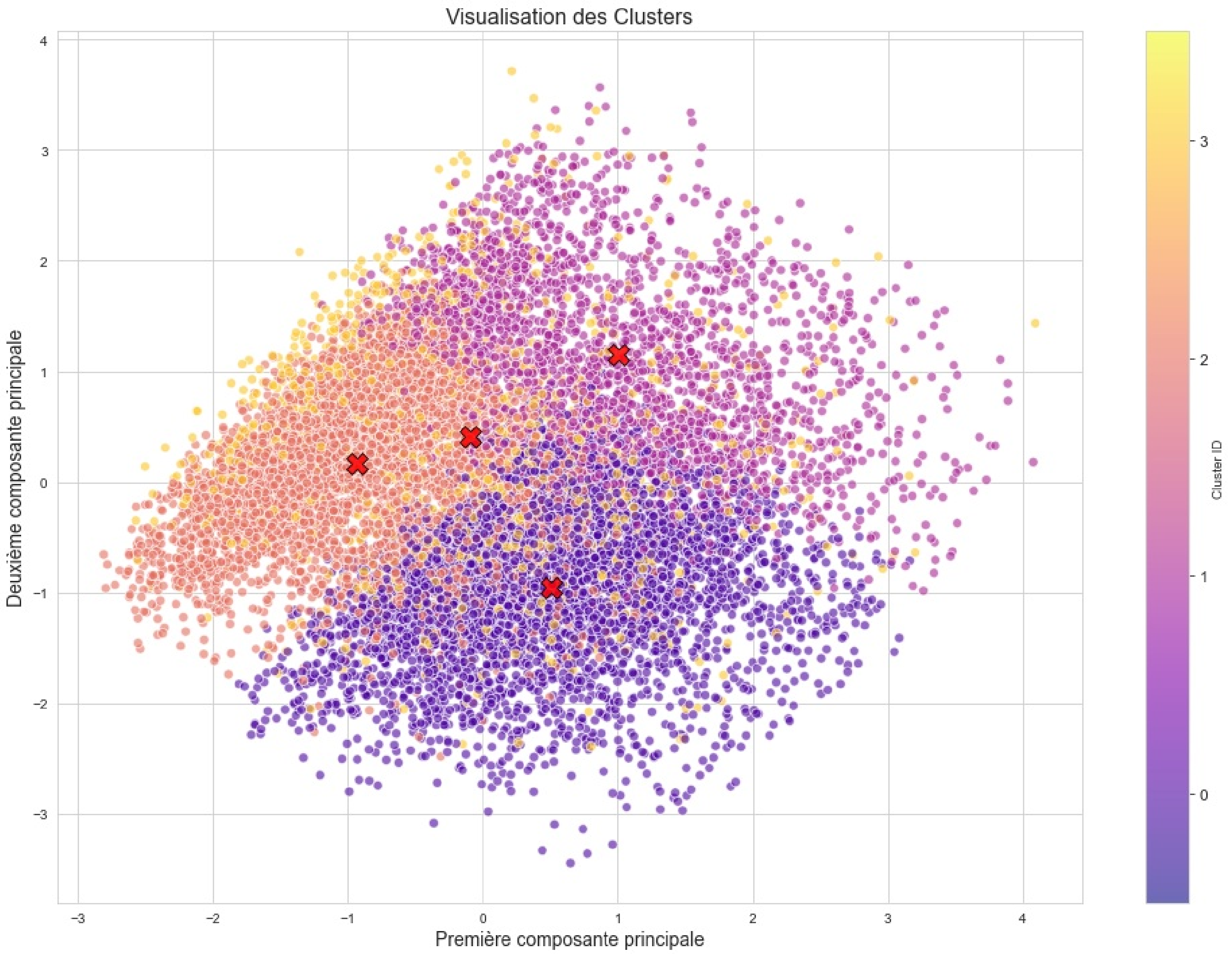
In [116…

```
kmeans = KMeans(n_clusters=4, init='k-means++', random_state=42)
kmeans.fit(X_pca)
clusters = kmeans.labels_
base_Finale['cluster'] = clusters
```

- Visualiser les Clusters

In [118…

```
# Configuration du style
sns.set_style("whitegrid")

plt.figure(figsize=(14, 10))

# Visualisation des clusters
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='plasma', s=50, alpha=0.6, edgecolors='w')

# Étiquettes, titre et barre de couleurs
plt.xlabel('Première composante principale', fontsize=14)
plt.ylabel('Deuxième composante principale', fontsize=14)
plt.title('Visualisation des Clusters', fontsize=16)
cbar = plt.colorbar(label='Cluster ID', ticks=range(4))
cbar.ax.tick_params(labelsize=12)  # Réglage de la taille des étiquettes de la barre de couleur
plt.clim(-0.5, 3.5)

# Centres des clusters
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=250, alpha=0.85, marker='X', edgecolors='black')

# Sauvegarder en haute qualité
plt.tight_layout()
plt.savefig('kmean.png', dpi=300)
plt.show()
```



- Analyser les Statistiques des Clusters

In [119…

```python
for i in range(4):
    print(f"Cluster {i}:")
    print(base_Finale[base_Finale['cluster'] == i].describe())
```

```
Cluster 0:
             PolNum      CalYear       Gender          Age       Group1  \
count  4.136000e+03  4136.000000  4136.000000  4136.000000  4136.000000   
mean   2.002040e+08  2009.532157     0.672872    38.531431    11.248308   
std    6.226245e+04     0.499025     0.469221    13.101962     4.422652   
min    2.001149e+08  2009.000000     0.000000    18.000000     1.000000   
25%    2.001408e+08  2009.000000     0.000000    28.000000     8.000000   
50%    2.002392e+08  2010.000000     1.000000    37.000000    11.000000   
75%    2.002620e+08  2010.000000     1.000000    47.000000    14.000000   
max    2.002858e+08  2010.000000     1.000000    75.000000    20.000000   

             Bonus       Poldur         Value   Adind      Density  \
count  4136.000000  4136.000000   4136.000000  4136.0  4136.000000   
mean     24.395551     5.046663  13655.313104     1.0   139.501228   
std      61.865183     4.508704   7217.599091     0.0    84.298786   
min     -50.000000     0.000000   1010.000000     1.0    14.377142   
25%     -30.000000     1.000000   7918.750000     1.0    63.828869   
50%      10.000000     4.000000  12977.500000     1.0   129.669008   
75%      70.000000     8.000000  19333.750000     1.0   212.424705   
max     150.000000    15.000000  38485.000000     1.0   297.385170   

           Expdays       nb_sin       chg_sin  cluster  
count  4136.000000  4136.000000   4136.000000   4136.0  
mean    359.230899     1.182302    755.946857      0.0  
std      19.112285     0.492886    827.960317      0.0  
min     247.000000     1.000000      0.180000      0.0  
25%     365.000000     1.000000    209.792500      0.0  
50%     365.000000     1.000000    508.500000      0.0  
75%     365.000000     1.000000   1021.457500      0.0  
max     365.000000     7.000000  12878.370000      0.0  
Cluster 1:
             PolNum      CalYear       Gender          Age       Group1  \
count  2.298000e+03  2298.000000  2298.000000  2298.000000  2298.000000   
mean   2.002011e+08  2009.499130     0.680592    38.159269    15.191036   
std    6.222303e+04     0.500108     0.466349    14.316190     3.418759   
min    2.001150e+08  2009.000000     0.000000    18.000000     1.000000   
25%    2.001413e+08  2009.000000     0.000000    26.000000    13.000000   
50%    2.001647e+08  2009.000000     1.000000    36.000000    16.000000   
75%    2.002622e+08  2010.000000     1.000000    48.000000    18.000000   
max    2.002857e+08  2010.000000     1.000000    75.000000    20.000000   

             Bonus       Poldur         Value        Adind      Density  \
count  2298.000000  2298.000000   2298.000000  2298.000000  2298.000000   
mean     16.392515     5.010879  32262.025674     0.352480   145.848479   
std      57.310168     4.421163   9228.202035     0.477847    83.624512   
min     -50.000000     0.000000   6215.000000     0.000000    14.377142   
25%     -30.000000     1.000000  25476.250000     0.000000    68.853880   
50%       0.000000     4.000000  29737.500000     0.000000   138.506671   
75%      60.000000     8.000000  39985.000000     1.000000   218.983230   
max     150.000000    20.000000  49990.000000     1.000000   297.385170   

           Expdays       nb_sin       chg_sin  cluster  
count  2298.000000  2298.000000   2298.000000   2298.0  
mean    357.424717     1.242385    916.480004      1.0  
std      23.606164     0.632866   1098.913029      0.0  
min     210.000000     1.000000      0.420000      1.0  
25%     365.000000     1.000000    231.390000      1.0  
50%     365.000000     1.000000    560.480000      1.0  
75%     365.000000     1.000000   1219.852500      1.0  
max     365.000000     7.000000  12324.740000      1.0  
Cluster 2:
             PolNum      CalYear       Gender          Age       Group1  \
count  4.603000e+03  4603.000000  4603.000000  4603.000000  4603.000000   
mean   2.002029e+08  2009.522485     0.681078    30.418640    10.445796   
std    6.219689e+04     0.499548     0.466110    11.552821     4.336188   
min    2.001149e+08  2009.000000     0.000000    18.000000     1.000000   
25%    2.001410e+08  2009.000000     0.000000    22.000000     7.000000   
50%    2.002376e+08  2010.000000     1.000000    27.000000    10.000000   
75%    2.002616e+08  2010.000000     1.000000    36.000000    14.000000   
max    2.002857e+08  2010.000000     1.000000    75.000000    20.000000   

             Bonus       Poldur         Value   Adind      Density  \
count  4603.000000  4603.000000   4603.000000  4603.0  4603.000000   
mean     22.446231     4.904627  12697.281121     0.0   137.495526   
std      53.273248     4.523278   6768.966399     0.0    83.909061   
min     -50.000000    -6.000000   1005.000000     0.0    14.377142   
25%     -10.000000     1.000000   7540.000000     0.0    61.944120   
50%       0.000000     4.000000  11470.000000     0.0   126.140188   
75%      50.000000     8.000000  17582.500000     0.0   210.189841   
max     150.000000    31.000000  39935.000000     0.0   297.385170   

           Expdays       nb_sin       chg_sin  cluster  
count  4603.000000  4603.000000   4603.000000   4603.0  
mean    359.812514     1.236802    965.656956      2.0  
std      17.725742     0.582947   1063.183435      0.0  
```

```
min     254.000000     1.000000      1.140000      2.0
25%     365.000000     1.000000    254.405000      2.0
50%     365.000000     1.000000    623.930000      2.0
75%     365.000000     1.000000   1302.915000      2.0
max     365.000000     7.000000  12055.250000      2.0
Cluster 3:
             PolNum      CalYear       Gender          Age       Group1  \
count  1.240000e+03  1240.000000  1240.000000  1240.000000  1240.000000
mean   2.002008e+08  2009.502419     0.704032    34.095968    11.938710
std    6.155717e+04     0.500196     0.456661    12.830854     4.439472
min    2.001149e+08  2009.000000     0.000000    18.000000     1.000000
25%    2.001413e+08  2009.000000     0.000000    24.000000     9.000000
50%    2.002361e+08  2010.000000     1.000000    31.000000    12.000000
75%    2.002612e+08  2010.000000     1.000000    42.000000    15.000000
max    2.002858e+08  2010.000000     1.000000    75.000000    20.000000

             Bonus       Poldur         Value        Adind      Density  \
count  1240.000000  1240.000000   1240.000000  1240.000000  1240.000000
mean     25.548387     5.071774  16723.822581     0.420161   143.710930
std      59.011408     4.527079   9944.096980     0.493784    84.877438
min     -50.000000     0.000000   1005.000000     0.000000    17.879958
25%     -20.000000     1.000000   8952.500000     0.000000    66.101880
50%      10.000000     4.000000  15310.000000     0.000000   138.506671
75%      70.000000     8.000000  22860.000000     1.000000   216.491601
max     150.000000    15.000000  49995.000000     1.000000   297.385170

           Expdays       nb_sin      chg_sin  cluster
count  1240.000000  1240.000000  1240.000000   1240.0
mean    190.174194     1.109677   783.191992      3.0
std      47.768563     0.365020   875.951665      0.0
min      91.000000     1.000000     0.360000      3.0
25%     153.000000     1.000000   209.577500      3.0
50%     195.000000     1.000000   510.965000      3.0
75%     229.000000     1.000000  1029.950000      3.0
max     278.000000     4.000000  7264.910000      3.0
```

In [120…

```python
cluster_summary = base_Finale.groupby('cluster').mean()
print(cluster_summary)
```

```
               PolNum      CalYear    Gender        Age     Group1      Bonus  \
cluster
0        2.002040e+08  2009.532157  0.672872  38.531431  11.248308  24.395551
1        2.002011e+08  2009.499130  0.680592  38.159269  15.191036  16.392515
2        2.002029e+08  2009.522485  0.681078  30.418640  10.445796  22.446231
3        2.002008e+08  2009.502419  0.704032  34.095968  11.938710  25.548387

           Poldur         Value     Adind     Density     Expdays    nb_sin  \
cluster
0        5.046663  13655.313104  1.000000  139.501228  359.230899  1.182302
1        5.010879  32262.025674  0.352480  145.848479  357.424717  1.242385
2        4.904627  12697.281121  0.000000  137.495526  359.812514  1.236802
3        5.071774  16723.822581  0.420161  143.710930  190.174194  1.109677

            chg_sin
cluster
0        755.946857
1        916.480004
2        965.656956
3        783.191992
```

In [121…

```python
cluster_summary = base_Finale.groupby('cluster').mean()
print(cluster_summary)
```

```
               PolNum      CalYear    Gender        Age     Group1      Bonus  \
cluster
0        2.002040e+08  2009.532157  0.672872  38.531431  11.248308  24.395551
1        2.002011e+08  2009.499130  0.680592  38.159269  15.191036  16.392515
2        2.002029e+08  2009.522485  0.681078  30.418640  10.445796  22.446231
3        2.002008e+08  2009.502419  0.704032  34.095968  11.938710  25.548387

           Poldur         Value     Adind     Density     Expdays    nb_sin  \
cluster
0        5.046663  13655.313104  1.000000  139.501228  359.230899  1.182302
1        5.010879  32262.025674  0.352480  145.848479  357.424717  1.242385
2        4.904627  12697.281121  0.000000  137.495526  359.812514  1.236802
3        5.071774  16723.822581  0.420161  143.710930  190.174194  1.109677

            chg_sin
cluster
0        755.946857
1        916.480004
2        965.656956
3        783.191992
```

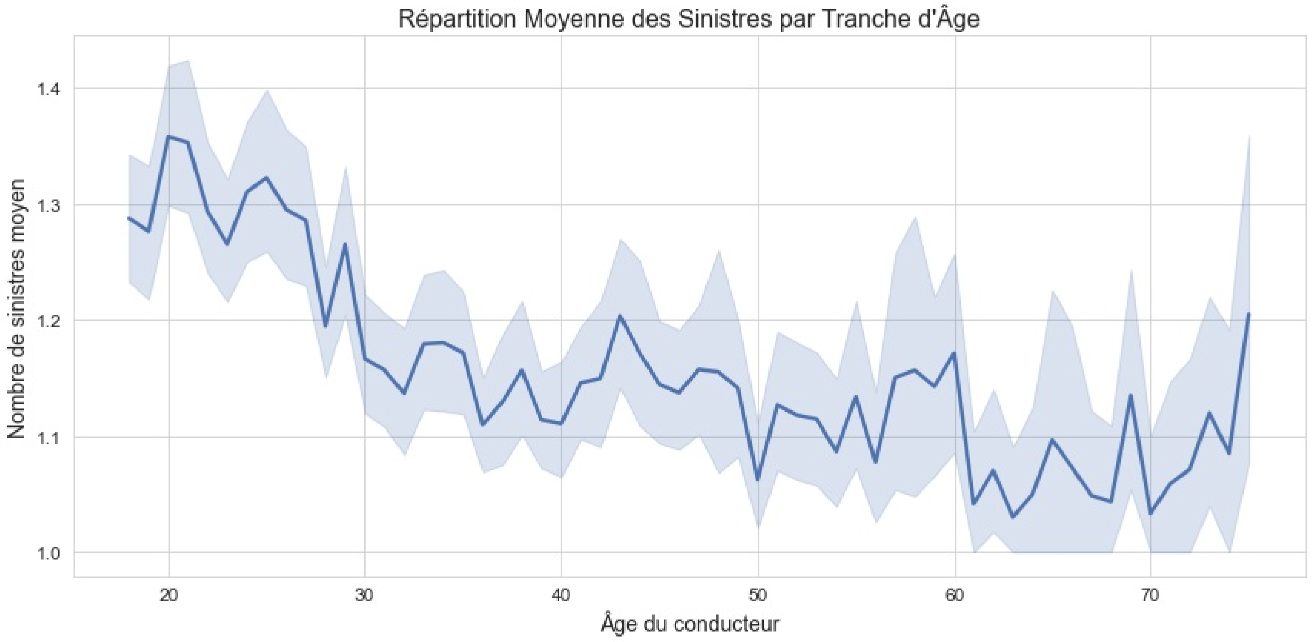
In [122…

```python
sinistre_moyen_par_age = base_Finale[["Age","nb_sin"]]
```

In [123…

```python
# Configuration du style
sns.set_style("whitegrid")
sns.set_palette("deep")
```

```python
plt.figure(figsize=[12, 6])

# Tracer le graphique avec une ligne un peu plus épaisse
sns.lineplot(x='Age', y='nb_sin', data=sinistre_moyen_par_age, lw=2.5)

# Étiquettes et titre
plt.xlabel('Âge du conducteur', fontsize=14)
plt.ylabel('Nombre de sinistres moyen', fontsize=14)
plt.title('Répartition Moyenne des Sinistres par Tranche d\'Âge', fontsize=16)

# Légèrement augmenter la taille des ticks
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Sauvegarder en haute qualité
plt.tight_layout()
plt.savefig('Âge_sinistre.png', dpi=300)

plt.show()
```



In [ ]: