Rapport

Nombre de lignes et de colonnes:

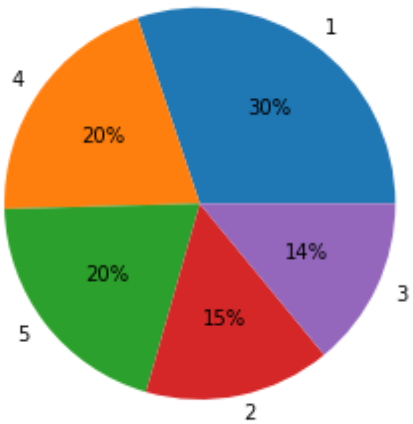
- Trainset: 24105 lignes et 5 colonnes
- Testset: 10331 lignes et 5 colonnes

Type de variables:

- variable à prédire:
    - note: quatitative discrête / qualitative ordinale (compris entre 1 et 5)
- autres variables du dataset:
    - date: date au format date1 (01 janvier 2000) 'suite à une expérience en' date2 (mois année)
    - auteur: catégorielle
    - avis: test
    - assureur: catégorielle
    - produit: catégorielle

Nombre de données manquantes:

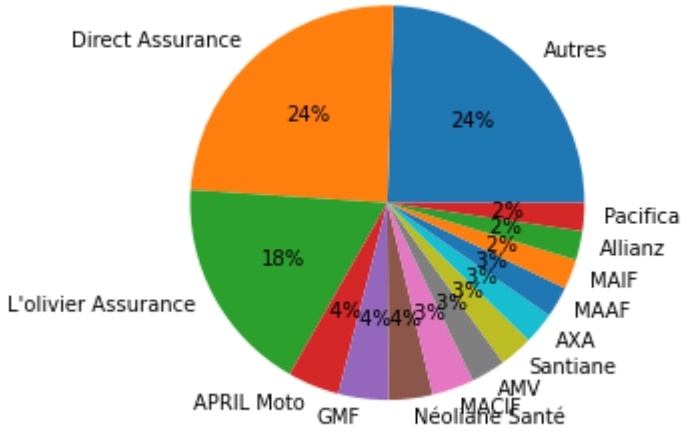
- Trainset: 2 lignes comportent des données manquantes
- Testset: 1 lignes comporte des données manquantes.
- Décision : Etant donnée que très très peu de lignes de nos datasets sont NaN je décide de les drop.



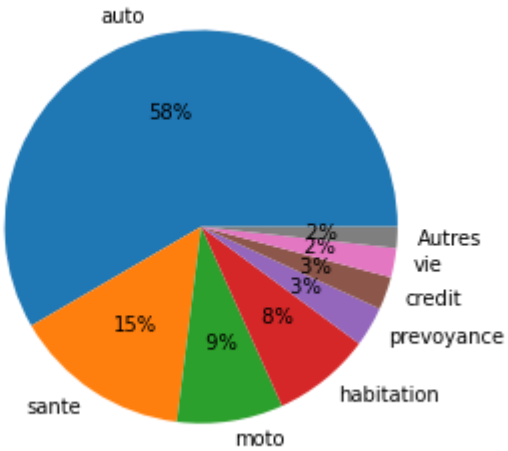
Analyses de la target:

Analyses des variables

- assureur:



- produit:



Analyses variable-target

- produit: Il semblerait que la distribution des notes ne soient pas la même pour chaque produit.
Par exemple, il semblerait que les notes correspondant aux assurances motos sont plus élevées que les autres.

- assureur: De même que pour la variable produit, les distributions des notes sembles différés d'une compagnies d'assurance à l'autre.

Préprocessing et pipeline Nous avons décidé de simplifier le texte d'avis au maximum dans une colonne clean, en enlevant les stopwords et les caractères spéciaux à l'aide de la fonction de re.sub pour rendre notre colonne avis plus homogène. Nous avons également récupérer le radical de chaque mot avec la fonction word.tokenize().

Après avoir cleaner notre colonne avis, nous créons une pipeline en utilisant une TfidfVectorizer qui permet de comptabiliser les mots en calculant la fréquence d'apparition. Ceci nous donnera la fréquence d'apparition pour les différentes catégories. Ensuite on utilise le SelectKBest qui fait un test de ki2 entre chaque variables précédentes et la variable de sortie (la note). En effet, il va venir récupérer les 10 milles variables les plus corrélées et par conséquent les plus importantes. Enfin nous avons

LinearSVC qui va effectuer la classification des différentes catégories par rapport aux variables selectionnées. Ainsi, nous récupérons le résultat de la pipeline afin d'afficher les tokens les plus fréquents dans chaque catégorie.En deuxième temps, nous cherchons le meilleur paramètre ngram_range, qui permet de définir le nombre mot maximum que l'on tokenise afin de prendre les négations et les groupes de mots afin d'avoir une vision sur le groupe de mot et non un seul mot.

Enfin, on obtimise le paramètre ngram_range du TfidfVectorizer en testant plusieur taille de token. Les meilleurs résultat sont obtenu avec 1 à 2 mots par token et 1 à 3 mots par token. Le rmse sur les données de test étant alors de 1.17.

Modèle prenant en compte les variables catégorielles du dataset

Jusque là nous avons analyser les variables categorielle mais nos modèle n'ont utiliser que la variable avis. Remédions à cela.

Nous alons donc répliquer le modèle précédent mais en prédisant cette fois les probabilités d'appartenance aux différentes classe de note plutôt que la note directement. (On utilisera ici un GradientBoostingClassifier au lieu du linearSVC car le second ne dispose pas de la méthode predict_proba)

On ajoute ensuite les variables catégorielle à partir d'un oneHotencoder. On contatènne les pipelines nlp et catégorielle puis on applique enfin un SelectKBest et un LinearSVC.

On obtient alors une rmse à la première étape (prédict proba) de 1.21 puis à la fin du modèle un rmse de 1.0921.

Observation: On se retrouve avec la même RMSE que le modèle ne prenant pas en considérant les variables catégorielles.

R&D

Dans le but de rechercher d'améliorer notre modèle nous avons décider de de tester une autre pipeline de traitement naturel basé sur le module spacy.

On commence par se demander si les adjectifs dans l'avis ne seraient pas un bon indicatif de la note qu'à donnée l'assuré. On affiche donc les adjectifs et adverbe les plus utilisé par note. Nous obtenons le résultat suivant:

Concrètement en affiant on remarque que cela ne suffit pas forcément pour discriminer les notes. En effet par exemple on obtient statisfait dans les avis de note 5 (là où il prend tout sont sens) mais aussi dans les avis 2 et 1 (là où il n'a pas vraiment ça place seul). Il est très propable qu'il serait utile de récupérer les mots de négation qui vont de paire avec les adjectifs et adverbes afin de ne pas faire de mauvaises interprétations.

- Nous avons essayer d'utiliser BERT ou plus précisément Camembert pour avoir le modèle de BERT entrainer sur des données Françaises. Cependant l'instalation d'arrivait pas à sa fin à cause d'une erreur dû au puce M1. Nous avons aussi essayer d'utiliser fast-bert mais l'erreur était malheuresement la même.

Axes d'exploration supplémentaires :

- utiliser le token.vector pour chaque mot de la phrase. Faire calculer le vecteur moyen sur la phrase et utiliser cette variables en tant qu'input dans un model dans un modèle de deep learning afin de prédict la note.

- On pourrait effectuer des tests anova pour déterminer si les distributions des différents individus de chaque variable catégorielle est significative distincte ou non.

In [21]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [88]:
```
data_train = pd.read_csv('avisassurance_train.csv', sep=";")
data_test = pd.read_csv('avisassurance_test.csv', sep=";")
```

In [89]:
```
df = data_train.copy()
df_test = data_test.copy()
```

In [ ]:
```
df.head()
```

Out[ ]:

	date	note	auteur	avis	assureur	produit
0	06 septem...	5	brahim--k-131532	Meilleurs assurances, prix, solutions, écoute,...	Direct Assurance	auto
1	03 mai 20...	4	bernard-g-112497	je suis globalement satisfait , sauf que vous ...	Direct Assurance	auto
2	21 mars 2...	5	virginie-t-107352	Prix tres abordable plusieurs options s'offren...	Direct Assurance	auto
3	10 juin 2...	4	boulain-f-116580	je satisfait du service, une réponse très rapi...	L'olivier Assurance	auto
4	29 janvie...	1	ouaille31-51798	Client depuis plus de 25 ans, très déçu de cet...	Matmut	auto

In [ ]:
```
df.isna().sum()
```

Out[ ]:
```
date        0
note        0
auteur      1
avis        1
assureur    0
produit     0
dtype: int64
```

In [ ]:
```
df.shape
```

Out[ ]:
```
(24105, 6)
```

In [ ]:
```
df.dropna(axis=0).shape
```

Out[ ]:
```
(24103, 6)
```

In [ ]:
```
df_test.shape
```

Out[ ]:
```
(10332, 5)
```

In [ ]:
```
df_test.dropna(axis=0).shape
```

Out[ ]:
```
(10331, 5)
```

In [ ]:
```
df = df.dropna(axis=0)
df_test = df.dropna(axis=0)
```

In [ ]:
```
df.dtypes
```

Out[ ]:
```
date        object
note         int64
auteur      object
avis        object
assureur    object
produit     object
dtype: object
```

In [ ]:
```
df.note.agg(['min','max'])
```

Out[ ]:
```
min    1
max    5
Name: note, dtype: int64
```

In [ ]:
```
df[df.date.str.contains('suite à une expérience en')].shape == df.shape
```

Out[ ]:
```
True
```

In [ ]:
```
df_test[df_test.date.str.contains('suite à une expérience en')].shape == df_test.shape
```

Out[ ]:
```
True
```

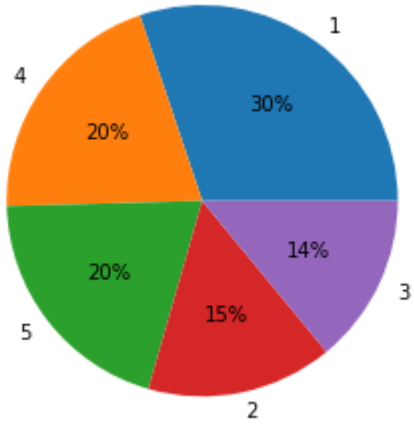
Note: on vérifie aussi que cela est vrai pour les données de test car on va rediviser la variable date en 2.

In [ ]:
```
df.note.value_counts(normalize=True).sort_index()
```

Out[ ]:
```
1    0.301664
2    0.154172
3    0.140273
4    0.202672
5    0.201220
Name: note, dtype: float64
```

In [ ]:
```
def percent_pie_chart(df, variable):
    idx = df[variable].value_counts(normalize=True)
    labels = idx.index
    sizes = idx.values

    p, tx, autotexts = plt.pie(sizes, labels=labels,
            autopct="")

    for i, a in enumerate(autotexts):
        a.set_text("{0:2.0f}%".format(sizes[i]*100))

    plt.axis('equal')
    plt.savefig('Analyse-'+variable+'.png')
    plt.show()
```

In [ ]:
```
percent_pie_chart(df, 'note')
```



Observation des variables catégorielles

In [ ]:
```
df.auteur.value_counts()
```

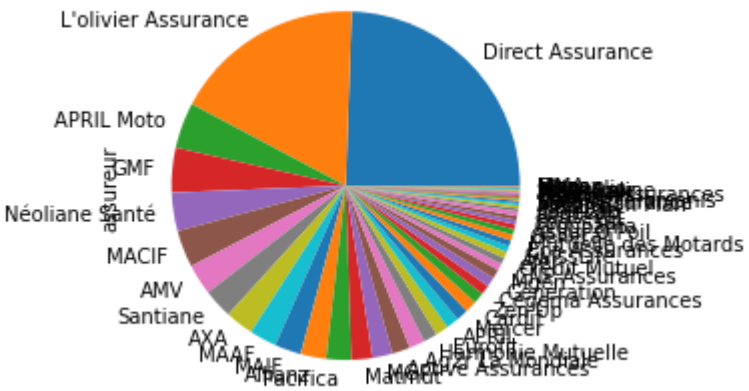
Out[ ]:
```
mikado-50419             4
mm-53953                 4
pj-97010                 4
tontonbubu-54321         4
lotus5-51217             3
                        ..
elya973-80260            1
jean-pierre-s-137446     1
sass-117513              1
marco14-138660           1
jesse-51459              1
Name: auteur, Length: 23675, dtype: int64
```

In [ ]:
```
df.assureur.unique()
```

Out[ ]:
```
array(['Direct Assurance', "L'olivier Assurance", 'Matmut',
       'Néoliane Santé', 'APRIL', 'SantéVet', 'Mercer', 'Generali',
       'Allianz', 'APRIL Moto', 'Cegema Assurances', 'LCL', 'Afer',
       'Pacifica', 'SwissLife', 'MAAF', 'Solly Azar', 'GMF', 'AMV',
       'CNP Assurances', 'MAIF', 'Sogecap', 'Harmonie Mutuelle',
       'Mutuelle des Motards', 'MACIF', 'Eurofil', 'Active Assurances',
       'AXA', 'Sogessur', 'Ag2r La Mondiale', 'Mgen', "Zen'Up", 'MGP',
       'Intériale', 'Génération', 'Cardif', 'Santiane', 'Eca Assurances',
       'Groupama', "Assur O'Poil", 'MMA', 'MetLife', 'Crédit Mutuel',
       'Afi Esca', 'Gan', 'Magnolia', 'Suravenir', 'Assur Bon Plan',
       'AssurOnline', 'Carac', 'Mapa', 'Malakoff Humanis',
       'Euro-Assurance', 'Peyrac Assurances', 'Sma', 'Hiscox'],
      dtype=object)
```

In [ ]:
```
df.assureur.value_counts().plot.pie()
```

Out[ ]:
```
<AxesSubplot:ylabel='assureur'>
```



Faisons un pie chart plus lissible.

In [ ]:
```
df.assureur.value_counts(normalize=True).sort_values(ascending=False)
```

Out[ ]:
```
Direct Assurance         0.244617
L'olivier Assurance      0.177903
APRIL Moto               0.042443
GMF                      0.041364
Néoliane Santé           0.035722
MACIF                    0.034394
AMV                      0.028544
Santiane                 0.027300
AXA                      0.026096
MAAF                     0.025391
MAIF                     0.024520
Allianz                  0.023814
Pacifica                 0.022943
Matmut                   0.019666
MGP                      0.018504
Active Assurances        0.016720
Ag2r La Mondiale         0.014521
Harmonie Mutuelle        0.012944
Eurofil                  0.011990
APRIL                    0.011741
Mercer                   0.011077
Cardif                   0.010994
Zen'Up                   0.010165
Cegema Assurances        0.008879
Génération               0.008796
Mgen                     0.008671
CNP Assurances           0.006099
Crédit Mutuel            0.006057
Afer                     0.005974
SwissLife                0.005725
Eca Assurances           0.005476
Mutuelle des Motards     0.005311
Generali                 0.005311
Assur O'Poil             0.004564
Groupama                 0.004024
Sogessur                 0.003983
SantéVet                 0.003983
Sogecap                  0.002987
Intériale                0.002572
MetLife                  0.002199
Assur Bon Plan           0.002116
Euro-Assurance           0.001784
Malakoff Humanis         0.001577
Gan                      0.001369
Carac                    0.001328
Peyrac Assurances        0.001162
Suravenir                0.001120
Afi Esca                 0.001079
AssurOnline              0.001037
Solly Azar               0.001037
Magnolia                 0.000788
LCL                      0.000747
Mapa                     0.000415
Sma                      0.000249
MMA                      0.000166
Hiscox                   0.000041
Name: assureur, dtype: float64
```

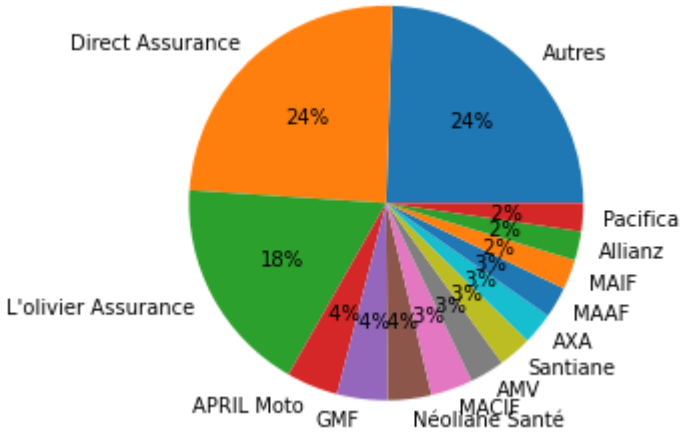
In [ ]:
```
df.assureur.value_counts(normalize=True).loc[lambda x: x>0.02].index
```

Out[ ]:
```
Index(['Direct Assurance', 'L'olivier Assurance', 'APRIL Moto', 'GMF',
       'Néoliane Santé', 'MACIF', 'AMV', 'Santiane', 'AXA', 'MAAF', 'MAIF',
       'Allianz', 'Pacifica'],
      dtype='object')
```

In [ ]:
```
def filter_percent_threshold(df, variable, threshold):
    df_ = df.copy()
    counts = df_[variable].value_counts(normalize=True)
```

```
    idx = counts[counts.lt(threshold)].index

    df_.loc[df_[variable].isin(idx), variable] = 'Autres'
    return df_
```

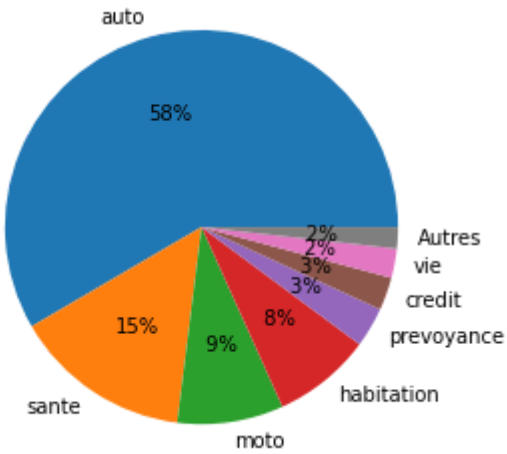
In [ ]:
```
df_plot = filter_percent_threshold(df, 'assureur', 0.02)
percent_pie_chart(df_plot, 'assureur')
```



In [ ]:
```
df.produit.unique()
```

Out[ ]:
```
array(['auto', 'sante', 'animaux', 'vie', 'moto', 'credit', 'habitation',
       'prevoyance', 'responsabilite-civile-professionnelle',
       'multirisque-professionnelle', 'assurances-professionnelles',
       'garantie-decennale', 'flotte-automobile'], dtype=object)
```

In [ ]:
```
df_plot = filter_percent_threshold(df, 'produit', 0.02)
percent_pie_chart(df_plot, 'produit')
```



Analyse variable-target

In [85]:
```
from wordcloud import WordCloud, ImageColorGenerator
import matplotlib.pyplot as plt
from PIL import Image
```

In [86]:
```
!pip install wordcloud
```

```
Requirement already satisfied: wordcloud in /usr/local/lib/python3.7/dist-packages (1.
5.0)
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.7/dist-packages
(from wordcloud) (1.19.5)
Requirement already satisfied: pillow in /usr/local/lib/python3.7/dist-packages (from w
ordcloud) (7.1.2)
```

In [90]:
```
df.avis
```

Out[90]:
```
0        Meilleurs assurances, prix, solutions, écoute,...
1        je suis globalement satisfait , sauf que vous ...
2        Prix tres abordable plusieurs options s'offren...
3        je satisfait du service, une réponse très rapi...
4        Client depuis plus de 25 ans, très déçu de cet...
                              ...
24100    Assurance moto chez la mutuel des motards en F...
24101    Même les demandes les plus simples n'aboutisse...
24102    En décembre 2019, j'ai souscrit à un contrat C...
24103    Je suis assurer à la gmf depuis plus de 15 ans...
24104    Bonjour\r\nMon ami vient de se faire voler sa ...
Name: avis, Length: 24105, dtype: object
```

In [106…
```
wc = WordCloud(background_color="white")
```

In [100…
```
Text = ''.join([str(text) for _,text in df.avis.items()])
```

In [107…
```
wc.generate(Text)
```

Out[107…
```
<wordcloud.wordcloud.WordCloud at 0x7ff732ba0950>
```

In [110…
```
plt.figure(figsize=(12,8))
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Produit:

In [ ]:
```
results = df.pivot_table(index='produit', columns='note',
                         aggfunc={"note":['count']})
results
```

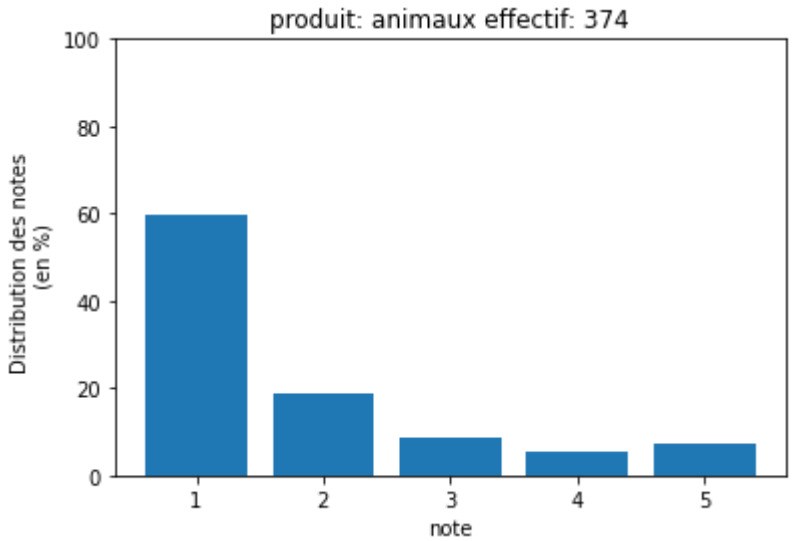
Out[ ]:

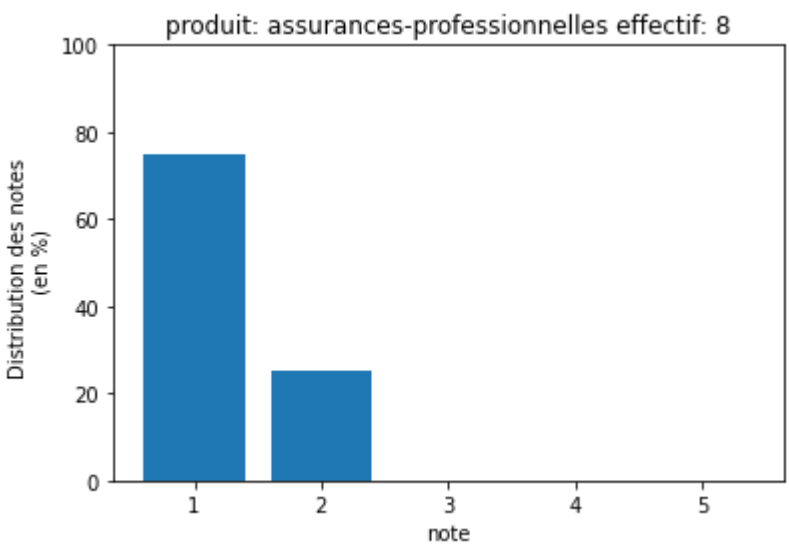
	note
	count
note	
1	
2	
3	
4	
5	

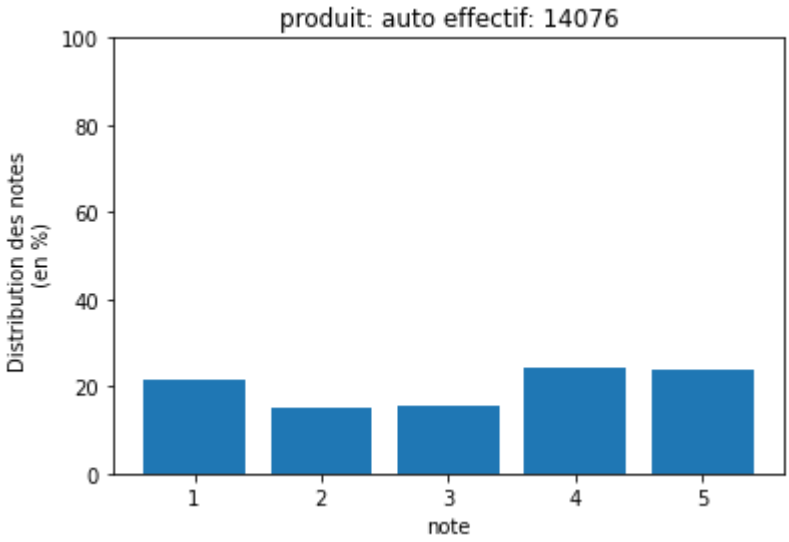
produit	note				
	count				
note	1	2	3	4	5
produit					
animaux	223.0	70.0	33.0	21.0	27.0
assurances-professionnelles	6.0	2.0	NaN	NaN	NaN
auto	3021.0	2093.0	2158.0	3426.0	3378.0
credit	269.0	97.0	46.0	77.0	164.0
flotte-automobile	1.0	NaN	NaN	NaN	NaN
garantie-decennale	10.0	NaN	NaN	1.0	1.0
habitation	1073.0	534.0	195.0	95.0	59.0
moto	318.0	267.0	267.0	548.0	705.0
multirisque-professionnelle	8.0	1.0	NaN	8.0	3.0
prevoyance	522.0	119.0	80.0	34.0	35.0
responsabilite-civile-professionnelle	5.0	NaN	NaN	NaN	NaN
sante	1408.0	443.0	539.0	663.0	472.0
vie	407.0	90.0	63.0	12.0	6.0

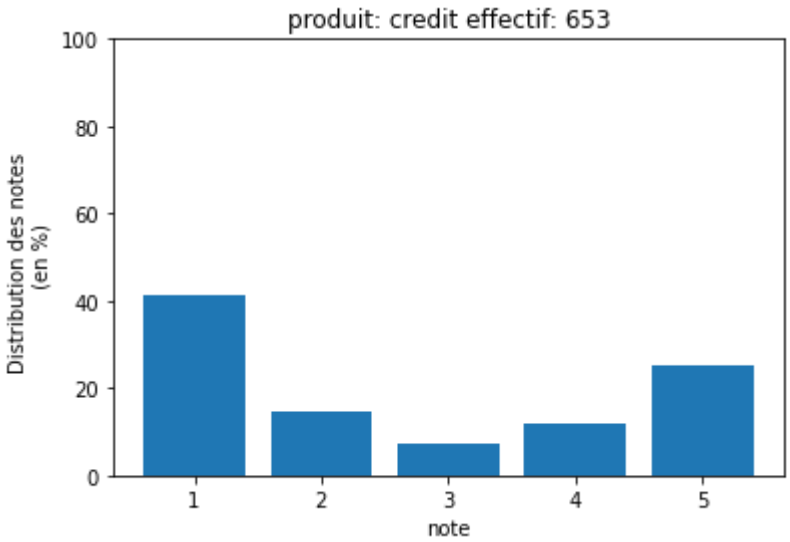
In [ ]:

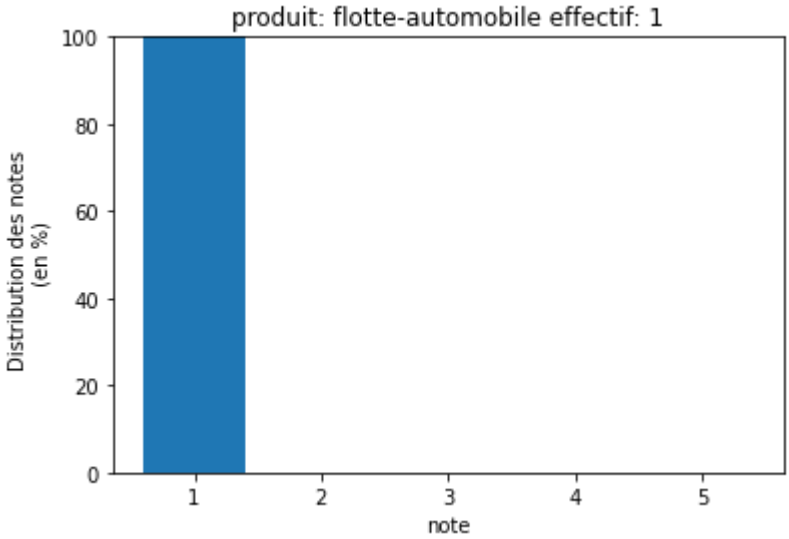
```
for produit in results.index:
    counts = pd.Series(results.loc[produit, :].values)
    counts = counts.fillna(0)
    eff = sum(counts)
    counts = 100*counts / eff

    fig, ax = plt.subplots()
    plt.bar(range(1,6), counts)
    ax.set_ylim(0,100)
    plt.title('produit: ' + produit + ' effectif: '+str(int(eff)))
    plt.ylabel('Distribution des notes \n (en %)')
    plt.xlabel('note')
    plt.show()
```

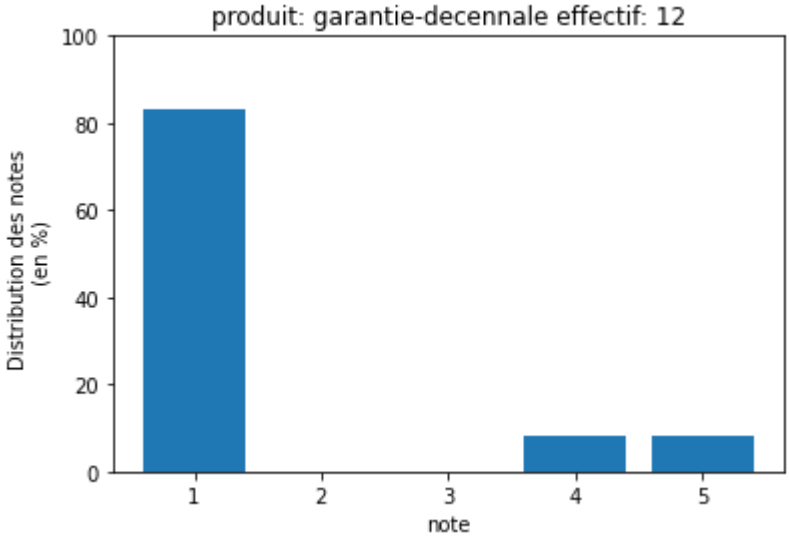


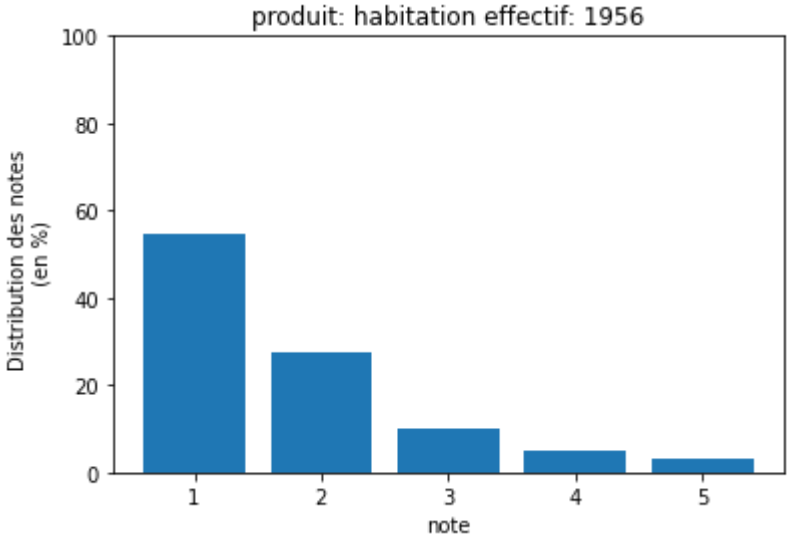

produit: assurances-professionnelles effectif: 8
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5

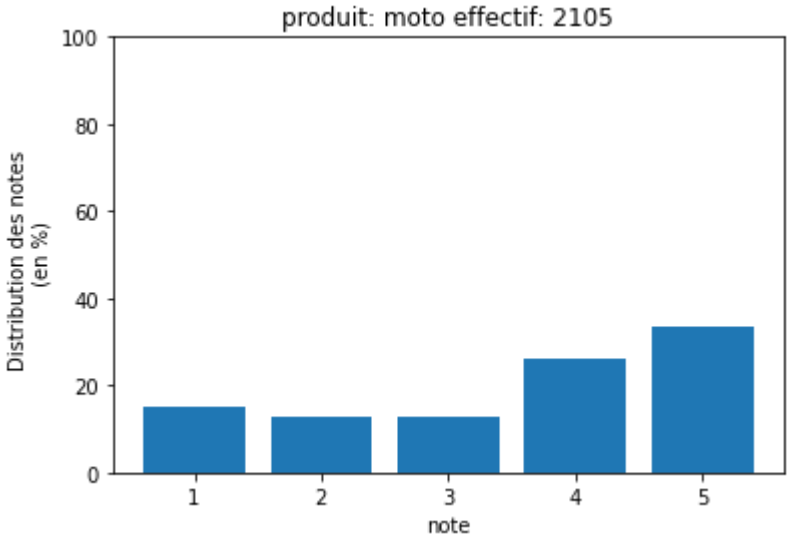

produit: auto effectif: 14076
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5

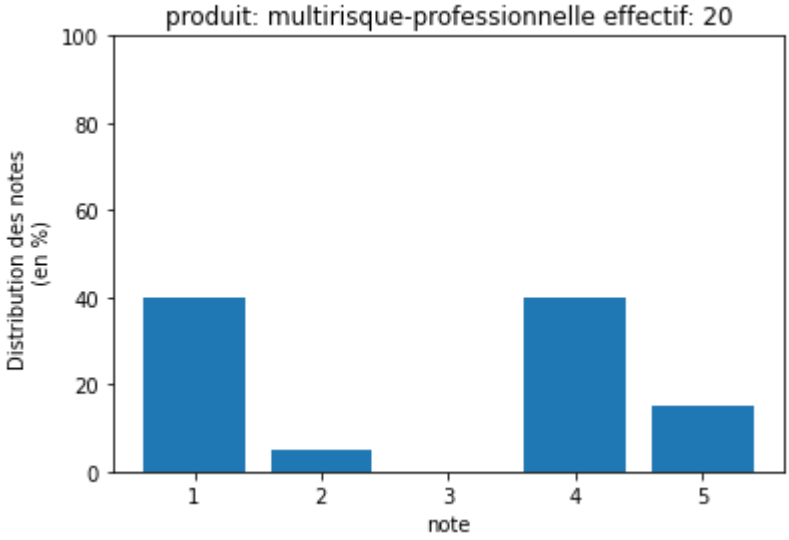

produit: credit effectif: 653
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5

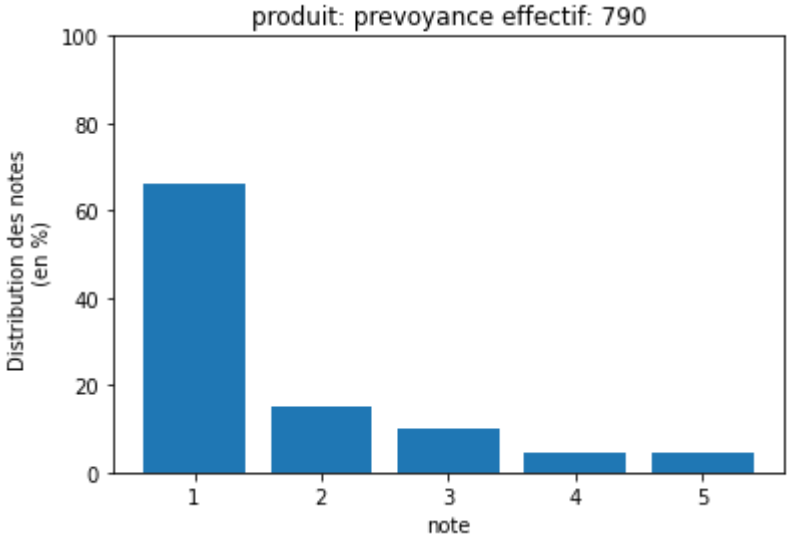

produit: flotte-automobile effectif: 1
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5

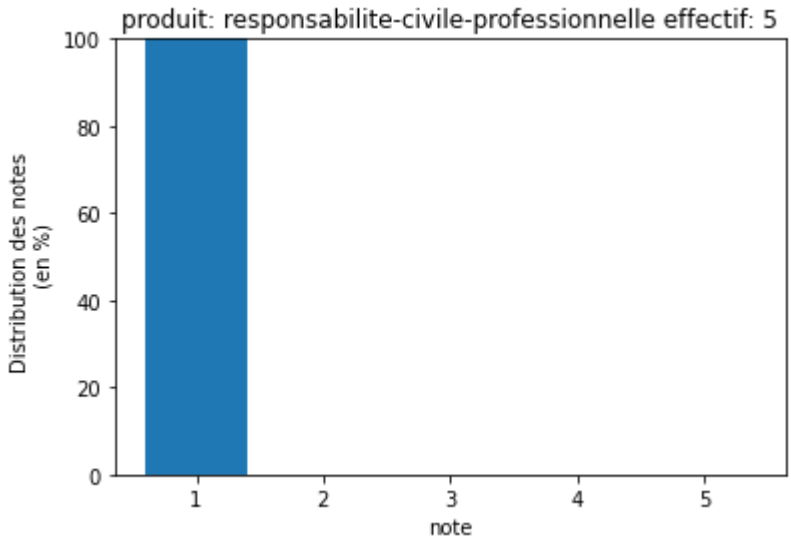

produit: garantie-decennale effectif: 12
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5

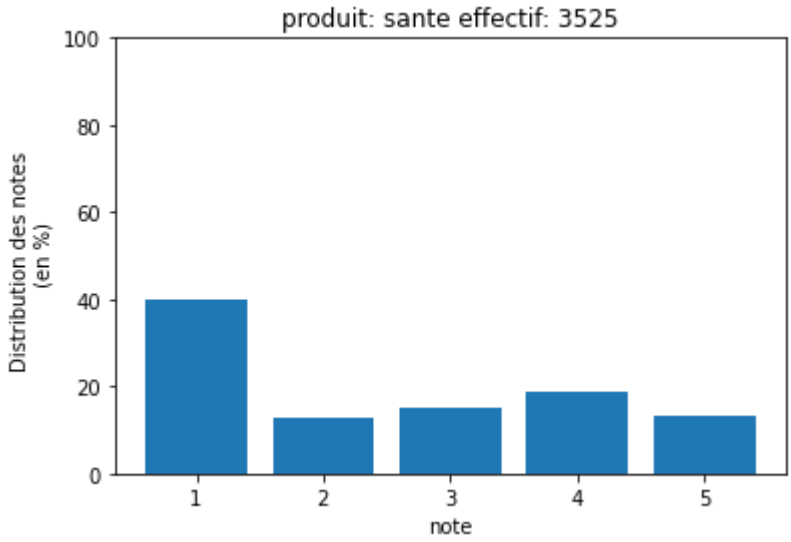

produit: habitation effectif: 1956
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5

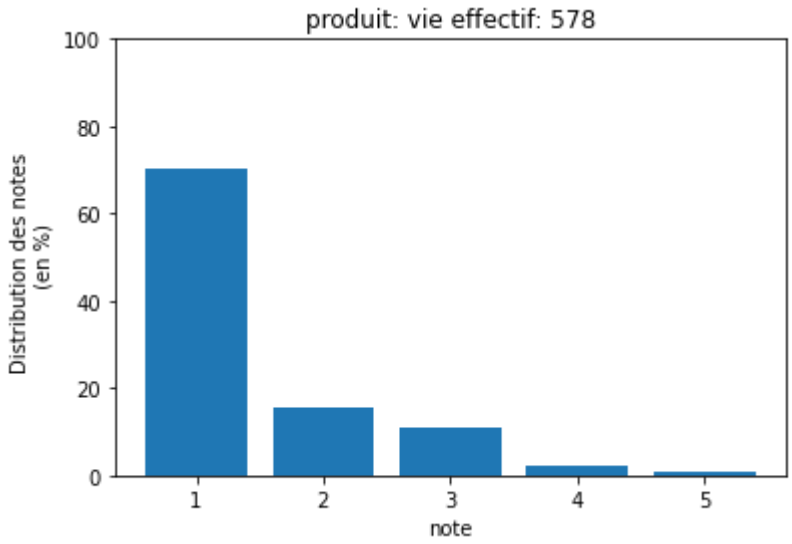

produit: moto effectif: 2105
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5


produit: multirisque-professionnelle effectif: 20
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5


produit: prevoyance effectif: 790
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5







Assureurs

In [ ]:

```
df_plot = df[df.assureur.isin(df.assureur.value_counts(normalize=True).loc[lambda x: x>
results = df_plot.pivot_table(index='assureur', columns='note',
                              aggfunc={"note":['count']})
results
```

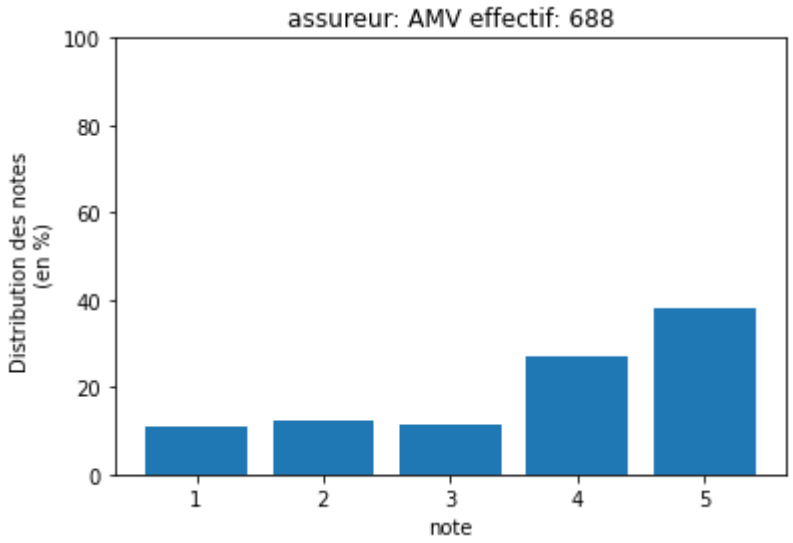
Out[ ]:

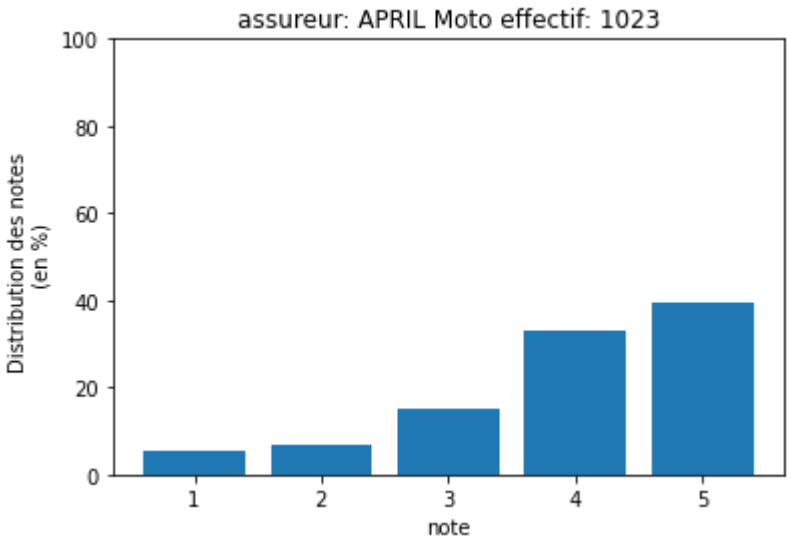
	note
	count
note	1 2 3 4 5
assureur	

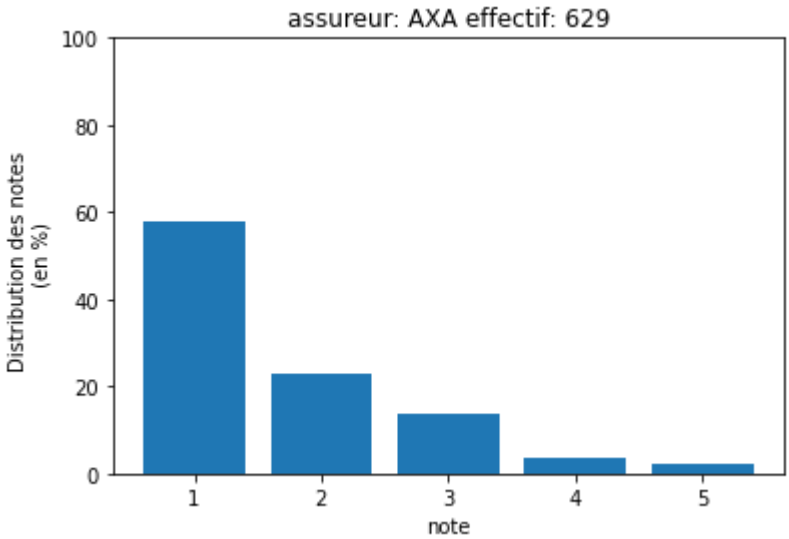
	note				
	count				
note	1	2	3	4	5
assureur					
AMV	74	86	80	186	262
APRIL Moto	56	70	152	340	405
AXA	363	143	86	22	15
Allianz	383	112	51	18	10
Direct Assurance	959	767	1111	1577	1482
GMF	215	220	180	227	155
L'olivier Assurance	383	332	530	1430	1613
MAAF	290	189	86	31	16
MACIF	423	250	81	43	32
MAIF	314	145	69	41	22
Néoliane Santé	267	106	166	180	142
Pacifica	289	122	56	63	23
Santiane	85	57	140	217	159

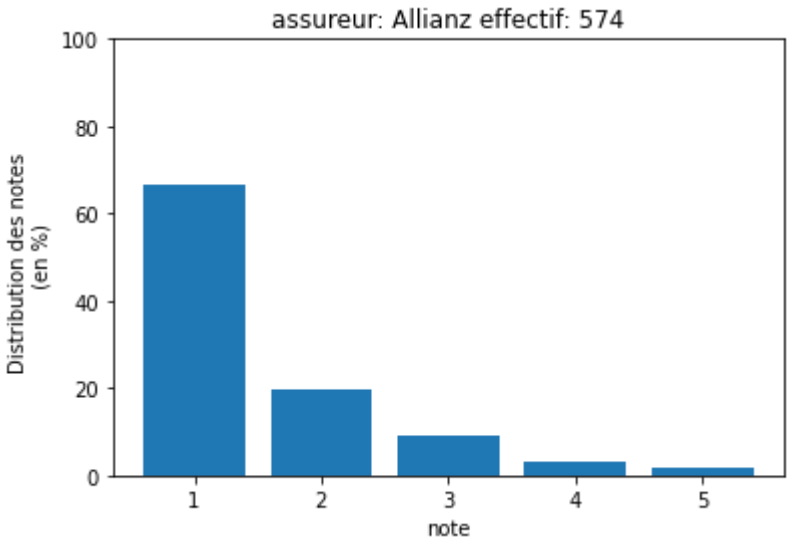
In [ ]:

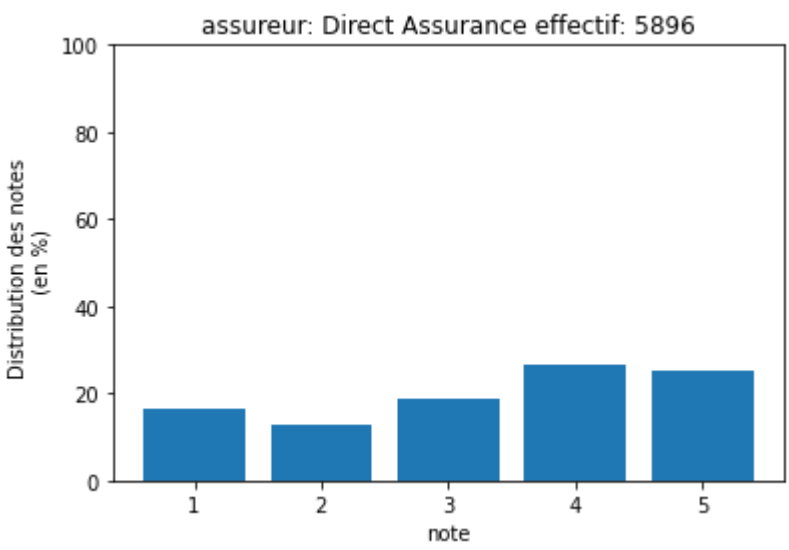
```
for assureur in results.index:
    counts = pd.Series(results.loc[assureur, :].values)
    counts = counts.fillna(0)
    eff = sum(counts)
    counts = 100*counts / eff

    fig, ax = plt.subplots()
    plt.bar(range(1,6), counts)
    ax.set_ylim(0,100)
    plt.title('assureur: ' + assureur + ' effectif: '+str(int(eff)))
    plt.ylabel('Distribution des notes \n (en %)')
    plt.xlabel('note')
    plt.show()
```

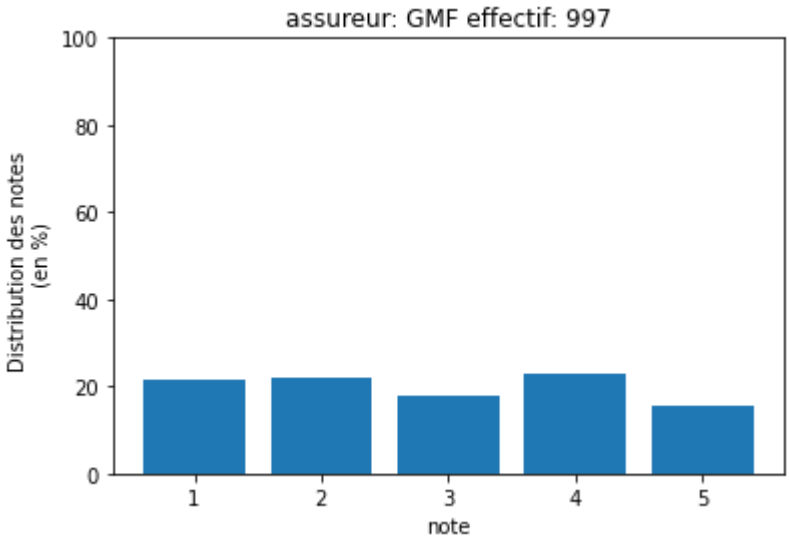


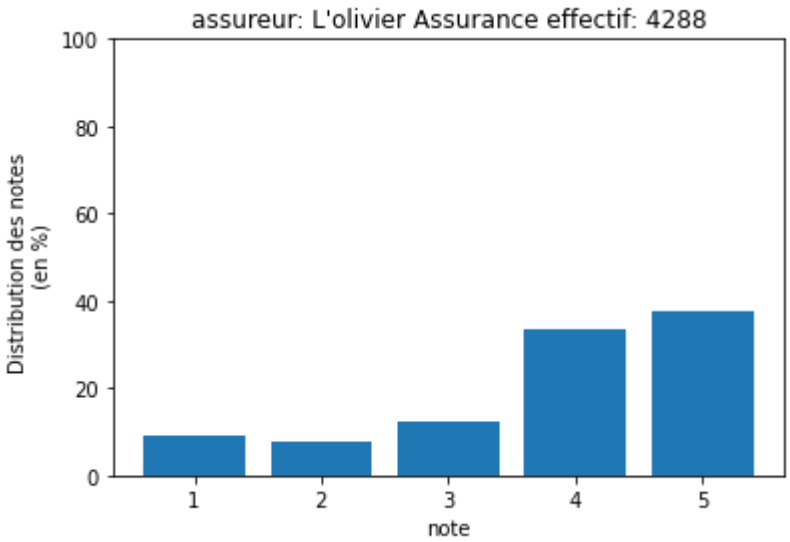

assureur: APRIL Moto effectif: 1023
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5

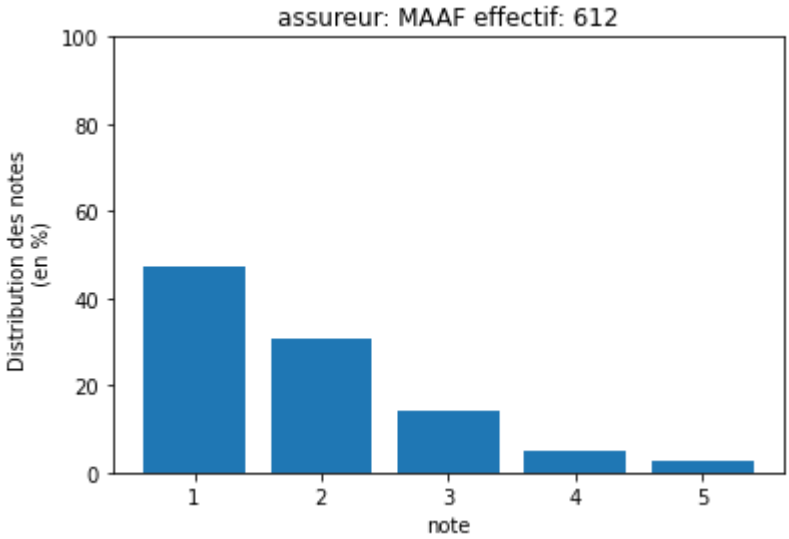

assureur: AXA effectif: 629
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5

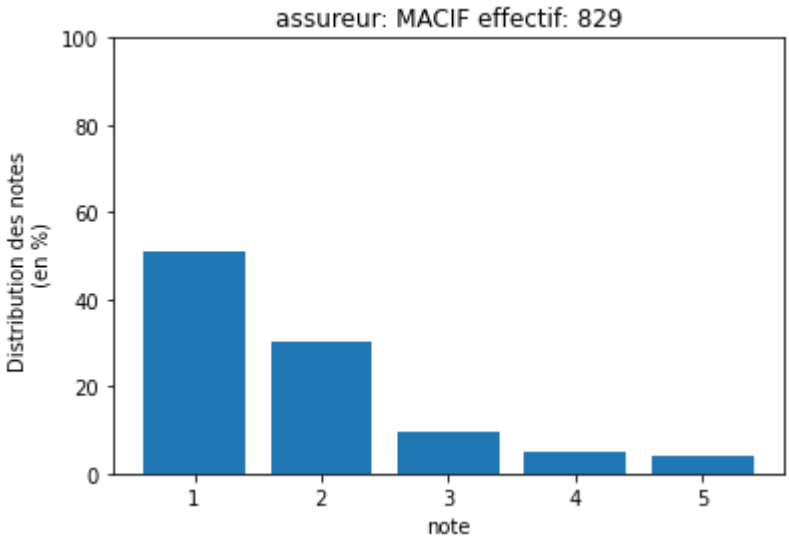

assureur: Allianz effectif: 574
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5

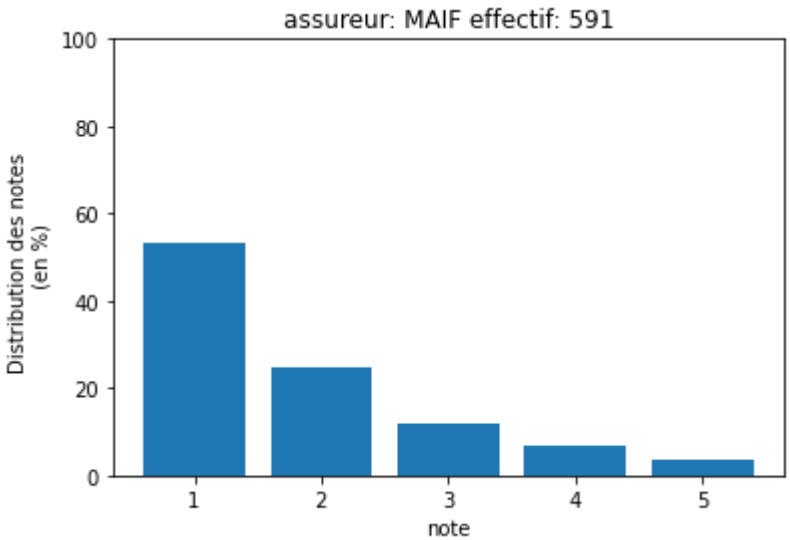

assureur: Direct Assurance effectif: 5896
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5

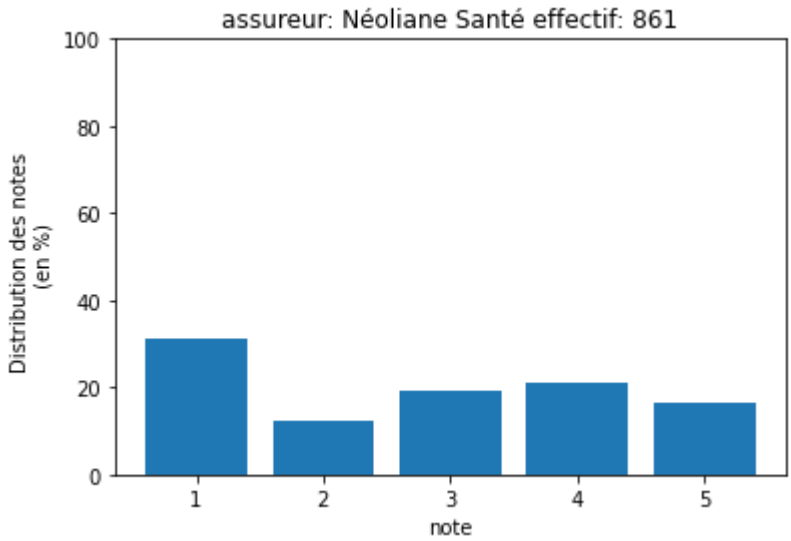

assureur: GMF effectif: 997
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5

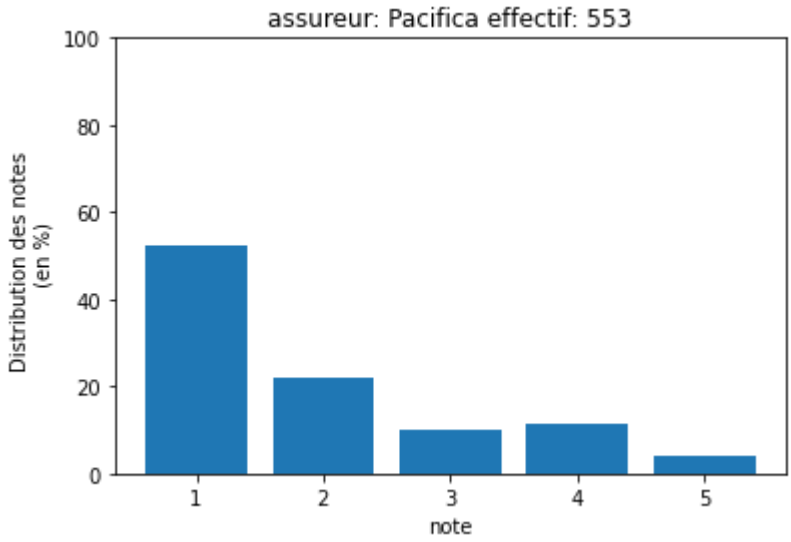

assureur: L'olivier Assurance effectif: 4288
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5

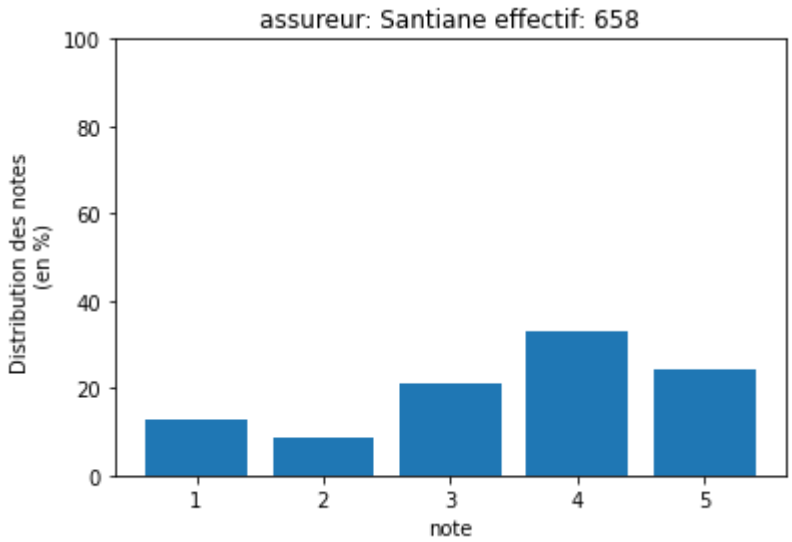

assureur: MAAF effectif: 612
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5


assureur: MACIF effectif: 829
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5


assureur: MAIF effectif: 591
Distribution des notes (en %)
note
0
20
40
60
80
100
1
2
3
4
5







Modèle

In [54]:

```
from sklearn.metrics import mean_squared_error

from nltk.corpus import stopwords
from nltk import word_tokenize
from nltk.stem import SnowballStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
import re

from sklearn.svm import LinearSVC
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, chi2
```

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
```

In [ ]:
```
def netoyage_text_avis(A):
    # on enlève les stopwords et les carractères spéciaux
    B = [ i for i in  re.sub("[^a-zà-ÿ]", " ", str(A)).split() if i not in stop_words_l
    #C = [ i.replace(',','') for i in B]
    # On récupère le radical de chaque mot
    C = [[fr.stem(word) for word in word_tokenize(B[i])][0] for i in range(len(B))]
    # On reconstitue la phrase
    D = ' '.join(C)
    return D
```

In [56]:
```
stemmer = SnowballStemmer('french')
import nltk
nltk.download('stopwords')
stop_words_list = stopwords.words("french")
fr = SnowballStemmer('french')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In [ ]:
```
data_train['cleaned'] = data_train['avis'].apply(lambda x: netoyage_text_avis(x))

X_train, X_test, y_train, y_test = train_test_split(data_train['cleaned'], data_train.r

pipeline = Pipeline([('vect', TfidfVectorizer(ngram_range=(1,1), stop_words=stop_words_
                     ('chi',  SelectKBest(chi2, k=10000)),
                     ('clf', LinearSVC(C=1.0, penalty='l1', max_iter=3000, dual=False))
```

In [ ]:
```
model = pipeline.fit(X_train, y_train)

vectorizer = model.named_steps['vect']
chi = model.named_steps['chi']
clf = model.named_steps['clf']

feature_names = vectorizer.get_feature_names()
feature_names = [feature_names[i] for i in chi.get_support(indices=True)]
feature_names = np.asarray(feature_names)

target_names = ['1', '2', '3', '4', '5']
print("top 10 keywords per class:")
for i, label in enumerate(target_names):
    top10 = np.argsort(clf.coef_[i])[-10:]
    print("%s: %s" % (label, " ".join(feature_names[top10])))

print("accuracy score: " + str(model.score(X_test, y_test)))
```

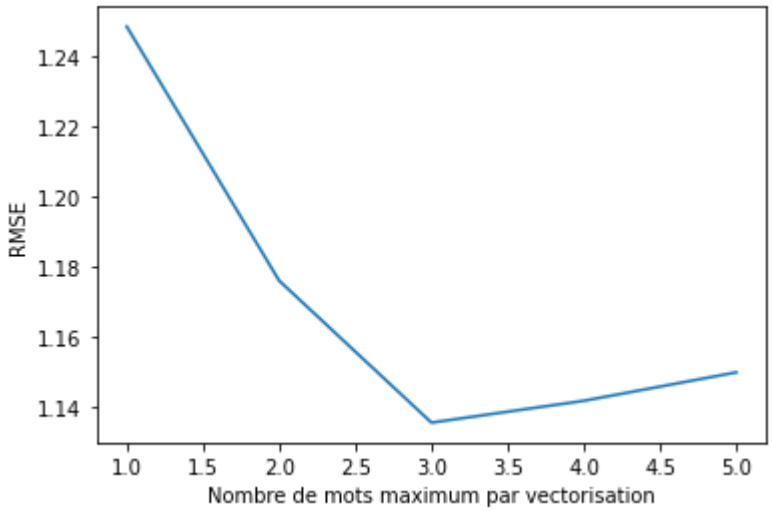
```
top 10 keywords per class:
1: ompte luis ridicul benghal quelquet déduis détach polo nce contactent
2: égarent abinet échapp traumatis sacrifi lex involontair tal thyroid aper
3: interviendr cheneau bobos lav plag alde ortugal élarg audienc enant
4: éger jesper fournitur sv felicit visualis expliquent périmetr chaton soupl
5: céler beurr luid poyr incompar conclur execut vot mme urofyl
accuracy score: 0.5042522298278366
/opt/homebrew/Caskroom/miniforge/base/lib/python3.9/site-packages/sklearn/utils/depreca
tion.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names
is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out inste
ad.
  warnings.warn(msg, category=FutureWarning)
```

Recherche du meilleur parramètre TfidfVectorizer ngram_range

In [ ]:
```
Score = []
Parami = []

for i in range(1, 6):
    pipeline = Pipeline([
        ('vect', TfidfVectorizer(ngram_range=(1,i), stop_words=stop_words_list, subline
        ('chi',  SelectKBest(chi2, k=10000)),
        ('clf', LinearSVC(C=1.0, penalty='l1', max_iter=3000, dual=False))
    ])
    model = pipeline.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    RMSE = mean_squared_error(y_test, y_pred)
    Score.append(RMSE)
    Parami.append(i)
```

In [ ]:
```
plt.plot(Parami, Score)
plt.xlabel('Nombre de mots maximum par vectorisation')
plt.ylabel('RMSE')
```

Out[ ]:
```
Text(0, 0.5, 'RMSE')
```



Prédiction :

In [ ]:
```
data_to_predict = pd.read_csv('avisassurance_test.csv', sep=';')

X = data_train['cleaned']
y = data_train.note

data_to_predict['cleaned'] = data_to_predict['avis'].apply(lambda x: netoyage_text_avis

X_data_to_predict = data_to_predict['cleaned']

pipeline = Pipeline([('vect', TfidfVectorizer(ngram_range=(1,2), stop_words=stop_words_
                     ('chi',  SelectKBest(chi2, k=10000)),
                     ('clf', LinearSVC(C=1.0, penalty='l1', max_iter=3000, dual=False))
model = pipeline.fit(X, y) # On entraine le modèle sur l'ensemble des données de test
y_predict = model.predict(X_data_to_predict)
Data_to_save = data_to_predict.copy()
Data_to_save['note'] = y_predict
Data_to_save.to_csv('model1_predictions.csv', sep=';')
```

Ajout des variables catégorielles au modèle.

In [45]:
```
# Pipeline
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.compose import make_column_selector, make_column_transformer, ColumnTransf
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report
```

In [57]:
```
avis_pipeline = Pipeline([
        ('vect', TfidfVectorizer(ngram_range=(1,2), stop_words=stop_words_list, subline
        ('chi',  SelectKBest(chi2, k=10000)),
        ('clf', GradientBoostingClassifier(criterion='squared_error'))
    ])
categorical_pipeline = Pipeline(steps=[( 'onehot',OneHotEncoder(handle_unknown = 'ignor
```

In [58]:
```
preprocessor = ColumnTransformer(
transformers=[
                ('avis-pipeline', avis_pipeline, ['avis']),
                ('cat', categorical_pipeline, ['auteur', 'assureur', 'produit'])
])
```

In [59]:
```
model = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('rfc', RandomForestClassifier())
    ])
```

In [60]:
```
df = pd.read_csv('avisassurance_train.csv', sep=";")
data = df.copy().dropna(axis=0)
X = data.copy().drop(['note'], axis=1)
y = data.note.copy()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [61]:
```
len(X), len(y)
```

Out[61]: (24103, 24103)

In [62]:
```
len(X_train), len(y_train)
```

Out[62]: (19282, 19282)

In [63]:
```
X_train.shape, y_train.shape
```

Out[63]: ((19282, 5), (19282,))

In [64]:
```
y_train = np.array(y_train)
```

In [65]:
```
cat_features = ['auteur', 'assureur', 'produit']

cat_model = categorical_pipeline.fit(X_train[cat_features])
X_train_cat = cat_model.transform(X_train[cat_features])
X_train_cat
```

Out[65]:
```
<19282x19081 sparse matrix of type '<class 'numpy.float64'>'
        with 57846 stored elements in Compressed Sparse Row format>
```

In [66]:
```
nlp_model = avis_pipeline.fit(X_train['avis'], y_train)
X_train_avis_prob = nlp_model.predict_proba(X_train['avis'])
X_train_avis_prob
```

Out[66]:
```
array([[0.03403386, 0.03305624, 0.10317184, 0.40216813, 0.42756993],
       [0.36341052, 0.21925767, 0.18674615, 0.13144566, 0.09913999],
       [0.46326553, 0.38019261, 0.08256055, 0.04034914, 0.03363216],
       ...,
       [0.05425919, 0.04649799, 0.29640994, 0.33895022, 0.26388265],
       [0.32466938, 0.24419712, 0.18745806, 0.15712396, 0.08655148],
       [0.48946832, 0.30578189, 0.10638948, 0.05598895, 0.04237135]])
```

In [67]:
```
X_train_avis_prob.shape
```

Out[67]:
```
(19282, 5)
```

In [68]:
```
y_pred = nlp_model.predict(X_test['avis'])
```

In [69]:
```
print(classification_report(y_test, y_pred))
print('RMSE : ',mean_squared_error(y_test, y_pred, squared=False))
```

```
              precision    recall  f1-score   support

           1       0.55      0.87      0.67      1426
           2       0.35      0.09      0.14       732
           3       0.34      0.11      0.16       658
           4       0.41      0.43      0.42       990
           5       0.51      0.57      0.54      1015

    accuracy                           0.49      4821
   macro avg       0.43      0.41      0.39      4821
weighted avg       0.45      0.49      0.44      4821

RMSE :  1.2346548249685365
```

In [70]:
```
X_train_cat.toarray().shape
```

Out[70]:
```
(19282, 19081)
```

In [71]:
```
print(X_train_cat.shape)
print(X_train_avis_prob.shape)
A1 = np.array(X_train_avis_prob)
A2 = X_train_cat.toarray()
print(A1.shape, A2.shape)
X_merge_train = np.hstack((A1, A2))
X_merge_train
```

```
(19282, 19081)
(19282, 5)
(19282, 5) (19282, 19081)
```

Out[71]:
```
array([[0.03403386, 0.03305624, 0.10317184, ..., 0.        , 0.        ,
        0.        ],
       [0.36341052, 0.21925767, 0.18674615, ..., 0.        , 1.        ,
        0.        ],
       [0.46326553, 0.38019261, 0.08256055, ..., 0.        , 1.        ,
        0.        ],
       ...,
       [0.05425919, 0.04649799, 0.29640994, ..., 0.        , 0.        ,
        0.        ],
       [0.32466938, 0.24419712, 0.18745806, ..., 0.        , 0.        ,
        0.        ],
       [0.48946832, 0.30578189, 0.10638948, ..., 0.        , 0.        ,
        0.        ]])
```

In [72]:
```
X_test_avis_prob = nlp_model.predict_proba(X_test['avis'])
X_test_cat = cat_model.transform(X_test[cat_features])

print(X_train_cat.shape)
```

```
print(X_train_avis_prob.shape)
A1 = np.array(X_test_avis_prob)
A2 = X_test_cat.toarray()
print(A1.shape, A2.shape)
X_merge_test = np.hstack((A1, A2))
X_merge_test
```

```
(19282, 19081)
(19282, 5)
(4821, 5) (4821, 19081)
```

Out[72]:
```
array([[0.04975827, 0.05980354, 0.14897091, ..., 0.        , 0.        ,
        0.        ],
       [0.02919449, 0.02468054, 0.11054581, ..., 0.        , 0.        ,
        0.        ],
       [0.0776624 , 0.06186106, 0.13881924, ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.20901816, 0.09429646, 0.12725754, ..., 0.        , 1.        ,
        0.        ],
       [0.55741833, 0.21425795, 0.10816404, ..., 0.        , 0.        ,
        0.        ],
       [0.30250555, 0.18429215, 0.20013908, ..., 0.        , 0.        ,
        0.        ]])
```

In [73]:
```
X_merge_train.shape
```

Out[73]:
```
(19282, 19086)
```

In [74]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [75]:
```
scaler = MinMaxScaler()
sclaer = scaler.fit(X_merge_train)
X_merge_train_scaled = scaler.transform(X_merge_train)
X_merge_test_scaled = scaler.transform(X_merge_test)
```

In [76]:
```
X_merge_train_scaled
```

Out[76]:
```
array([[0.03089974, 0.03894753, 0.10571008, ..., 0.        , 0.        ,
        0.        ],
       [0.38036235, 0.2973646 , 0.20027852, ..., 0.        , 1.        ,
        0.        ],
       [0.4863067 , 0.52071592, 0.08238739, ..., 0.        , 1.        ,
        0.        ],
       ...,
       [0.05235845, 0.05760248, 0.32436849, ..., 0.        , 0.        ,
        0.        ],
       [0.33925872, 0.33197647, 0.20108408, ..., 0.        , 0.        ,
        0.        ],
       [0.51410737, 0.41744603, 0.109351  , ..., 0.        , 0.        ,
        0.        ]])
```

In [77]:
```
model = Pipeline([
    ('chi',  SelectKBest(chi2, k=100)),
    ('lSVC',  LinearSVC())
])
model.fit(X_merge_train_scaled, y_train)
y_pred = model.predict(X_merge_test_scaled)
print(mean_squared_error(y_test, y_pred, squared=False))
```

```
1.0974126195654097
```

In [83]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
import xgboost as xgb
```

```
model=xgb.XGBRegressor()
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
print("Accuracy: ", model.score(X_test,y_test))
print("MSE: ", mean_squared_error(y_pred,y_test))
print("RMSE: ", np.sqrt(mean_squared_error(y_pred,y_test)))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-83-9cb3c1968378> in <module>()
      4
      5 model=xgb.XGBRegressor()
----> 6 model.fit(X_train,y_train)
      7 y_pred=model.predict(X_test)
      8 print("Accuracy: ", model.score(X_test,y_test))

/usr/local/lib/python3.7/dist-packages/xgboost/sklearn.py in fit(self, X, y, sample_weight, eval_set, eval_metric, early_stopping_rounds, verbose, xgb_model, sample_weight_eval_set, callbacks)
    358                                     missing=self.missing, nthread=self.n_jobs)
    359         else:
--> 360             trainDmatrix = DMatrix(X, label=y, missing=self.missing, nthread=self.n_jobs)
    361
    362         evals_result = {}

/usr/local/lib/python3.7/dist-packages/xgboost/core.py in __init__(self, data, label, missing, weight, silent, feature_names, feature_types, nthread)
    378         data, feature_names, feature_types = _maybe_pandas_data(data,
    379                                                                 feature_names,
--> 380                                                                 feature_types)
    381
    382         data, feature_names, feature_types = _maybe_dt_data(data,

/usr/local/lib/python3.7/dist-packages/xgboost/core.py in _maybe_pandas_data(data, feature_names, feature_types)
    237         msg = """DataFrame.dtypes for data must be int, float or bool.
    238                 Did not expect the data types in fields """
--> 239         raise ValueError(msg + ', '.join(bad_fields))
    240
    241     if feature_names is None:

ValueError: DataFrame.dtypes for data must be int, float or bool.
                Did not expect the data types in fields date, auteur, avis, assureur, produit
```

Recherche en vue de la création d'une autre pipline de preposessing pour la variable avis.

In [2]:

```
!pip install --upgrade spacy
!python -m spacy download fr_core_news_sm
```

```
Requirement already satisfied: spacy in /usr/local/lib/python3.7/dist-packages (2.2.4)
Collecting spacy
  Downloading spacy-3.2.1-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (6.0 MB)
     |████████████████████████████████| 6.0 MB 5.1 MB/s
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (from spacy) (2.11.3)
Collecting srsly<3.0.0,>=2.4.1
  Downloading srsly-2.4.2-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (451 kB)
     |████████████████████████████████| 451 kB 66.9 MB/s
Collecting pydantic!=1.8,!=1.8.1,<1.9.0,>=1.7.4
  Downloading pydantic-1.8.2-cp37-cp37m-manylinux2014_x86_64.whl (10.1 MB)
     |████████████████████████████████| 10.1 MB 39.3 MB/s
Collecting typer<0.5.0,>=0.3.0
  Downloading typer-0.4.0-py3-none-any.whl (27 kB)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.7/di
```

```
st-packages (from spacy) (1.0.6)
Collecting thinc<8.1.0,>=8.0.12
  Downloading thinc-8.0.13-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (6
28 kB)
     |████████████████████████████████| 628 kB 55.0 MB/s
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.7/dist-pac
kages (from spacy) (2.0.6)
Collecting spacy-loggers<2.0.0,>=1.0.0
  Downloading spacy_loggers-1.0.1-py3-none-any.whl (7.0 kB)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in /usr/local/lib/python3.7/dist-pack
ages (from spacy) (0.4.1)
Requirement already satisfied: wasabi<1.1.0,>=0.8.1 in /usr/local/lib/python3.7/dist-pa
ckages (from spacy) (0.9.0)
Collecting catalogue<2.1.0,>=2.0.6
  Downloading catalogue-2.0.6-py3-none-any.whl (17 kB)
Collecting pathy>=0.3.5
  Downloading pathy-0.6.1-py3-none-any.whl (42 kB)
     |████████████████████████████████| 42 kB 1.0 MB/s
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-package
s (from spacy) (21.3)
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.7/dist-packages
(from spacy) (1.19.5)
Collecting spacy-legacy<3.1.0,>=3.0.8
  Downloading spacy_legacy-3.0.8-py2.py3-none-any.whl (14 kB)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.7/dist-pac
kages (from spacy) (4.62.3)
Collecting langcodes<4.0.0,>=3.2.0
  Downloading langcodes-3.3.0-py3-none-any.whl (181 kB)
     |████████████████████████████████| 181 kB 34.5 MB/s
Requirement already satisfied: typing-extensions<4.0.0.0,>=3.7.4 in /usr/local/lib/pyth
on3.7/dist-packages (from spacy) (3.10.0.2)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.7/dist-p
ackages (from spacy) (3.0.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (fr
om spacy) (57.4.0)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.7/dist
-packages (from spacy) (2.23.0)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (fro
m catalogue<2.1.0,>=2.0.6->spacy) (3.7.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dis
t-packages (from packaging>=20.0->spacy) (3.0.6)
Requirement already satisfied: smart-open<6.0.0,>=5.0.0 in /usr/local/lib/python3.7/dis
t-packages (from pathy>=0.3.5->spacy) (5.2.1)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pack
ages (from requests<3.0.0,>=2.13.0->spacy) (2021.10.8)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packa
ges (from requests<3.0.0,>=2.13.0->spacy) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/li
b/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
(from requests<3.0.0,>=2.13.0->spacy) (2.10)
Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.7/dist-pac
kages (from typer<0.5.0,>=0.3.0->spacy) (7.1.2)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packag
es (from jinja2->spacy) (2.0.1)
Installing collected packages: catalogue, typer, srsly, pydantic, thinc, spacy-loggers,
spacy-legacy, pathy, langcodes, spacy
  Attempting uninstall: catalogue
    Found existing installation: catalogue 1.0.0
    Uninstalling catalogue-1.0.0:
      Successfully uninstalled catalogue-1.0.0
  Attempting uninstall: srsly
    Found existing installation: srsly 1.0.5
    Uninstalling srsly-1.0.5:
      Successfully uninstalled srsly-1.0.5
  Attempting uninstall: thinc
    Found existing installation: thinc 7.4.0
    Uninstalling thinc-7.4.0:
      Successfully uninstalled thinc-7.4.0
  Attempting uninstall: spacy
    Found existing installation: spacy 2.2.4
    Uninstalling spacy-2.2.4:
      Successfully uninstalled spacy-2.2.4
Successfully installed catalogue-2.0.6 langcodes-3.3.0 pathy-0.6.1 pydantic-1.8.2 spacy
```

```
-3.2.1 spacy-legacy-3.0.8 spacy-loggers-1.0.1 srsly-2.4.2 thinc-8.0.13 typer-0.4.0
Collecting fr-core-news-sm==3.2.0
  Downloading https://github.com/explosion/spacy-models/releases/download/fr_core_news_sm-3.2.0/fr_core_news_sm-3.2.0-py3-none-any.whl (17.4 MB)
     |████████████████████████████████| 17.4 MB 464 kB/s
Requirement already satisfied: spacy<3.3.0,>=3.2.0 in /usr/local/lib/python3.7/dist-packages (from fr-core-news-sm==3.2.0) (3.2.1)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (2.23.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (21.3)
Requirement already satisfied: typing-extensions<4.0.0.0,>=3.7.4 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (3.10.0.2)
Requirement already satisfied: wasabi<1.1.0,>=0.8.1 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (0.9.0)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (4.62.3)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (2.0.6)
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (1.19.5)
Requirement already satisfied: pathy>=0.3.5 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (0.6.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (57.4.0)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (2.0.6)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (3.3.0)
Requirement already satisfied: srsly<3.0.0,>=2.4.1 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (2.4.2)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (0.4.1)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (1.0.1)
Requirement already satisfied: typer<0.5.0,>=0.3.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (0.4.0)
Requirement already satisfied: thinc<8.1.0,>=8.0.12 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (8.0.13)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (1.0.6)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (3.0.6)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (2.11.3)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<1.9.0,>=1.7.4 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (1.8.2)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.8 in /usr/local/lib/python3.7/dist-packages (from spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (3.0.8)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from catalogue<2.1.0,>=2.0.6->spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (3.7.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging>=20.0->spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (3.0.6)
Requirement already satisfied: smart-open<6.0.0,>=5.0.0 in /usr/local/lib/python3.7/dist-packages (from pathy>=0.3.5->spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (5.2.1)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (2021.10.8)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (1.24.3)
Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.7/dist-packages (from typer<0.5.0,>=0.3.0->spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (7.1.2)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2->spacy<3.3.0,>=3.2.0->fr-core-news-sm==3.2.0) (2.0.1)
Installing collected packages: fr-core-news-sm
Successfully installed fr-core-news-sm-3.2.0
✔ Download and installation successful
You can now load the package via spacy.load('fr_core_news_sm')
```

In [3]:
```
import spacy
# Load Our Model & NLP(TALN) Object
nlp = spacy.load('fr_core_news_sm')
```

In [7]:
```
df = pd.read_csv('avisassurance_train.csv', sep=";")
```

In [8]:
```
data = df.copy()
data = data.dropna(axis=0)
data
```

Out[8]:

	date	note	auteur	avis	assureur	produit
0	06 septem...	5	brahim--k-131532	Meilleurs assurances, prix, solutions, écoute,...	Direct Assurance	auto
1	03 mai 20...	4	bernard-g-112497	je suis globalement satisfait , sauf que vous ...	Direct Assurance	auto
2	21 mars 2...	5	virginie-t-107352	Prix tres abordable plusieurs options s'offren...	Direct Assurance	auto
3	10 juin 2...	4	boulain-f-116580	je satisfait du service, une réponse très rapi...	L'olivier Assurance	auto
4	29 janvie...	1	ouaille31-51798	Client depuis plus de 25 ans, très déçu de cet...	Matmut	auto
...	...	...	...	...	...	...
24100	22 mars 2...	1	hophop-107522	Assurance moto chez la mutuel des motards en F...	Mutuelle des Motards	moto
24101	06 décemb...	1	tzl-81680	Même les demandes les plus simples n'aboutisse...	Allianz	habitation
24102	14 avril ...	1	jmr-72500-110395	En décembre 2019, j'ai souscrit à un contrat C...	Cegema Assurances	sante
24103	11 juille...	3	cris-77532	Je suis assurer à la gmf depuis plus de 15 ans...	GMF	auto
24104	19 janvie...	1	jesse-51459	Bonjour\r\nMon ami vient de se faire voler sa ...	AMV	moto

24103 rows × 6 columns

In [9]:
```
data['avis'][1]
```

Out[9]:
```
"je suis globalement satisfait , sauf que vous avez un problème avec votre site interne
t ,impossible de déclarer un sinistre en ligne après plusieurs tentatives déclaration f
aite par téléphone ou tout c'est très bien passé , interlocutrice compétente et très ag
réable "
```

In [10]:
```
doc = nlp(data['avis'][1])
type(doc)
```

Out[10]:
```
spacy.tokens.doc.Doc
```

In [11]:
```
for token in doc:
    if token.pos_ == 'ADJ':
        print(token)
```

```
globalement
satisfait
impossible
```

```
compétente
agréable
```

In [122…

```
from collections import Counter

def mots_les_plus_frequents_par_note(data, note):

  words = []
  avis = data.loc[lambda x: x['note']==note, 'avis']

  for i in range(len(avis)):
    doc = nlp(avis.iloc[i])
    # tous les tokens qui ne sont pas des stop words ou des mots de ponctuation
    words = words + [token.lemma_
          for token in doc
          if not token.is_stop and not token.is_punct]

  # Mots les plus fréquents
  word_freq = Counter(words)
  common_words = word_freq.most_common(20)
  return common_words

def adjectifs_les_plus_frequents_par_note(data, note):

  adjectifs = []
  avis = data.loc[lambda x: x['note']==note, 'avis']

  for i in range(len(avis)):

    doc = nlp(avis.iloc[i])
    # tous les tokens qui ne sont pas des stop words ou des mots de ponctuation
    adjectifs = adjectifs + [token.lemma_
          for token in doc
          if token.pos_ == "ADJ" and not token.is_stop and not token.is_punct]

  # Mots les plus fréquents
  word_freq = Counter(adjectifs)
  common_words = word_freq.most_common(20)
  return common_words
```

In [123…

```
mots_les_plus_frequents_par_note(data, 5)
```

Out[123…

```
[('\r\n', 3843),
 ('assurance', 2201),
 ('prix', 2125),
 ('service', 2057),
 ('satisfait', 1725),
 ('rapide', 1154),
 ('bon', 1116),
 (' ', 1063),
 ('bien', 1042),
 ('recommande', 974),
 ('tarif', 751),
 ('simple', 739),
 ('conseiller', 698),
 ('être', 672),
 ('contrat', 589),
 ('client', 550),
 ('satisfaire', 513),
 ('efficace', 466),
 ('écoute', 456),
 ('assurer', 446)]
```

In [124…

```
mots_les_plus_frequents_par_note(data, 4)
```

Out[124…

```
[('\r\n', 3784),
 ('prix', 1896),
 ('service', 1894),
```

```
 ('assurance', 1796),
 ('satisfait', 1488),
 ('bon', 1175),
 (' ', 1061),
 ('bien', 1055),
 ('rapide', 984),
 ('être', 754),
 ('contrat', 683),
 ('simple', 656),
 ('tarif', 650),
 ('conseiller', 607),
 ('client', 596),
 ('satisfaire', 567),
 ('recommande', 526),
 ('voir', 506),
 ('site', 475),
 ('efficace', 427)]
```

In [125…
```
mots_les_plus_frequents_par_note(data, 3)
```

Out[125…
```
[('\r\n', 2765),
 ('assurance', 1465),
 (' ', 1324),
 ('service', 990),
 ('prix', 965),
 ('contrat', 806),
 ('bien', 673),
 ('être', 644),
 ('faire', 574),
 ('satisfait', 562),
 ('client', 540),
 ('an', 516),
 ('sinistre', 484),
 ('bon', 476),
 ('véhicule', 452),
 ('mois', 415),
 ('tarif', 401),
 ('voir', 401),
 ('assurer', 397),
 ('cher', 379)]
```

In [126…
```
mots_les_plus_frequents_par_note(data, 2)
```

Out[126…
```
[('\r\n', 4274),
 ('assurance', 2239),
 (' ', 1826),
 ('être', 1292),
 ('contrat', 1290),
 ('sinistre', 1179),
 ('mois', 1114),
 ('faire', 1083),
 ('aucun', 1070),
 ('service', 1022),
 ('an', 1006),
 ('véhicule', 959),
 ('client', 903),
 ('bien', 808),
 ('assurer', 799),
 ('\r\n\r\n', 765),
 ('jour', 739),
 ('euro', 732),
 ('payer', 727),
 ('prendre', 717)]
```

In [127…
```
mots_les_plus_frequents_par_note(data, 1)
```

Out[127…
```
[('\r\n', 9168),
 ('assurance', 4663),
 (' ', 3961),
 ('contrat', 2992),
```

```
 ('mois', 2762),
 ('être', 2600),
 ('aucun', 2553),
 ('faire', 2271),
 ('an', 2181),
 ('service', 1973),
 ('sinistre', 1945),
 ('client', 1926),
 ('euro', 1842),
 ('payer', 1753),
 ('bien', 1715),
 ('rien', 1561),
 ('jour', 1522),
 ('dossier', 1522),
 ('prendre', 1510),
 ('\r\n\r\n', 1505)]
```

On remarque que les mots les plus fréquent pour les 2 meilleurs notes sont quasiments les mêmes.

In [128…
```
adjectifs_les_plus_frequents_par_note(data, 5)
```

Out[128…
```
[('satisfait', 1701),
 ('rapide', 1120),
 ('bon', 1082),
 ('simple', 601),
 ('efficace', 441),
 ('clair', 378),
 ('cher', 289),
 ('téléphonique', 277),
 ('agréable', 241),
 ('attractif', 236),
 ('direct', 234),
 ('intéressant', 228),
 ('compétitif', 213),
 ('professionnel', 203),
 ('correct', 191),
 ('réactif', 190),
 ('disponible', 145),
 ('aimable', 131),
 ('jeune', 131),
 ('meilleur', 125)]
```

In [ ]:
```
adjectifs_les_plus_frequents_par_note(data, 1)
```