NoteBook_Courant_du_Futur_Final

January 9, 2021

Visualisation de données : Telecom Churn Prediction

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
import plotly.graph_objs as go
from scipy import stats
from plotly.offline import iplot, init_notebook_mode
```

```
In C:\Users\gth\Anaconda3\lib\site-packages\matplotlib\mpl-
data\stylelib\_classic_test.mplstyle:
The text.latex.preview rcparam was deprecated in Matplotlib 3.3 and will be
removed two minor releases later.
In C:\Users\gth\Anaconda3\lib\site-packages\matplotlib\mpl-
data\stylelib\_classic_test.mplstyle:
The mathtext.fallback_to_cm rcparam was deprecated in Matplotlib 3.3 and will be
removed two minor releases later.
In C:\Users\gth\Anaconda3\lib\site-packages\matplotlib\mpl-
data\stylelib\_classic_test.mplstyle: Support for setting the
'mathtext.fallback_to_cm' rcParam is deprecated since 3.3 and will be removed
two minor releases later; use 'mathtext.fallback : 'cm' instead.
In C:\Users\gth\Anaconda3\lib\site-packages\matplotlib\mpl-
data\stylelib\_classic_test.mplstyle:
The validate_bool_maybe_none function was deprecated in Matplotlib 3.3 and will
be removed two minor releases later.
In C:\Users\gth\Anaconda3\lib\site-packages\matplotlib\mpl-
data\stylelib\_classic_test.mplstyle:
The savefig.jpeg_quality rcparam was deprecated in Matplotlib 3.3 and will be
removed two minor releases later.
In C:\Users\gth\Anaconda3\lib\site-packages\matplotlib\mpl-
data\stylelib\_classic_test.mplstyle:
The keymap.all_axes rcparam was deprecated in Matplotlib 3.3 and will be removed
two minor releases later.
In C:\Users\gth\Anaconda3\lib\site-packages\matplotlib\mpl-
data\stylelib\_classic_test.mplstyle:
The animation.avconv_path rcparam was deprecated in Matplotlib 3.3 and will be
removed two minor releases later.
```

```
In C:\Users\gth\Anaconda3\lib\site-packages\matplotlib\mpl-
data\stylelib\_classic_test.mplstyle:
The animation.avconv_args rcparam was deprecated in Matplotlib 3.3 and will be
removed two minor releases later.
```

[2]:
```
sns.set(style='white')
pd.set_option('display.max_columns',None)
```

[5]:
```
df=pd.read_excel("Telco_customer_churn.csv.xlsx")
```

[6]:
```
# Une fonction qui nous permet d'avoir un resumé rapide du dataset
def resumetable(df):
    print(f"Dataset Shape: {df.shape}")
    summary = pd.DataFrame(df.dtypes,columns=['dtypes'])
    summary = summary.reset_index()
    summary['Nom'] = summary['index']
    summary = summary[['Nom','dtypes']]
    summary['Manquants'] = df.isnull().sum().values
    summary['Uniques'] = df.nunique().values
    summary['Premiere Valeur'] = df.loc[0].values
    summary['Deuxieme Valeur'] = df.loc[2].values
    summary['Avant derniere Valeur'] = df.iloc[-2].values
    summary['Derniere Valeur'] = df.iloc[-1].values

    for name in summary['Nom'].value_counts().index:
        summary.loc[summary['Nom'] == name, 'Entropy'] = round(stats.
↪entropy(df[name].value_counts(normalize=True), base=10),2)
    return summary
```

[7]:
```
resumetable(df)
```

```
Dataset Shape: (7043, 33)
```

[7]:
```
                 Nom   dtypes  Manquants  Uniques  \
0         CustomerID   object          0     7043
1              Count    int64          0        1
2            Country   object          0        1
3              State   object          0        1
4               City   object          0     1129
5           Zip Code    int64          0     1652
6           Lat Long   object          0     1652
7           Latitude  float64          0     1652
8          Longitude  float64          0     1651
9             Gender   object          0        2
10    Senior Citizen   object          0        2
11           Partner   object          0        2
12        Dependents   object          0        2
13     Tenure Months    int64          0       73
```

```
14      Phone Service   object        0       2
15     Multiple Lines   object        0       3
16   Internet Service   object        0       3
17    Online Security   object        0       3
18      Online Backup   object        0       3
19  Device Protection   object        0       3
20       Tech Support   object        0       3
21       Streaming TV   object        0       3
22   Streaming Movies   object        0       3
23           Contract   object        0       3
24  Paperless Billing   object        0       2
25     Payment Method   object        0       4
26    Monthly Charges  float64        0    1585
27      Total Charges   object        0    6531
28        Churn Label   object        0       2
29        Churn Value    int64        0       2
30        Churn Score    int64        0      85
31               CLTV    int64        0    3438
32       Churn Reason   object     5174      20

               Premiere Valeur         Deuxieme Valeur  \
0                   3668-QPYBK              9305-CDSKC
1                            1                       1
2                United States           United States
3                   California              California
4                  Los Angeles             Los Angeles
5                        90003                   90006
6        33.964131, -118.272783  34.048013, -118.293953
7                      33.9641                  34.048
8                     -118.273                -118.294
9                         Male                  Female
10                          No                      No
11                          No                      No
12                          No                     Yes
13                           2                       8
14                         Yes                     Yes
15                          No                     Yes
16                         DSL             Fiber optic
17                         Yes                      No
18                         Yes                      No
19                          No                     Yes
20                          No                      No
21                          No                     Yes
22                          No                     Yes
23              Month-to-month          Month-to-month
24                         Yes                     Yes
25                Mailed check        Electronic check
```

```
26                        53.85                     99.65
27                       108.15                     820.5
28                          Yes                       Yes
29                            1                         1
30                           86                        86
31                         3239                      5372
32  Competitor made better offer                     Moved

   Avant derniere Valeur             Derniere Valeur  Entropy
0             4801-JZAZL                  3186-AJIEK     3.85
1                      1                           1     0.00
2          United States               United States     0.00
3             California                  California     0.00
4           Angelus Oaks                Apple Valley     2.86
5                  92305                       92308     3.22
6    34.1678, -116.86433     34.424926, -117.184503     3.22
7                34.1678                     34.4249     3.22
8               -116.864                    -117.185     3.22
9                 Female                        Male     0.30
10                    No                          No     0.19
11                   Yes                          No     0.30
12                   Yes                          No     0.23
13                    11                          66     1.78
14                    No                         Yes     0.14
15      No phone service                          No     0.41
16                   DSL                 Fiber optic     0.46
17                   Yes                         Yes     0.45
18                    No                          No     0.46
19                    No                         Yes     0.46
20                    No                         Yes     0.45
21                    No                         Yes     0.46
22                    No                         Yes     0.46
23        Month-to-month                    Two year     0.43
24                   Yes                         Yes     0.29
25      Electronic check   Bank transfer (automatic)     0.59
26                  29.6                      105.65     3.02
27                346.45                      6844.5     3.80
28                    No                          No     0.25
29                     0                           0     0.25
30                    59                          38     1.89
31                  2793                        5097     3.47
32                   NaN                         NaN     1.22
```

[8]:

```
df.describe()
```

[8]:

```
        Count     Zip Code     Latitude    Longitude  Tenure Months  \
count  7043.0  7043.000000  7043.000000  7043.000000    7043.000000
```

```
mean      1.0  93521.964646    36.282441  -119.798880      32.371149
std       0.0   1865.794555     2.455723     2.157889      24.559481
min       1.0  90001.000000    32.555828  -124.301372       0.000000
25%       1.0  92102.000000    34.030915  -121.815412       9.000000
50%       1.0  93552.000000    36.391777  -119.730885      29.000000
75%       1.0  95351.000000    38.224869  -118.043237      55.000000
max       1.0  96161.000000    41.962127  -114.192901      72.000000

       Monthly Charges  Churn Value  Churn Score         CLTV
count      7043.000000  7043.000000  7043.000000  7043.000000
mean         64.761692     0.265370    58.699418  4400.295755
std          30.090047     0.441561    21.525131  1183.057152
min          18.250000     0.000000     5.000000  2003.000000
25%          35.500000     0.000000    40.000000  3469.000000
50%          70.350000     0.000000    61.000000  4527.000000
75%          89.850000     1.000000    75.000000  5380.500000
max         118.750000     1.000000   100.000000  6500.000000
```

[9]:
```python
print(df.columns.values)
```

```
['CustomerID' 'Count' 'Country' 'State' 'City' 'Zip Code' 'Lat Long'
 'Latitude' 'Longitude' 'Gender' 'Senior Citizen' 'Partner' 'Dependents'
 'Tenure Months' 'Phone Service' 'Multiple Lines' 'Internet Service'
 'Online Security' 'Online Backup' 'Device Protection' 'Tech Support'
 'Streaming TV' 'Streaming Movies' 'Contract' 'Paperless Billing'
 'Payment Method' 'Monthly Charges' 'Total Charges' 'Churn Label'
 'Churn Value' 'Churn Score' 'CLTV' 'Churn Reason']
```

On constate que le “Total charges” est un string, donc on va le convertir en int

[10]:
```python
# Le total charges est un string
# donc on va le convertir en un type numeric
df['Total Charges'] = pd.to_numeric(df['Total Charges'],errors='coerce')
```

On visualise le nombre de valeur null dans les champs

[11]:
```python
df.isnull().sum()
```

[11]:
```
CustomerID           0
Count                0
Country              0
State                0
City                 0
Zip Code             0
Lat Long             0
Latitude             0
Longitude            0
Gender               0
Senior Citizen       0
```

```
Partner                 0
Dependents              0
Tenure Months           0
Phone Service           0
Multiple Lines          0
Internet Service        0
Online Security         0
Online Backup           0
Device Protection       0
Tech Support            0
Streaming TV            0
Streaming Movies        0
Contract                0
Paperless Billing       0
Payment Method          0
Monthly Charges         0
Total Charges          11
Churn Label             0
Churn Value             0
Churn Score             0
CLTV                    0
Churn Reason         5174
dtype: int64
```

On peut constater :

qu'il ya 5174 personnes qui y sont encore

qu'il ya 11 'Total charges' non calculés

Le nombre de lignes contenant un 'Total charges' n'est pas considérables vu la taille des données donc on peut les supprimer :

[12]:
```
#df=df[pd.notnull(df['Total Charges'])]
df=df[df['Total Charges'].notna()]
```

[13]:
```
df.isnull().sum()
```

[13]:
```
CustomerID              0
Count                   0
Country                 0
State                   0
City                    0
Zip Code                0
Lat Long                0
Latitude                0
Longitude               0
Gender                  0
Senior Citizen          0
```

```
Partner                0
Dependents             0
Tenure Months          0
Phone Service          0
Multiple Lines         0
Internet Service       0
Online Security        0
Online Backup          0
Device Protection      0
Tech Support           0
Streaming TV           0
Streaming Movies       0
Contract               0
Paperless Billing      0
Payment Method         0
Monthly Charges        0
Total Charges          0
Churn Label            0
Churn Value            0
Churn Score            0
CLTV                   0
Churn Reason        5163
dtype: int64
```

[14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7032 entries, 0 to 7042
Data columns (total 33 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   CustomerID         7032 non-null   object
 1   Count              7032 non-null   int64
 2   Country            7032 non-null   object
 3   State              7032 non-null   object
 4   City               7032 non-null   object
 5   Zip Code           7032 non-null   int64
 6   Lat Long           7032 non-null   object
 7   Latitude           7032 non-null   float64
 8   Longitude          7032 non-null   float64
 9   Gender             7032 non-null   object
 10  Senior Citizen     7032 non-null   object
 11  Partner            7032 non-null   object
 12  Dependents         7032 non-null   object
 13  Tenure Months      7032 non-null   int64
 14  Phone Service      7032 non-null   object
 15  Multiple Lines     7032 non-null   object
 16  Internet Service   7032 non-null   object
```

```
 17  Online Security    7032 non-null   object
 18  Online Backup      7032 non-null   object
 19  Device Protection  7032 non-null   object
 20  Tech Support       7032 non-null   object
 21  Streaming TV       7032 non-null   object
 22  Streaming Movies   7032 non-null   object
 23  Contract           7032 non-null   object
 24  Paperless Billing  7032 non-null   object
 25  Payment Method     7032 non-null   object
 26  Monthly Charges    7032 non-null   float64
 27  Total Charges      7032 non-null   float64
 28  Churn Label        7032 non-null   object
 29  Churn Value        7032 non-null   int64
 30  Churn Score        7032 non-null   int64
 31  CLTV               7032 non-null   int64
 32  Churn Reason       1869 non-null   object
dtypes: float64(4), int64(6), object(23)
memory usage: 1.8+ MB
```

On peut aussi observer si certaines variables (catégoriques) auront un impact sur les resultats, en utilisant les variables fictives :

[15]:
```
# je retire l'ID Custumer qui n'a pas d'impact
df2= df.iloc[:,1:]
df2=df2.drop(['Zip Code','Lat Long','Latitude','Longitude','Churn␣
↪Label'],axis=1)
df3=df2
df_fic=pd.get_dummies(df2) # pour transformer toutes les variables categoriques␣
↪en num
df2
```

[15]:
```
      Count        Country       State          City  Gender Senior Citizen  \
0         1  United States  California   Los Angeles    Male             No
1         1  United States  California   Los Angeles  Female             No
2         1  United States  California   Los Angeles  Female             No
3         1  United States  California   Los Angeles  Female             No
4         1  United States  California   Los Angeles    Male             No
…       …              …           …             …     …              …
7038      1  United States  California       Landers  Female             No
7039      1  United States  California      Adelanto    Male             No
7040      1  United States  California         Amboy  Female             No
7041      1  United States  California  Angelus Oaks  Female             No
7042      1  United States  California  Apple Valley    Male             No

     Partner Dependents  Tenure Months Phone Service    Multiple Lines  \
0         No         No              2           Yes                No
1         No        Yes              2           Yes                No
2         No        Yes              8           Yes               Yes
```

```
3        Yes        Yes             28           Yes                 Yes
4         No        Yes             49           Yes                 Yes
…       …        …            …           …              …
7038      No         No             72           Yes                  No
7039     Yes        Yes             24           Yes                 Yes
7040     Yes        Yes             72           Yes                 Yes
7041     Yes        Yes             11            No    No phone service
7042      No         No             66           Yes                  No

     Internet Service      Online Security        Online Backup  \
0                 DSL                  Yes                  Yes
1         Fiber optic                   No                   No
2         Fiber optic                   No                   No
3         Fiber optic                   No                   No
4         Fiber optic                   No                  Yes
…              …                 …                  …
7038               No  No internet service  No internet service
7039              DSL                  Yes                   No
7040      Fiber optic                   No                  Yes
7041              DSL                  Yes                   No
7042      Fiber optic                  Yes                   No

        Device Protection         Tech Support         Streaming TV  \
0                      No                   No                   No
1                      No                   No                   No
2                     Yes                   No                  Yes
3                     Yes                  Yes                  Yes
4                     Yes                   No                  Yes
…                  …                  …                  …
7038  No internet service  No internet service  No internet service
7039                  Yes                  Yes                  Yes
7040                  Yes                   No                  Yes
7041                   No                   No                   No
7042                  Yes                  Yes                  Yes

         Streaming Movies        Contract Paperless Billing  \
0                      No  Month-to-month               Yes
1                      No  Month-to-month               Yes
2                     Yes  Month-to-month               Yes
3                     Yes  Month-to-month               Yes
4                     Yes  Month-to-month               Yes
…                  …             …                …
7038  No internet service        Two year               Yes
7039                  Yes        One year               Yes
7040                  Yes        One year               Yes
7041                   No  Month-to-month               Yes
7042                  Yes        Two year               Yes
```

```
                  Payment Method  Monthly Charges  Total Charges  Churn Value  \
0                   Mailed check            53.85         108.15            1
1               Electronic check            70.70         151.65            1
2               Electronic check            99.65         820.50            1
3               Electronic check           104.80        3046.05            1
4      Bank transfer (automatic)           103.70        5036.30            1
…                            …                …             …          …
7038   Bank transfer (automatic)            21.15        1419.40            0
7039                Mailed check            84.80        1990.50            0
7040     Credit card (automatic)           103.20        7362.90            0
7041            Electronic check            29.60         346.45            0
7042   Bank transfer (automatic)           105.65        6844.50            0

      Churn Score  CLTV                   Churn Reason
0              86  3239   Competitor made better offer
1              67  2701                          Moved
2              86  5372                          Moved
3              84  5003                          Moved
4              89  5340  Competitor had better devices
…             …   …                            …
7038           45  5306                            NaN
7039           59  2140                            NaN
7040           71  5560                            NaN
7041           59  2793                            NaN
7042           38  5097                            NaN

[7032 rows x 27 columns]
```

[16]:
```
df2['Senior Citizen'].replace(to_replace='Yes',value=1,inplace=True)
df2['Senior Citizen'].replace(to_replace='No',value=0,inplace=True)
df2['Dependents'].replace(to_replace='Yes',value=1,inplace=True)
df2['Dependents'].replace(to_replace='No',value=0,inplace=True)
df2['Multiple Lines'].replace(to_replace='Yes',value=1,inplace=True)
df2['Multiple Lines'].replace(to_replace='No',value=0,inplace=True)
df2.tail()
```

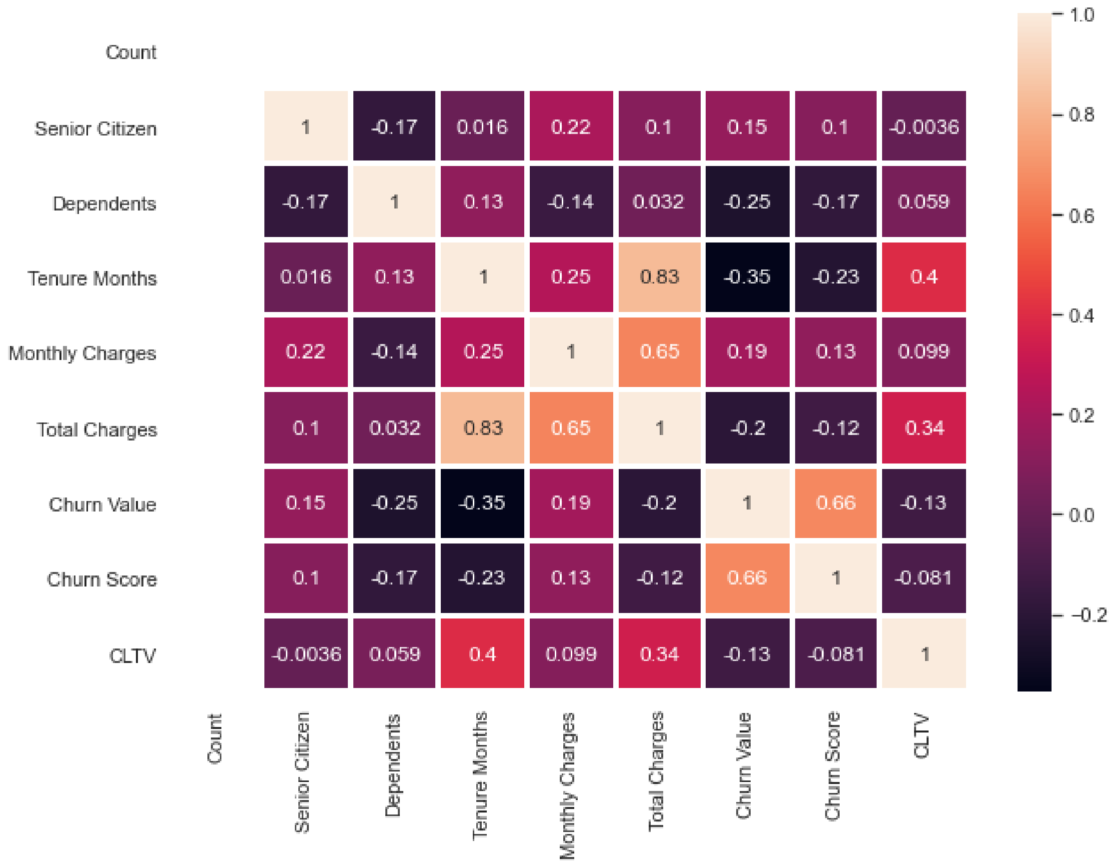
[16]:
```
      Count        Country       State          City  Gender  Senior Citizen  \
7038      1  United States  California       Landers  Female               0
7039      1  United States  California      Adelanto    Male               0
7040      1  United States  California         Amboy  Female               0
7041      1  United States  California  Angelus Oaks  Female               0
7042      1  United States  California  Apple Valley    Male               0

     Partner  Dependents  Tenure Months Phone Service   Multiple Lines  \
7038      No           0             72           Yes                0
7039     Yes           1             24           Yes                1
```

```
7040     Yes             1             72           Yes                 1
7041     Yes             1             11            No  No phone service
7042      No             0             66           Yes                 0

     Internet Service      Online Security        Online Backup  \
7038               No  No internet service  No internet service
7039              DSL                  Yes                   No
7040      Fiber optic                   No                  Yes
7041              DSL                  Yes                   No
7042      Fiber optic                  Yes                   No

        Device Protection         Tech Support         Streaming TV  \
7038  No internet service  No internet service  No internet service
7039                  Yes                  Yes                  Yes
7040                  Yes                   No                  Yes
7041                   No                   No                   No
7042                  Yes                  Yes                  Yes

         Streaming Movies        Contract Paperless Billing  \
7038  No internet service        Two year               Yes
7039                  Yes        One year               Yes
7040                  Yes        One year               Yes
7041                   No  Month-to-month               Yes
7042                  Yes        Two year               Yes

                Payment Method  Monthly Charges  Total Charges  Churn Value  \
7038  Bank transfer (automatic)           21.15        1419.40            0
7039               Mailed check           84.80        1990.50            0
7040    Credit card (automatic)          103.20        7362.90            0
7041           Electronic check           29.60         346.45            0
7042  Bank transfer (automatic)          105.65        6844.50            0

      Churn Score  CLTV Churn Reason
7038           45  5306          NaN
7039           59  2140          NaN
7040           71  5560          NaN
7041           59  2793          NaN
7042           38  5097          NaN
```

[17]:

```
corr_data=pd.DataFrame(df2)
mask=np.zeros_like(corr_data)
plt.figure(figsize=(10,7))
sns.heatmap(corr_data.corr(),annot=True,linewidths=2)
#df_fic.corr()['Churn Value'].sort_values(ascending = False).plot(kind='bar')
```

[17]: <AxesSubplot:>



Exploration Proprement Dite

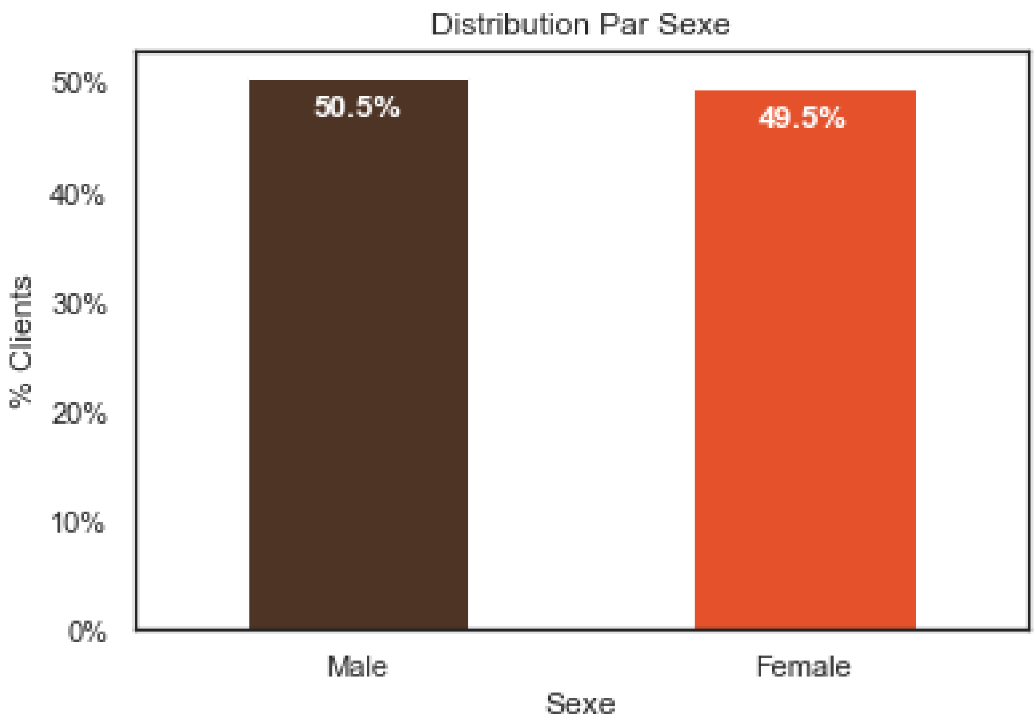
Afin de mieux comprendre la tendance de nos données, il sera efficace pour nous de les explorer en visualisant les différentes tendances des variables (de facon individuelles, en tranches puis en dès). Tout ceci pour se faire de probables hypothèses

I. Démographie

I.1 Le sexe

[18]:
```
colors=['#4D3425','#E4512B']
ax=(df2['Gender'].value_counts()*100.0/len(df2)).
 ↪plot(kind='bar',stacked=True,rot=0,color=colors)
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.set_ylabel("% Clients")
ax.set_xlabel("Sexe")
ax.set_ylabel("% Clients")
ax.set_title("Distribution Par Sexe")


#Collection des données dans une liste pour l'affichage
```

```
#dans les barres
totals=[]

for i in ax.patches:
    totals.append(i.get_width())

total = sum(totals)

for i in ax.patches:
    ax.text(i.get_x()+.15,i.get_height()-3.5,\
            str(round((i.get_height()/total),1))+'%',
            fontsize=12,
            color='white',
            weight = 'bold'
           )
```



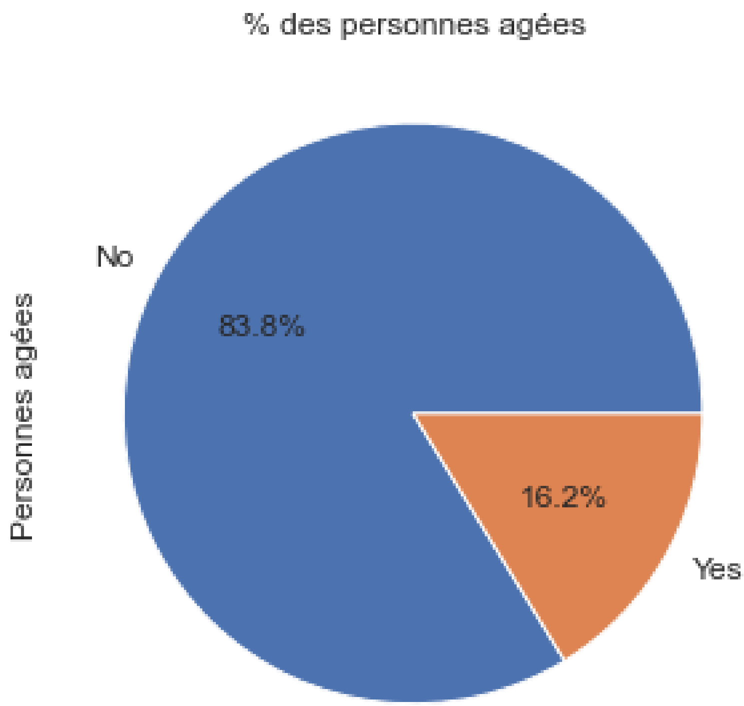
Ainsi on peut noter qu'environ la moitiée de notre DataSet est constituée d'hommes et l'autre moitiée de femmes

I.2 Les personnes agées

[19]:
```
ax=(df3['Senior Citizen'].value_counts()*100.0/len(df3))\
.plot.pie(autopct='%.1f%%',labels=['No','Yes'],figsize=(5,5),fontsize=12)
```

```
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.set_ylabel('Personnes agées',fontsize = 12)
ax.set_title('% des personnes agées', fontsize = 12)
```

[19]: Text(0.5, 1.0, '% des personnes agées')

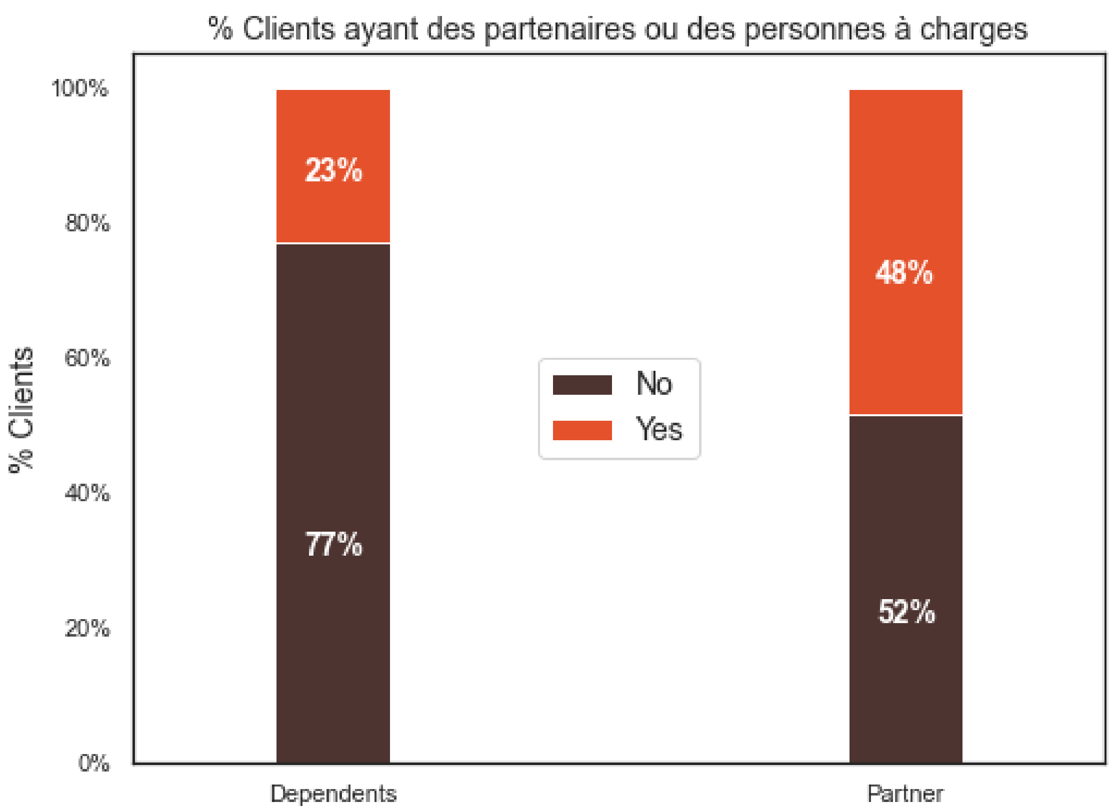


16,2% des clients est très agée, donc la plupart des personnes de notre dataset est jeune

I.3 Statut de partenaires et de personnes à charge

[20]:
```
df4=pd.melt(df,id_vars=['CustomerID'],value_vars=['Dependents','Partner'])
df5=df4.groupby(['variable','value']).count().unstack()
df5=df5*100/len(df)
colors = ['#4D3430','#E4512B']
ax=df5.loc[:,'CustomerID'].plot.
 ↪bar(stacked=True,color=colors,figsize=(8,6),rot=0,width=.2)

ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.set_ylabel('% Clients',size = 14)
ax.set_xlabel('')
ax.set_title('% Clients ayant des partenaires ou des personnes à charges',size␣
 ↪= 14)
```

```
ax.legend(loc = 'center',prop={'size':14})

for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.
↪4*height),
                color = 'white',
               weight = 'bold',
               size = 14)
```



De cette figure on peut conclure que:

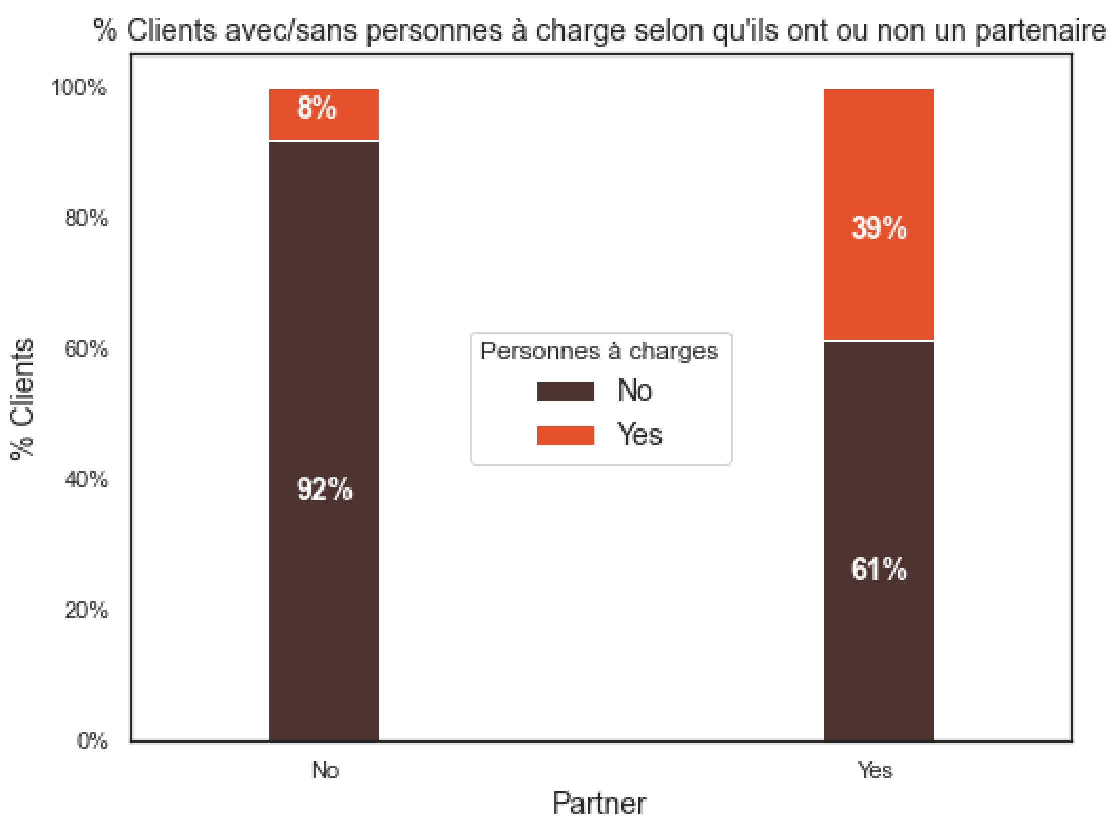
23% des clients ont des personnes à charges

48% des clients ont des partenaires

Il serait aussi interessant de visualiser les clients ayant à la fois des partenaires et des personnes à charge

[21]:
```
partner_dependents = df.groupby(['Partner','Dependents']).size().unstack()

colors = ['#4D3430','#E4512B']
```

```python
ax = (partner_dependents.T*100.0/partner_dependents.T.sum()).T.
 ↪plot(kind='bar',width=0.2,stacked=True,rot=0,figsize=(8,6),color=colors)

ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.legend(loc='center',prop={'size':14},title = 'Personnes à charges',fontsize␣
 ↪=14)
ax.set_ylabel('% Clients',size = 14)
ax.set_title("% Clients avec/sans personnes à charge selon qu'ils ont ou non un␣
 ↪partenaire",size = 14)
ax.xaxis.label.set_size(14)

for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.
 ↪4*height),
                color = 'white',
               weight = 'bold',
               size = 14)
```



On peut constater que :

La grande majorité (92%) des personnes n'ayant pas de partenaires n'ont pas de personnes à charge

40% des personnes ayant un partenaire a une personne à charge

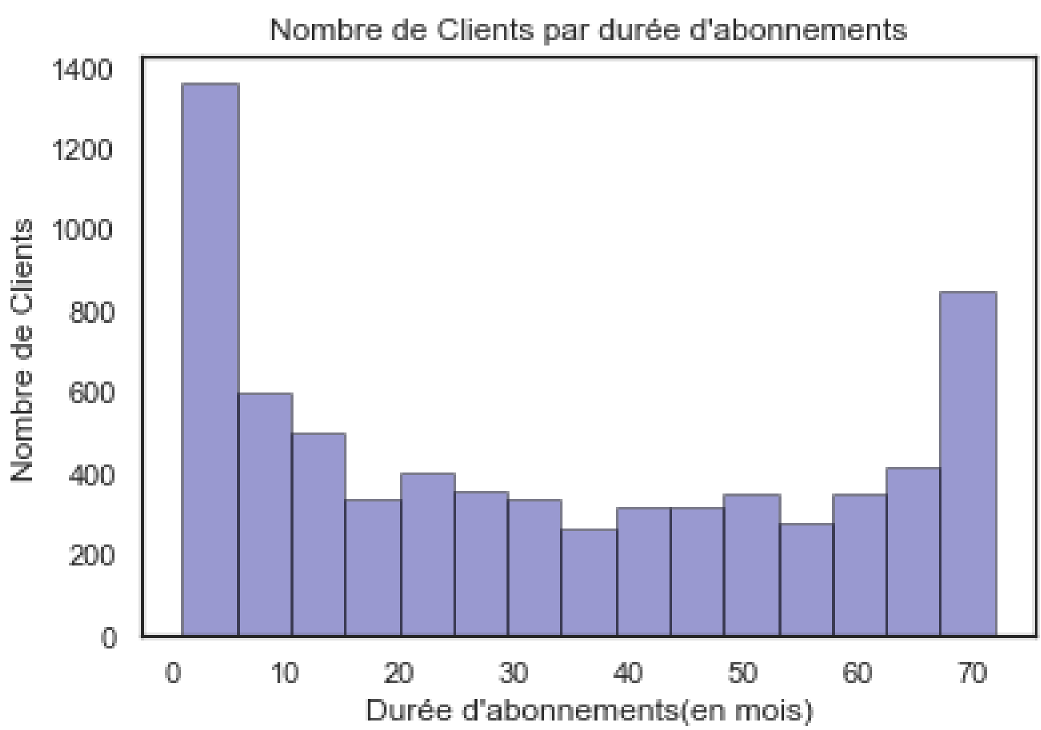
II. Information sur les comptes clients

II.1 Selon la durée d'abonnement

[22]:
```
ax= sns.distplot(df['Tenure Months'],hist=True,kde=False,color='darkblue',
                 hist_kws={'edgecolor':'black'},
                kde_kws={'linewidth':6})
ax.set_ylabel('Nombre de Clients')
ax.set_xlabel("Durée d'abonnements(en mois)")
ax.set_title("Nombre de Clients par durée d'abonnements")
```

```
C:\Users\gth\Anaconda3\lib\site-packages\seaborn\distributions.py:2551:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```

[22]: Text(0.5, 1.0, "Nombre de Clients par durée d'abonnements")



De cet histogramme nous pouvons constater:

Q'un grand nombre (>1200) de clients vient de s'abonner à la sociètè

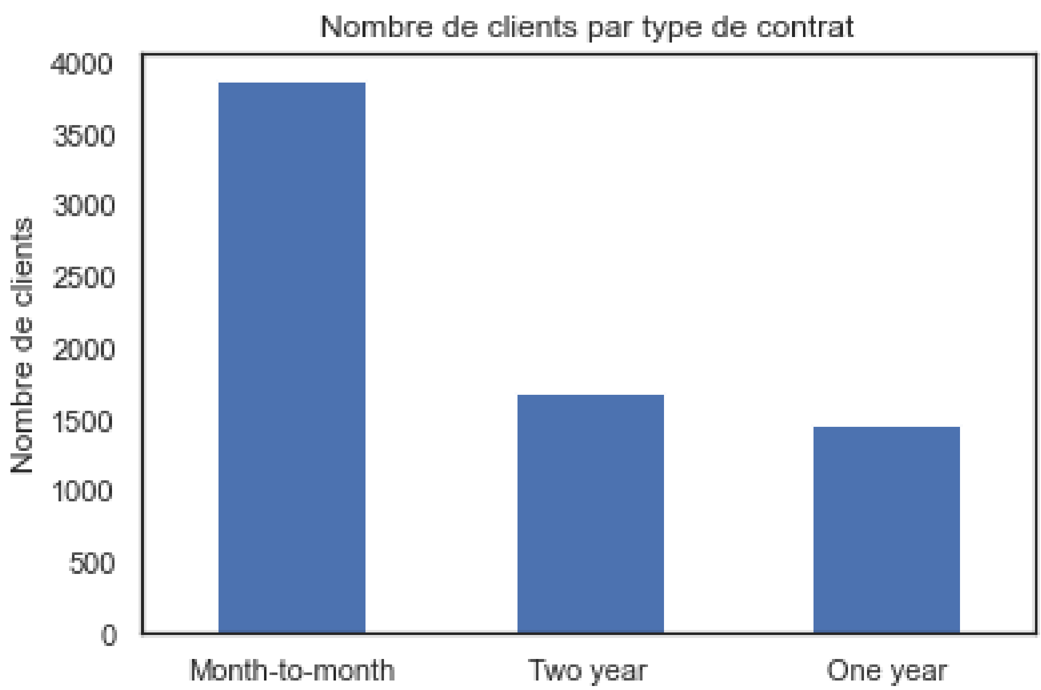
Q'un grand nombre (entre 800 et 100) de clients s'est abonné à la sociètè il ya environ 70 mois

Cela peut s'expliquer à cause du grand nombre de service. Et donc il ya des services interessants qui retiennent les clients et d'autres non

II.1 Selon le contrat

[23]:
```
ax = df['Contract'].value_counts().plot(kind = 'bar',rot = 0, width = 0.5)
ax.set_ylabel('Nombre de clients')
ax.set_title('Nombre de clients par type de contrat')
```

[23]:
```
Text(0.5, 1.0, 'Nombre de clients par type de contrat')
```



De cet histogramme on peut noter que

la grande majorité des clients a un contrat de paie mensuel

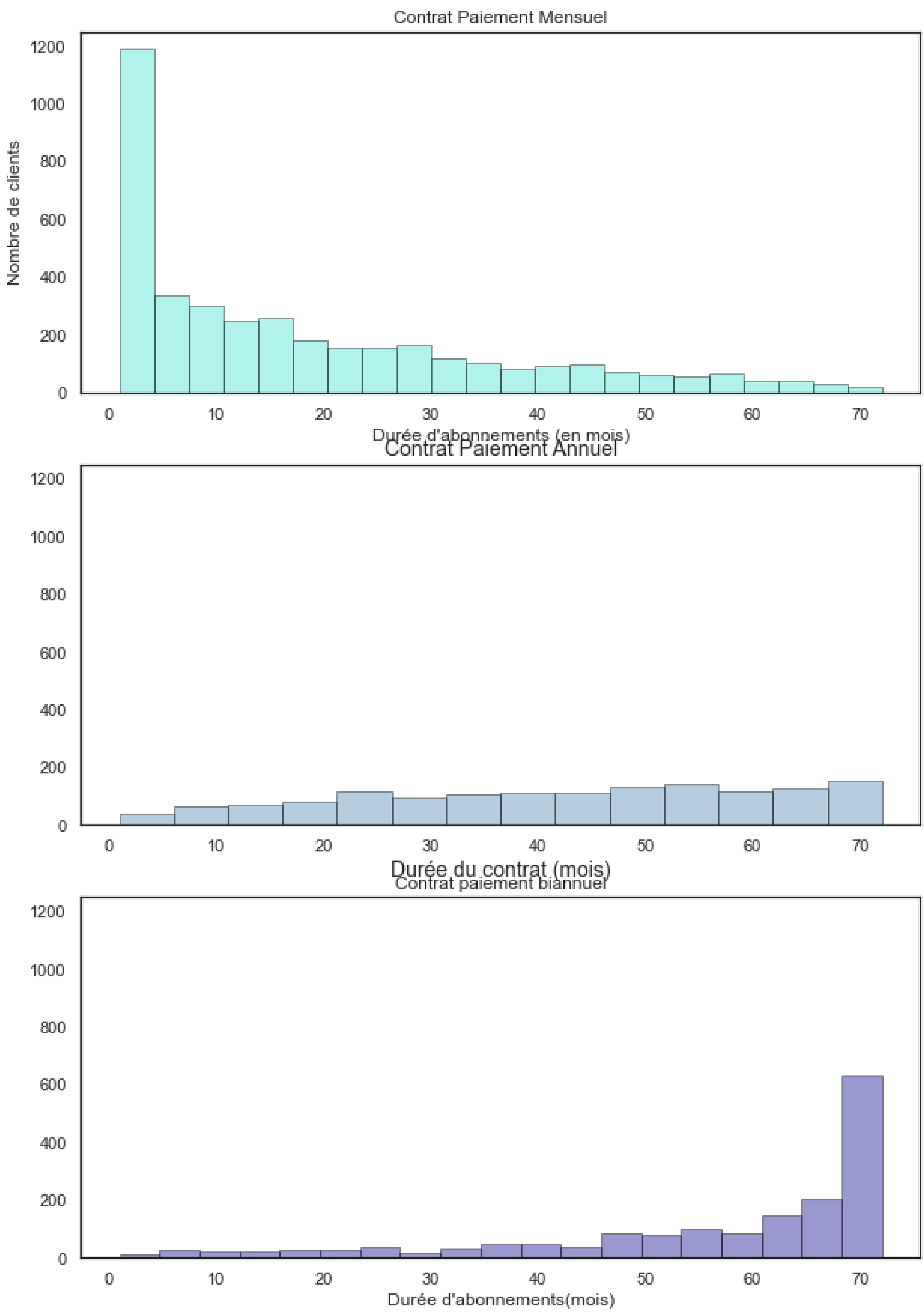
Il ya un nombre presque égale entre les abonnés à contrat annuel et biannuel

Il serait interessant de visualiser la durée du contrat par type d'abonnements

[24]:
```
fig, (ax1,ax2,ax3) = plt.subplots(nrows=3, ncols=1, sharey = True, figsize =␣
↪(10,15))

ax = sns.distplot(df[df['Contract']=='Month-to-month']['Tenure Months'],
```

```python
                   hist=True, kde=False,
                  color = 'turquoise',
                   hist_kws={'edgecolor':'black'},
                   kde_kws={'linewidth': 4},
                 ax=ax1)
ax.set_ylabel('Nombre de clients')
ax.set_xlabel("Durée d'abonnements (en mois)")
ax.set_title("Contrat Paiement Mensuel")

ax = sns.distplot(df[df['Contract']=='One year']['Tenure Months'],
                   hist=True, kde=False,
                  color = 'steelblue',
                   hist_kws={'edgecolor':'black'},
                   kde_kws={'linewidth': 4},
                 ax=ax2)
ax.set_xlabel('Durée du contrat (mois)',size = 14)
ax.set_title('Contrat Paiement Annuel',size = 14)

ax = sns.distplot(df[df['Contract']=='Two year']['Tenure Months'],
                   hist=True, kde=False,
                  color = 'darkblue',
                   hist_kws={'edgecolor':'black'},
                   kde_kws={'linewidth': 4},
                 ax=ax3)

ax.set_xlabel("Durée d'abonnements(mois)")
ax.set_title('Contrat paiement biannuel')
```

[24]: Text(0.5, 1.0, 'Contrat paiement biannuel')


Contrat Paiement Mensuel
Nombre de clients
Durée d'abonnements (en mois)
Contrat Paiement Annuel
Durée du contrat (mois)
Contrat paiement biannuel
Durée d'abonnements(mois)

De ces graphes nous constatons que :

Les contrats avec paiement Mensuel dure environ 2 mois

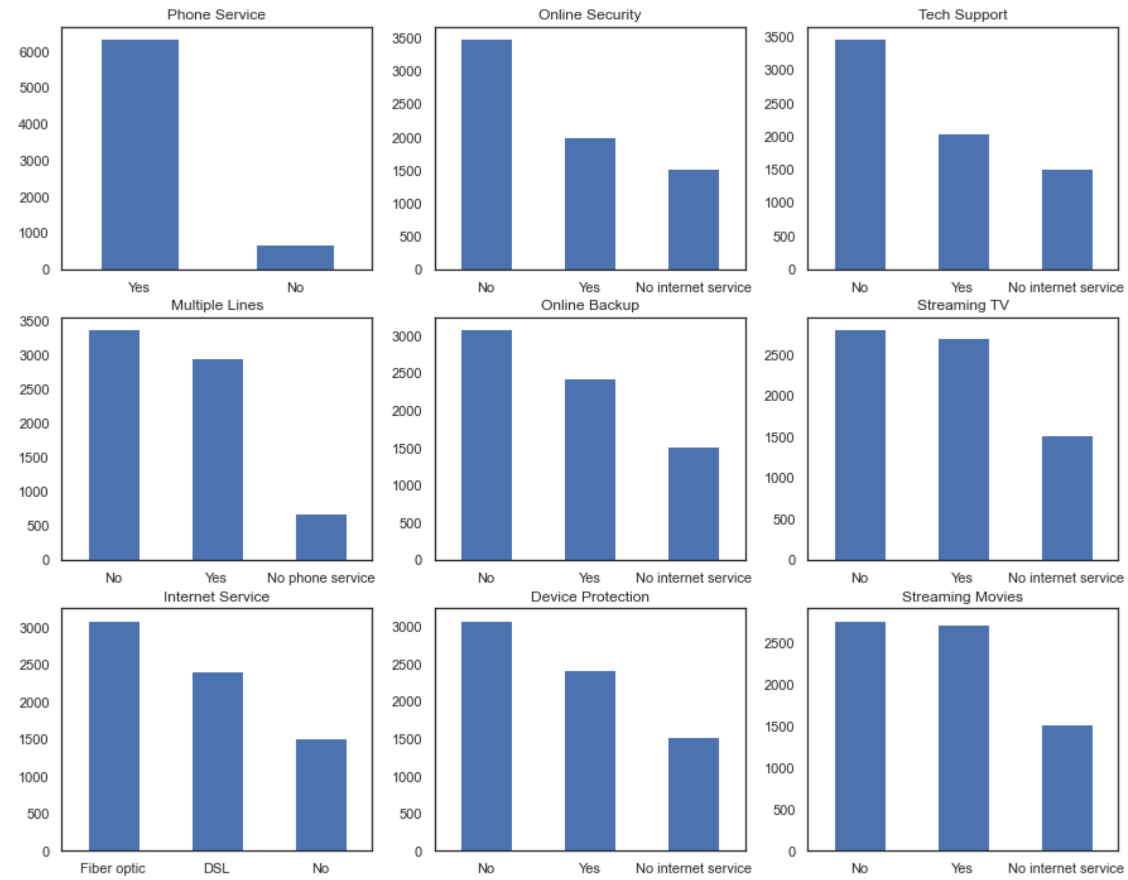
Ceux qui sont le plus fidèle sont les clients ayant souscrit à des services avec contrat à paiement Biannule avec une durée de plus de 70 mois

III. Information sur la repartition des services

[25]:
```
df.columns.values
```

[25]:
```
array(['CustomerID', 'Count', 'Country', 'State', 'City', 'Zip Code',
       'Lat Long', 'Latitude', 'Longitude', 'Gender', 'Senior Citizen',
       'Partner', 'Dependents', 'Tenure Months', 'Phone Service',
       'Multiple Lines', 'Internet Service', 'Online Security',
       'Online Backup', 'Device Protection', 'Tech Support',
       'Streaming TV', 'Streaming Movies', 'Contract',
       'Paperless Billing', 'Payment Method', 'Monthly Charges',
       'Total Charges', 'Churn Label', 'Churn Value', 'Churn Score',
       'CLTV', 'Churn Reason'], dtype=object)
```

[26]:
```
services = ['Phone Service',
       'Multiple Lines', 'Internet Service', 'Online Security',
       'Online Backup', 'Device Protection', 'Tech Support',
       'Streaming TV', 'Streaming Movies']
fig,axes = plt.subplots(nrows=3,ncols=3,figsize=(15,12))
for i,item in enumerate(services):
    if i<3:
        ax=df[item].value_counts().plot(kind='bar',ax=axes[i,0],rot=0)
    elif i>=3 and i <6:
        ax = df[item].value_counts().plot(kind = 'bar',ax=axes[i-3,1],rot = 0)
    elif i<9:
        ax = df[item].value_counts().plot(kind = 'bar',ax=axes[i-6,2],rot = 0)
    ax.set_title(item)
```



On peut ainsi faire une liste des services les plus utilisées :

Phones services

Streaming Service

Streaming Movies

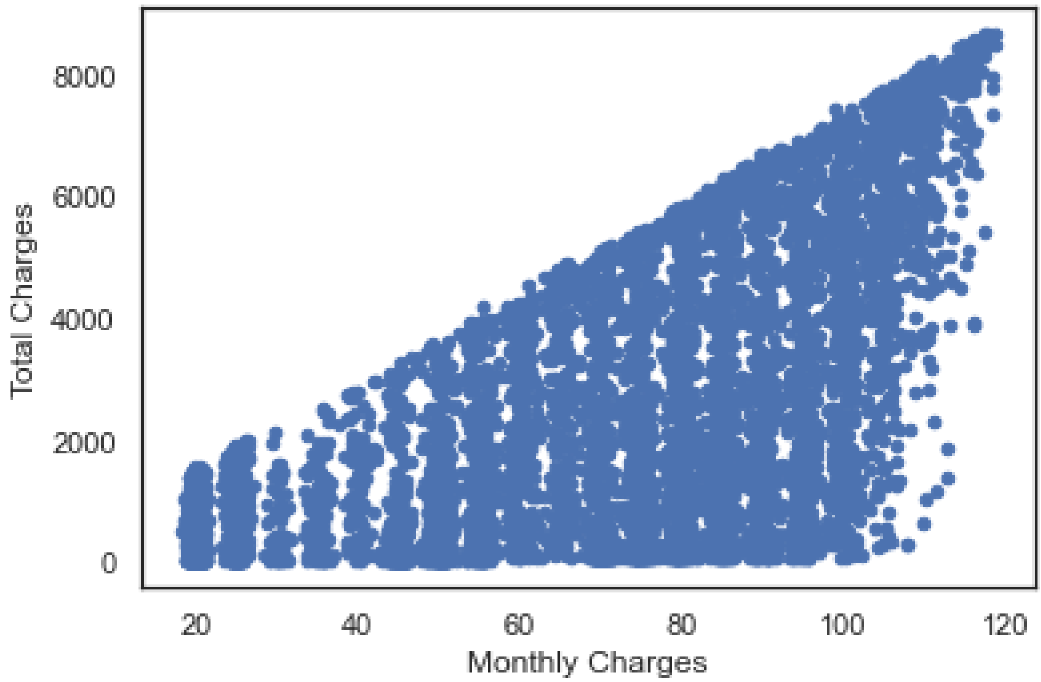
Internet (Fibre optique)

IV. Relation entre les frais mensuels et les frais totaux

[27]:
```
df[['Monthly Charges','Total Charges']].plot.scatter(x='Monthly Charges',y='Total Charges')
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
```

[27]:
```
<AxesSubplot:xlabel='Monthly Charges', ylabel='Total Charges'>
```



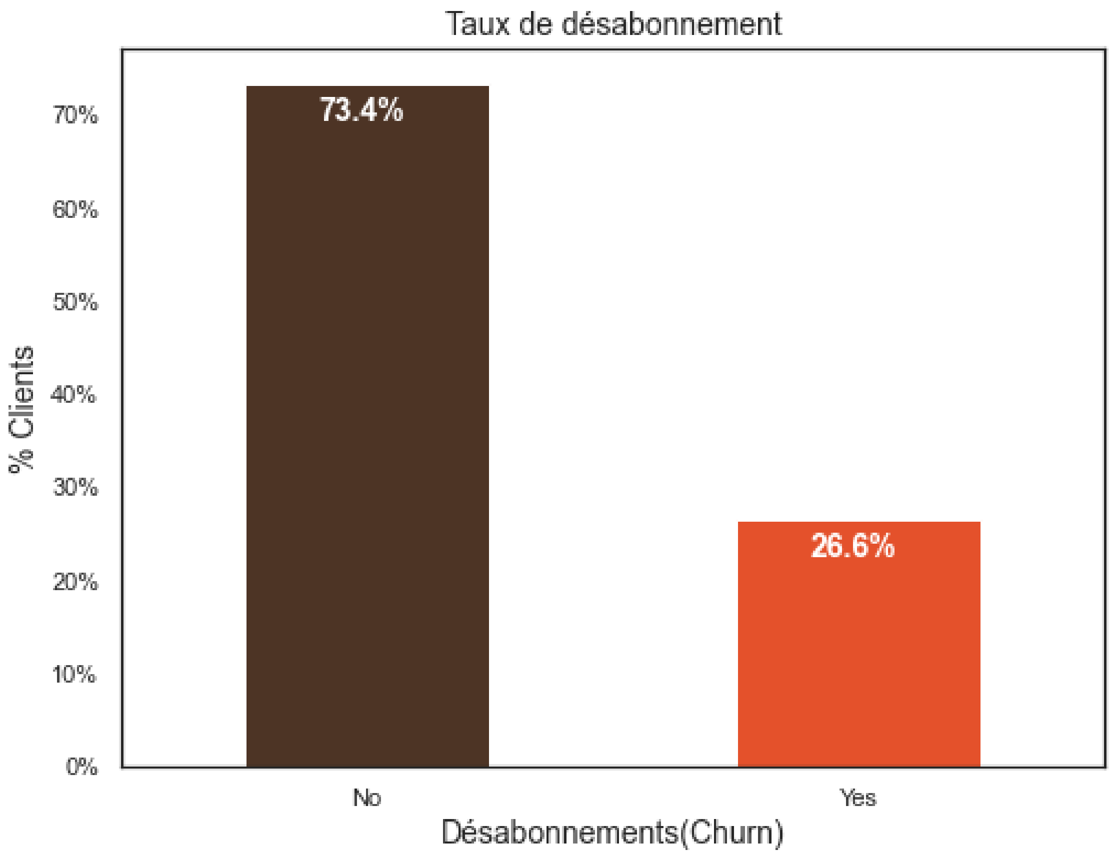
De cette figure nous constatons que le total des frais augmente au fur et à mesure que la facture mensuelle d'un client augmente.

V. Une analyse sur l'étiquettte (churn)

V.1 Taux de résiliation

[28]:
```
colors = ['#4D3425','#E4512B']
ax = (df['Churn Label'].value_counts()*100.0/len(df)).plot(kind='bar',
                                                            stacked = True,
                                                            rot = 0,
                                                            color = colors,
                                                            figsize = (8,6))
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.set_ylabel('% Clients',size = 14)
ax.set_xlabel('Désabonnements(Churn)',size = 14)
ax.set_title('Taux de désabonnement', size = 14)


#On affiche le taux sur chaque barre de l'histogramme
totals = []

for i in ax.patches:
    totals.append(i.get_width())
```

```
total = sum(totals)

for i in ax.patches:
    ax.text(i.get_x()+.15, i.get_height()-4.0, \
            str(round((i.get_height()/total), 1))+'%',
            fontsize=12,
            color='white',
          weight = 'bold',
          size = 14)
```



Ainsi:

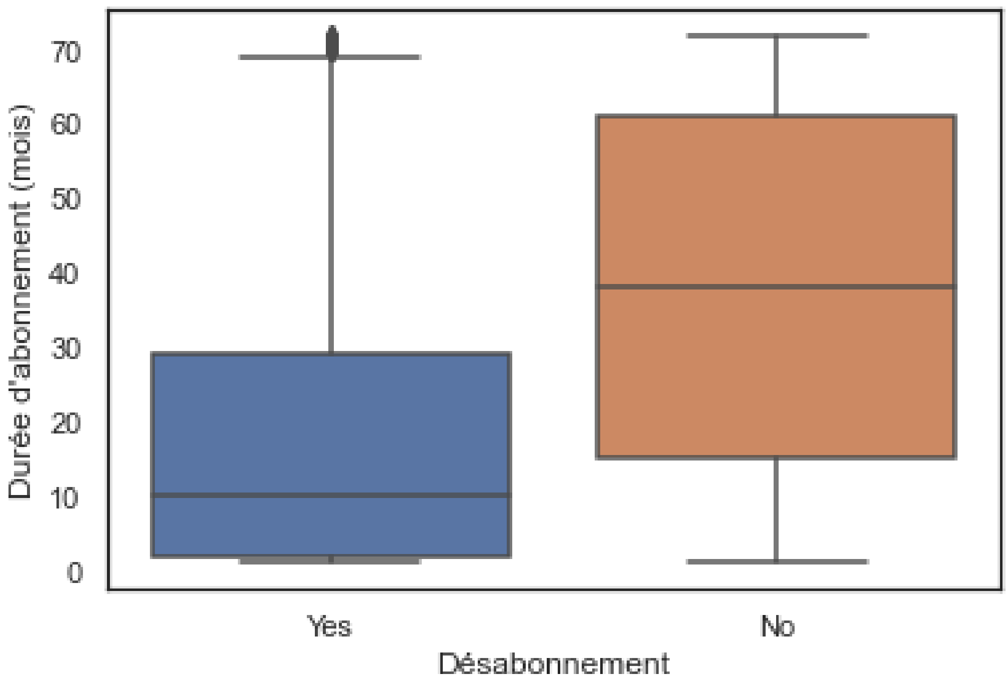
73.4% des clients ne sont pas désabonnés

26.4% des clients se sont désabonnées

V.2 Désabonnement & Durée d'abonnements

[29]:
```
sns.boxplot(x = df['Churn Label'], y = df['Tenure Months']).
 ↪set(xlabel='Désabonnement',ylabel="Durée d'abonnement (mois)")
```

[29]: `[Text(0.5, 0, 'Désabonnement'), Text(0, 0.5, "Durée d'abonnement (mois)")]`

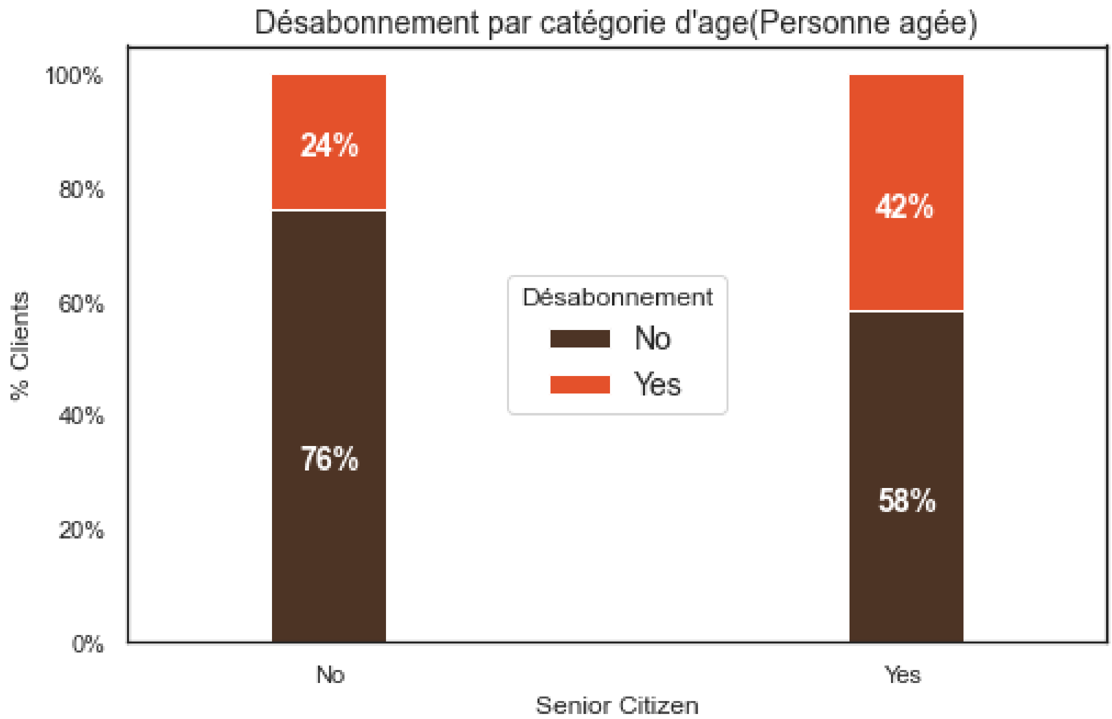


Ceux qui résilient leur contrat sont pour la majorité ceux qui ne durent pas dans l'entreprise

V.3 Désabonnement & la catégorie d'age

[30]:
```
colors = ['#4D3425','#E4512B']
seniority_churn = df.groupby(['Senior Citizen','Churn Label']).size().unstack()

ax=(seniority_churn.T*100.0/seniority_churn.T.sum()).T.plot(kind='bar',
                                                            width=0.2,
                                                            stacked=True,
                                                            rot=0,
                                                            figsize=(8,5),
                                                            color=colors
                                                           )
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.legend(loc='center',prop={'size':14},title='Désabonnement')
ax.set_ylabel('% Clients')
ax.set_title("Désabonnement par catégorie d'age(Personne agée)",size = 14)

for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.
 ↪4*height),
                color = 'white',
```

```
weight = 'bold',size =14)
```



Les clients qui resilient plus leur contrat sont les personnes les agées (2 fois plus elevé que les jeunes)

V.4 Désabonnement, charges Mensuelles & Quelques services

[31]:
```
color_op = ['#5527A0', '#BB93D7', '#834CF7', '#6C941E', '#93EAEA', '#7425FF',␣
↪'#F2098A', '#7E87AC',
            '#EBE36F', '#7FD394', '#49C35D', '#3058EE', '#44FDCF', '#A38F85',␣
↪'#C4CEE0', '#B63A05',
            '#4856BF', '#F0DB1B', '#9FDBD9', '#B123AC']

def PieChart(df_cat, df_value, title, limit=15):
    tmp_churn = df[df['Churn Value'] == 1].groupby(df_cat)[df_value].sum().
↪nlargest(limit).to_frame().reset_index()
    tmp_no_churn = df[df['Churn Value'] == 0].groupby(df_cat)[df_value].sum().
↪nlargest(limit).to_frame().reset_index()

    trace1 = go.Pie(labels=tmp_no_churn[df_cat],
                    values=tmp_no_churn[df_value], name= "No-Churn", hole= .5,
                    hoverinfo="label+percent+name+value", showlegend=True,
                    domain= {'x': [0, .48]})
```

```
trace2 = go.Pie(labels=tmp_churn[df_cat],
                values=tmp_churn[df_value], name="Churn", hole= .5,
                hoverinfo="label+percent+name+value", showlegend=False,
                domain= {'x': [.52, 1]})

layout = dict(title= title, height=450, font=dict(size=15),
              annotations = [
                  dict(
                      x=.20, y=.5,
                      text='No Churn',
                      showarrow=False,
                      font=dict(size=20)
                  ),
                  dict(
                      x=.80, y=.5,
                      text='Churn',
                      showarrow=False,
                      font=dict(size=20)
                  )
    ])

fig = dict(data=[trace1, trace2], layout=layout)
iplot(fig)
```

V.5.1 Internets

[32]:
```
no_churn_monthly_renenue = df['Monthly Charges'].sum()
PieChart("Internet Service", 'Monthly Charges', "Désabonnement Par service␣
↪Internet", limit=10)
```

La majorité des clients souscrivent plus au service de Fibre optique et se désabonnent aussi le plus de ce service

V.5.2 Types de contrats

[33]:
```
PieChart("Contract", 'Monthly Charges', "Désabonnement par type de contrat avec␣
↪pourcentage de frais mensuelles", limit=10)
```

V.5.3 Multiples Lines

[34]:
```
PieChart("Multiple Lines", 'Monthly Charges', "Désabonnement par Types de␣
↪Lignes", limit=10)
```

V.5.3 Protection de l'appareil

[35]:
```
PieChart("Device Protection", 'Monthly Charges', "Désabonnement de client ayant␣
↪souscrit à un service internet avec protection de l'appareil", limit=10)
```

Un résumé général sur les services

[36]:
```
df.loc[:,'Engaged'] = np.where(df['Contract'] != 'Month-to-month', 1,0)
df.loc[:,'YandNotE'] = np.where((df['Senior Citizen']==0) & (df['Engaged']==0),␣
↪1,0)
df.loc[:,'ElectCheck'] = np.where((df['Payment Method'] == 'Electronic check')␣
↪& (df['Engaged']==0), 1,0)
df.loc[:,'fiberopt'] = np.where((df['Internet Service'] != 'Fiber optic'), 1,0)
df.loc[:,'StreamNoInt'] = np.where((df['Streaming TV'] != 'No internet␣
↪service'), 1,0)
df.loc[:,'NoProt'] = np.where((df['Online Backup'] != 'No') |\
                               (df['Device Protection'] != 'No') |\
                               (df['Tech Support'] != 'No'), 1,0)

df['TotalServices'] = (df[['Phone Service', 'Internet Service', 'Online␣
↪Security',
                           'Online Backup', 'Device Protection',␣
↪'Tech Support',
                           'Streaming TV', 'Streaming Movies']]==␣
↪'Yes').sum(axis=1)
```

[37]:
```
from sklearn.preprocessing import LabelEncoder

#encodage de l'etiquette
le = LabelEncoder()

tmp_churn = df[df['Churn Value'] == 1]
tmp_no_churn = df[df['Churn Value'] == 0]

bi_cs = df.nunique()[df.nunique() == 2].keys()
dat_rad = df[bi_cs]

for cols in bi_cs :
    tmp_churn[cols] = le.fit_transform(tmp_churn[cols])


data_frame_x = tmp_churn[bi_cs].sum().reset_index()
data_frame_x.columns  = ["feature","yes"]
data_frame_x["no"]    = tmp_churn.shape[0]  - data_frame_x["yes"]
data_frame_x  = data_frame_x[data_frame_x["feature"] != "Churn"]

#nombre de 1 (oui)
trace1 = go.Scatterpolar(r = data_frame_x["yes"].values.tolist(),
                         theta = data_frame_x["feature"].tolist(),
                         fill  = "toself",name = "Churn 1's",
                         mode = "markers+lines", visible=True,
                         marker = dict(size = 5)
                        )
```

```
#nombre de 0 (non)
trace2 = go.Scatterpolar(r = data_frame_x["no"].values.tolist(),
                         theta = data_frame_x["feature"].tolist(),
                         fill  = "toself",name = "Churn 0's",
                         mode = "markers+lines", visible=True,
                         marker = dict(size = 5)
                        )
for cols in bi_cs :
    tmp_no_churn[cols] = le.fit_transform(tmp_no_churn[cols])

data_frame_x = tmp_no_churn[bi_cs].sum().reset_index()
data_frame_x.columns  = ["feature","yes"]
data_frame_x["no"]    = tmp_no_churn.shape[0]  - data_frame_x["yes"]
data_frame_x  = data_frame_x[data_frame_x["feature"] != "Churn"]

#nombre de 1(oui)
trace3 = go.Scatterpolar(r = data_frame_x["yes"].values.tolist(),
                         theta = data_frame_x["feature"].tolist(),
                         fill  = "toself",name = "NoChurn 1's",
                         mode = "markers+lines", visible=False,
                         marker = dict(size = 5)
                        )

#nombre de 0(non)
trace4 = go.Scatterpolar(r = data_frame_x["no"].values.tolist(),
                         theta = data_frame_x["feature"].tolist(),
                         fill  = "toself",name = "NoChurn 0's",
                         mode = "markers+lines", visible=False,
                         marker = dict(size = 5)
                        )

data = [trace1, trace2, trace3, trace4]

updatemenus = list([
    dict(active=0,
         x=-0.15,
         buttons=list([
            dict(
                label = 'Dist Désabonné',
                 method = 'update',
                 args = [{'visible': [True, True, False, False]},
                     {'title': "Repartition du nombre de Clients s'étant␣
 ↪désabonné par service"}]),

             dict(
                  label = 'Dest Non-Désabonné',
                 method = 'update',
```

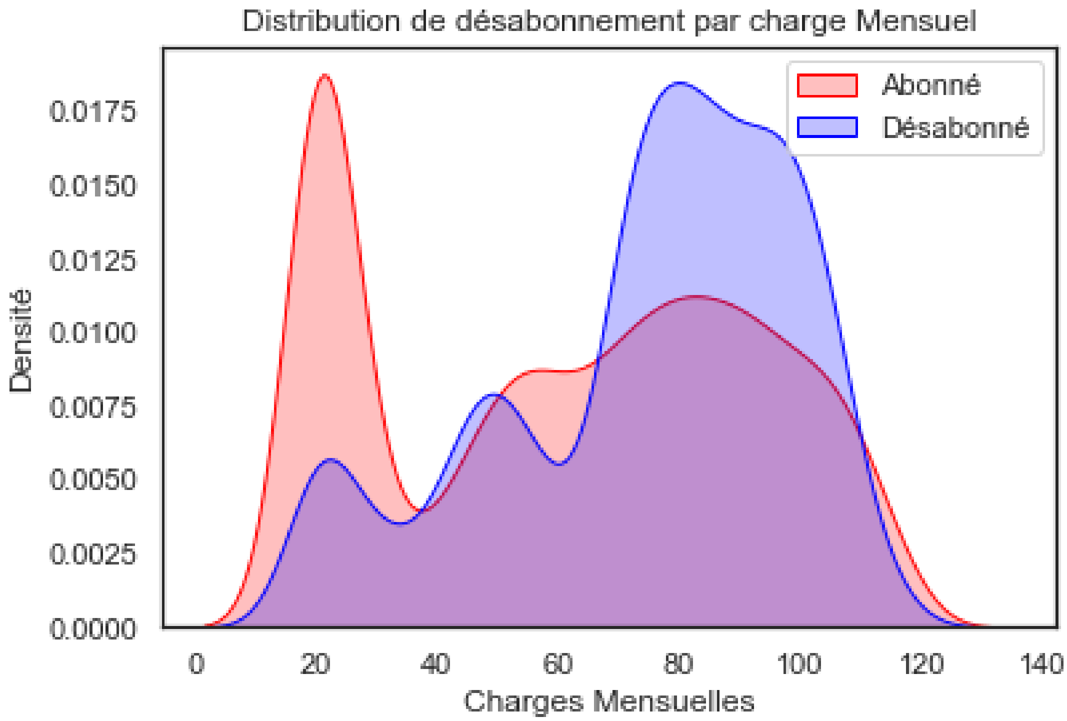
```
                 args = [{'visible': [False, False, True, True]},
                     {'title': "Repartition du nombre de Clients ne s'étant pas␣
 ↪désabonné par service"}]),

        ]),
    )
])

layout = dict(title='ScatterPolar de désabonnement des Clients',
              showlegend=False,
              updatemenus=updatemenus)

fig = dict(data=data, layout=layout)

iplot(fig)
```

```
C:\Users\gth\Anaconda3\lib\site-packages\ipykernel_launcher.py:13:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\gth\Anaconda3\lib\site-packages\ipykernel_launcher.py:37:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

V.6 Désabonnement & les charges mensuelles

[38]:
```
ax=sns.kdeplot(df['Monthly Charges'][(df['Churn Label']=='No')],
              color='Red',shade=True)
ax=sns.kdeplot(df['Monthly Charges'][(df['Churn Label']=='Yes')],
              color='Blue',shade=True)
ax.legend(["Abonné","Désabonné"],loc='upper right')
ax.set_ylabel('Densité')
ax.set_xlabel('Charges Mensuelles')
ax.set_title('Distribution de désabonnement par charge Mensuel')
```

[38]: Text(0.5, 1.0, 'Distribution de désabonnement par charge Mensuel')

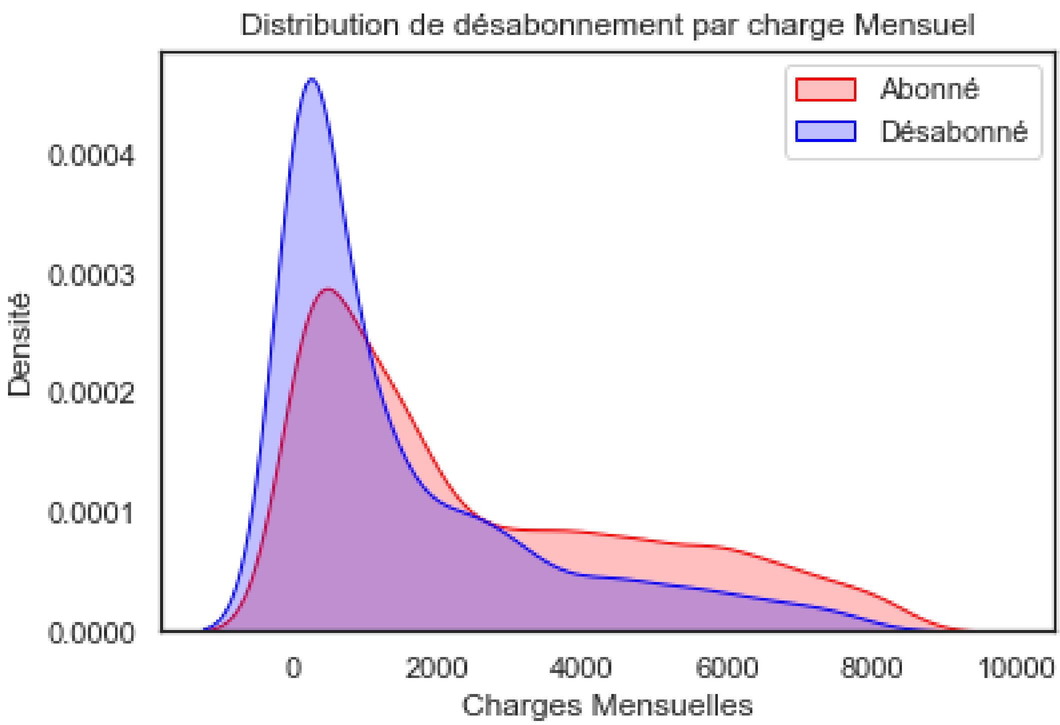


On peut voir que plus la charge mensuelle augmente, plus les clients se désabonnent

V.6 Désabonnement & les charges Totales

[39]:
```
ax=sns.kdeplot(df['Total Charges'][(df['Churn Label']=='No')],
              color='Red',shade=True)
ax=sns.kdeplot(df['Total Charges'][(df['Churn Label']=='Yes')],
              color='Blue',shade=True)
ax.legend(["Abonné","Désabonné"],loc='upper right')
ax.set_ylabel('Densité')
ax.set_xlabel('Charges Mensuelles')
ax.set_title('Distribution de désabonnement par charge Mensuel')
```

[39]: Text(0.5, 1.0, 'Distribution de désabonnement par charge Mensuel')



Le graphe laisse voir que plus le taux de charges totales augmentent, moins il ya de résilitation

Quelques Algorithmes Prédictifs

[620]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest,chi2
import seaborn as sns
import operator
from tabulate import tabulate
```

[621]:
```
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import␣
↪train_test_split,learning_curve,GridSearchCV,cross_val_score
from sklearn.svm import SVC,LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_selection import SelectKBest,f_classif
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from IPython.display import display, HTML
```

```
from sklearn.naive_bayes import␣
↪ComplementNB,GaussianNB,MultinomialNB,BernoulliNB,CategoricalNB
from sklearn.metrics import␣
↪auc,confusion_matrix,precision_score,precision_recall_fscore_support␣
↪,f1_score,consensus_score,recall_score,accuracy_score,plot_confusion_matrix,classification_
from xgboost import XGBClassifier # à installer
from lightgbm import LGBMClassifier # à installer
```

[622]:
```
pd.set_option("Display.max_columns",100)
import warnings
warnings.filterwarnings('ignore')
```

[623]:
```
df= pd.read_excel("Telco_customer_churn.csv.xlsx")

encodeur = LabelEncoder()
scaler=StandardScaler()
```

Nettoyage

[624]:
```
df["Total Charges"]=pd.to_numeric(df["Total Charges"],errors="coerce")
df["Total Charges"]=df["Total Charges"].astype(float)
df=df[df["Total Charges"].notna()]
```

[625]:
```
print("il y a {} valeur nulle".format(df.iloc[:,:-1].isnull().sum().sum()) )
```

```
il y a 0 valeur nulle
```

Fonctions utiles

[657]:
```
# pour evaluer une liste d'algorithme
def␣
↪evaluation(x_train,y_train,x_test,y_test,modelList={},confusion=False,roc_cu=True,learning_
↪
    areas={}
    models={}
    if len(modelList)!=0:
        if roc_cu:
            plt.figure(figsize=(12,8))
            plt.plot([0,1],[0,1],"r")
        for k,model in modelList.items():
            model.fit(x_train,y_train)
            models[k]=model
            if roc_cu:
                probs = model.predict_proba(x_test)[:,1]
                fpr , tpr , thresholds = roc_curve(y_test,probs)
                areas[k]=roc_auc_score(y_test,probs)
                plt.plot(fpr,tpr,"",label=k)
        if roc_cu:
```

```
            plt.legend()
            plt.show()

    areas = dict(sorted(areas.items(),key=operator.itemgetter(1),reverse=True))

    for k,area in areas.items():
        print(f"l aire du modele {k} est {area}")

    if confusion:
        for k,model in models.items():
            plt.figure()
            y_pred = model.predict(x_test)
            print(f"modele {k}")
            plot_confusion_matrix(model,x_test,y_test,display_labels=["No␣
↪churn","Churn"])
           ␣
↪print(classification_report(y_test,y_pred,digits=6,target_names=["No␣
↪churn","Churn"]))
            plt.show()

    if learning_cu:
        for k,model in models.items():
           ␣
↪N,train_score,val_score_train=learning_curve(model,x_train,y_train,cv=5,scoring="f1",train_
↪linspace(0.1,1,10))
           ␣
↪#N,test_score,val_score_test=learning_curve(model,x_test,y_test,cv=5,scoring="f1",train_siz
↪linspace(0.1,1,10))
            plt.figure(figsize=(12,8))
            plt.plot(N,train_score.mean(axis=1),label="Train score")
            plt.plot(N,val_score_train.mean(axis=1),label="validation score␣
↪train")
            #plt.plot(N,test_score.mean(axis=1),label="Test score")
            #plt.plot(N,val_score_test.mean(axis=1),label="validation score␣
↪test")
            plt.title(f"Modèle {k} ")
            plt.legend()
    return models


def featuresImportances(model,index,nb_features="None"):
    plt.figure(figsize=(14,5))
    serie=pd.Series(model.feature_importances_,index=index).
↪sort_values(ascending=False)
    if nb_features=="None":
        serie.plot.bar(log=True)
```

```python
    else:
        serie[:nb_features].plot.bar(log=True)

def final_model(model,X,seuil=0):
    return model.decision_function(X) > seuil

def comparaison_models(models,tests_set={}):
    arrays=[
        np.array([]),np.array([])
    ]
    d=np.array([])
    for nom,model in models.items():
        arrays[0]=np.append(arrays[0],[nom,nom])
        arrays[1]=np.append(arrays[1],["No churn","Churn"])

    for name,model in models.items():
        x_test,y_test=tests_set.get(name)[0],tests_set.get(name)[1]
        y_pred = model.predict(x_test)
        res=precision_recall_fscore_support(y_test,y_pred)
        accuracy=accuracy_score(y_test,y_pred)
        res=list(res)
        res.insert(3,np.array([accuracy,accuracy]))
        res=tuple(res)
        d=np.append(d,np.array([val[0]for val in res]))
        d=np.append(d,np.array([val[1]for val in res]))

    d=d.reshape(len(models)*2,5)
    arrays=np.array(arrays)
    data=pd.
 ↪DataFrame(d,columns=["précison","recall","f1_score","accuracy","test_size"])
    data.iloc[:,:-1]=data.iloc[:,:-1].apply(lambda x:x*100)
    data["Modèles"]=arrays[0,:]
    data["Etat"]=arrays[1,:]
    data=data.set_index(['Modèles'])
    print(tabulate(data, tablefmt='grid',headers="keys",showindex="always"))
    #data=data.set_index(['Modèles',"accuracy",'Etat'])
    #display(HTML(data.to_html()))

def precision_recall_f1(model):
    y_pred=model.predict(x_test)
    res=precision_recall_fscore_support(y_test,y_pred)
    data=pd.
 ↪DataFrame(res,index=["précison","recall","f1_score","test_size"],columns=["No␣
 ↪churn","Churn"])
    sns.heatmap(data.iloc[:-1,:].apply(lambda x:x*100),vmin=0,annot=True,fmt=".
 ↪6g",vmax=100)
```

NAIVES BAYES

[627]:
```
df1=df

df1['Gender'] = df1['Gender'].map({'Male': 1, 'Female': 0})
df1['Streaming Movies'] = df1['Streaming Movies'].map({'Yes': 1, 'No': 0, 'No␣
↪internet service':0})
df1['Streaming TV'] = df1['Streaming TV'].map({'Yes': 1, 'No': 0, 'No internet␣
↪service':0})
df1['Tech Support'] = df1['Tech Support'].map({'Yes': 1, 'No': 0, 'No internet␣
↪service':0})
df1['Device Protection'] = df1['Device Protection'].map({'Yes': 1, 'No': 0, 'No␣
↪internet service':0})
df1['Online Backup'] = df1['Online Backup'].map({'Yes': 1, 'No': 0, 'No␣
↪internet service':0})
df1['Online Security'] = df1['Online Security'].map({'Yes': 1, 'No': 0, 'No␣
↪internet service':0})
df1['Contract'] = df1['Contract'].map({'Month-to-month': 1, 'One year': 0, 'Two␣
↪year':2})
df1['Internet Service'] = df1['Internet Service'].map({'Fiber optic': 1, 'No':␣
↪0, 'DSL':2})
df1['Total Charges'] = pd.to_numeric(df1['Total Charges'],errors='coerce')
df1['Multiple Lines'] = df1['Multiple Lines'].map({'Yes': 1, 'No': 0, 'No phone␣
↪service':0})
df1['Phone Service'] = df1['Phone Service'].map({'Yes': 1, 'No': 0, 'No phone␣
↪service':0})
df1['Dependents'] = df1['Dependents'].map({'Yes': 1, 'No': 0, 'No internet␣
↪service':0})
df1['Partner'] = df1['Partner'].map({'Yes': 1, 'No': 0, 'No internet service':
↪0})
df1['Senior Citizen'] = df1['Senior Citizen'].map({'Yes': 1, 'No': 0, 'No␣
↪internet service':0})
df1['Paperless Billing'] = df1['Paperless Billing'].map({'Yes': 1, 'No': 0, 'No␣
↪internet service':0})
df1['Payment Method'] = df1['Payment Method'].map({'Mailed check': 1,␣
↪'Electronic check': 0, 'Bank transfer (automatic)':2, 'Credit card␣
↪(automatic)': 3,})
df1=df1.drop(['CustomerID','Count','Country','State','City','Zip Code','Lat␣
↪Long','Latitude','Longitude','Churn Label','Churn Reason'],axis=1)
df_fic=pd.get_dummies(df1) # pour transformer toutes les variables categoriques␣
↪en num
```

[628]:
```
#transformer les catégories de chaque caractéristique comme des valeurs␣
↪numériques
#On fait des changements a notre dataset en changeant toutes les donées en␣
↪nombres
```

```python
df_naives=df1

for col in df_naives.columns:
    df_naives[col] = encodeur.fit_transform(df_naives[col])

#séparer les entrées (caractéristiques) et la sortie (classe)
#On crée X qui contient nos données et Y qui contient les classes
y = df_naives['Churn Value']  #les résulats (classes)
X = df_naives.drop(['Churn Value'], axis =1) #les caractéristiques

X_trainN, X_testN, y_trainN, y_testN = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0,stratify=y)
```

[629]:

```python
#Cette fonction est essentiel pour le naive bayes classifier vu qu'il présente␣
 ↪plusieurs méthodes : Gaussien Multinomial et Bernoulli.
#On va calculer le score de chaqu'une de ces méthodes .

nb = {'gaussian': GaussianNB(),
      'bernoulli': BernoulliNB(),
      'multinomial': MultinomialNB()}
scores = {}
for key, model in nb.items():
    s = cross_val_score(model, X_trainN, y_trainN, cv=5, scoring='accuracy')
    scores[key] = np.mean(s)
scores
```

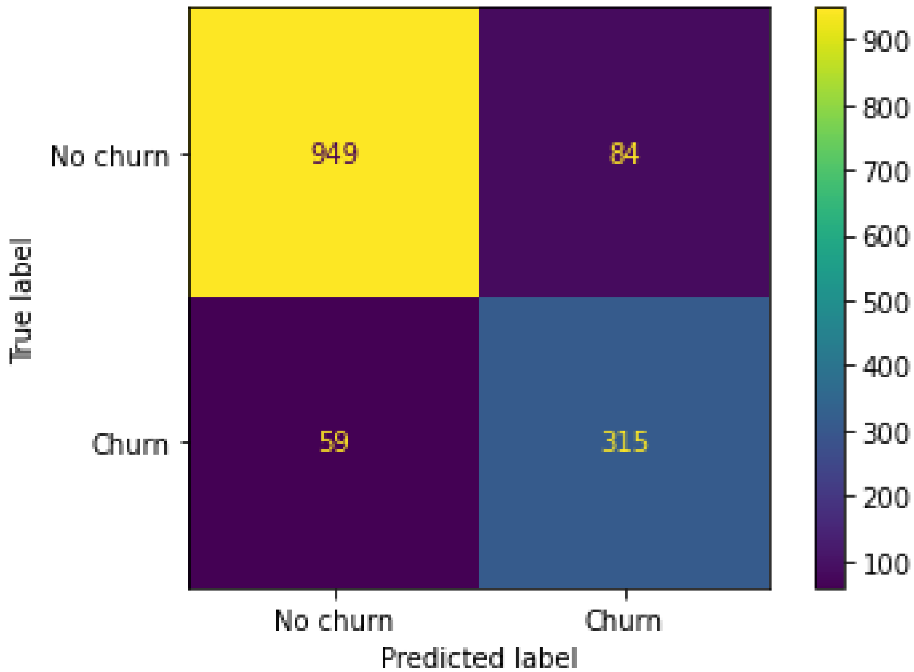
[629]:

```
{'gaussian': 0.8796444444444445,
 'bernoulli': 0.7848888888888889,
 'multinomial': 0.7409777777777777}
```

[648]:

```python
# modele à retenir celon celui qui a le meilleur score
modele = GaussianNB()

NaiveBayes=evaluation(X_trainN,y_trainN,X_testN,y_testN,modelList={"NaiveBayes":
 ↪modele},confusion=True,roc_cu=False,learning_cu=False)
```

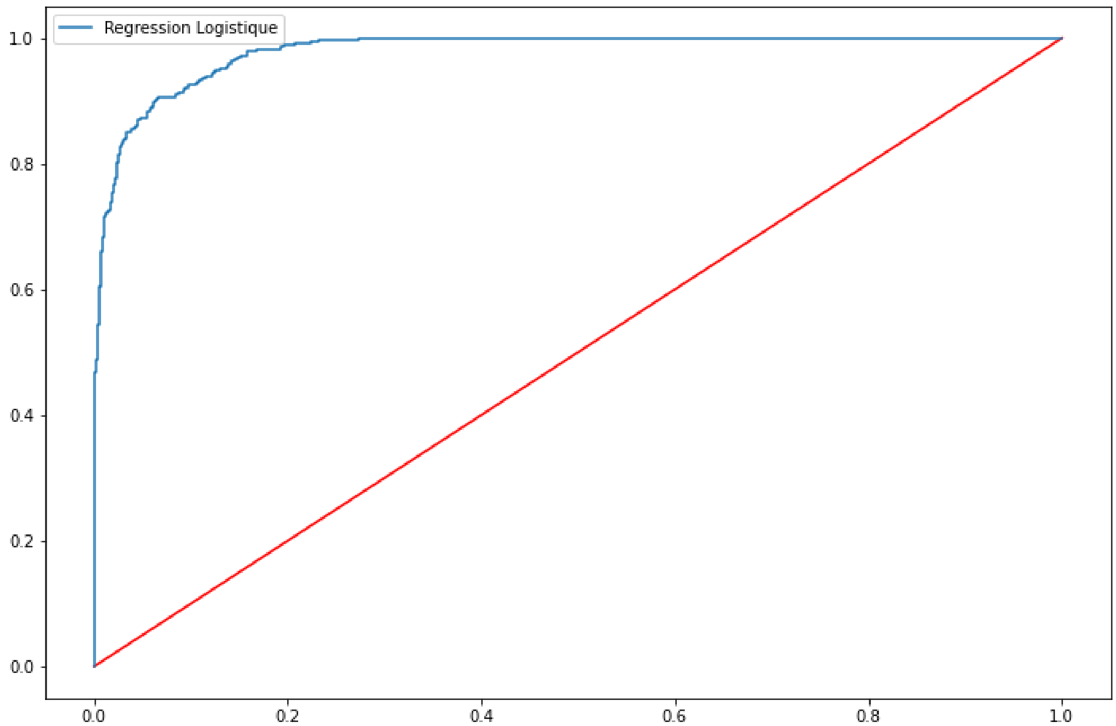
```
modele NaiveBayes
              precision    recall  f1-score   support

    No churn   0.941468  0.918683  0.929936      1033
       Churn   0.789474  0.842246  0.815006       374

    accuracy                       0.898365      1407
   macro avg   0.865471  0.880465  0.872471      1407
weighted avg   0.901066  0.898365  0.899386      1407


<Figure size 432x288 with 0 Axes>
```



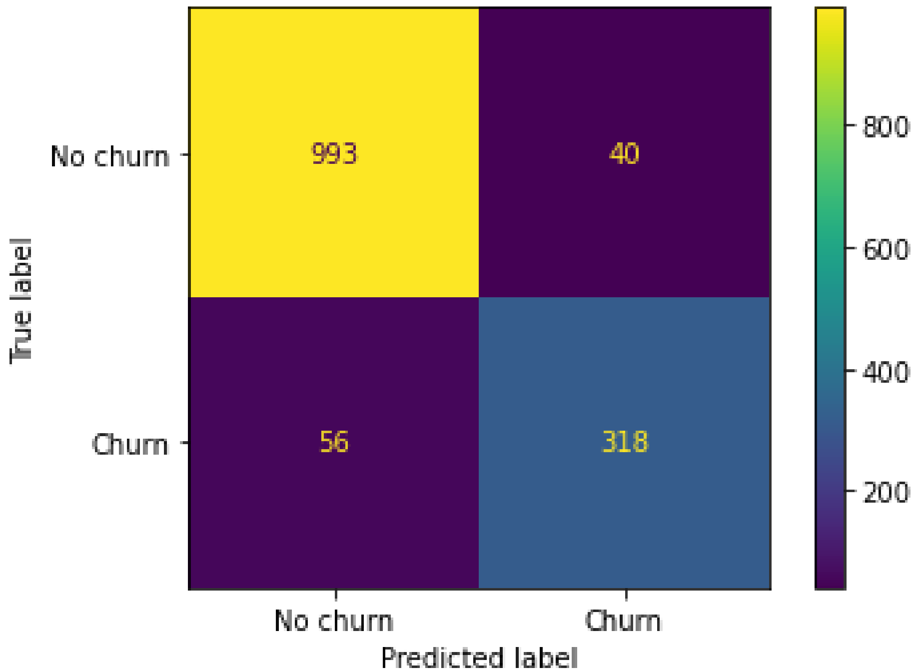
REGRESSION LOGISTIQUE

La régression logistique est utilisée pour le classement et pas la régression. Mais, elle est considéré comme une méthode de classification puisqu'elle sert à estimer la probabilité d'appartenir à une classe. Il y a trois types de régression logistique: - **Régression logistique binaire**: ici, le but de la classification est d'identifier si un échantillon appartient à une classe ou non. - **Régression logistique multinomiale**: ici, le but de la classification est d'identifier à quelle classe appartient-t-il un échantillon parmi plusieurs classes. - **Régression logistique ordinale**: ici, le but de la classification est de chercher la classe d'un échantillon parmi des classes ordonnées. Un exemple de classes: non satisfait, satisfait, très sataisfait.

[631]:
```
X = df1.drop("Churn Value", axis=1)
y = df1["Churn Value"]
X_trainL, X_testL, y_trainL, y_testL = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0,stratify=y)

X_trainL = scaler.fit_transform(X_trainL)
X_testL = scaler.transform(X_testL)

logreg = LogisticRegression(random_state=0)

regressionL=evaluation(X_trainL,y_trainL,X_testL,y_testL,modelList={"Regression␣
 ↪Logistique":logreg},confusion=True,roc_cu=True,learning_cu=False)
```




```
l aire du modele Regression Logistique est 0.9791480087590787
modele Regression Logistique
              precision    recall  f1-score   support

    No churn   0.946616  0.961278  0.953890      1033
       Churn   0.888268  0.850267  0.868852       374

    accuracy                       0.931770      1407
   macro avg   0.917442  0.905773  0.911371      1407
weighted avg   0.931106  0.931770  0.931286      1407


<Figure size 432x288 with 0 Axes>
```



[632]:
```
param =  {'penalty': ['l1', 'l2', 'elasticnet', 'none'],
          'C' : np.linspace(0,10, 100),
          'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
          'max_iter': [100,1000, 2000, 3000]}

log = LogisticRegression(random_state=0)
grid = GridSearchCV(log, param_grid=param, cv =5, n_jobs=-1)
#grid.fit(X_train, y_train)
#grid.best_params_
```

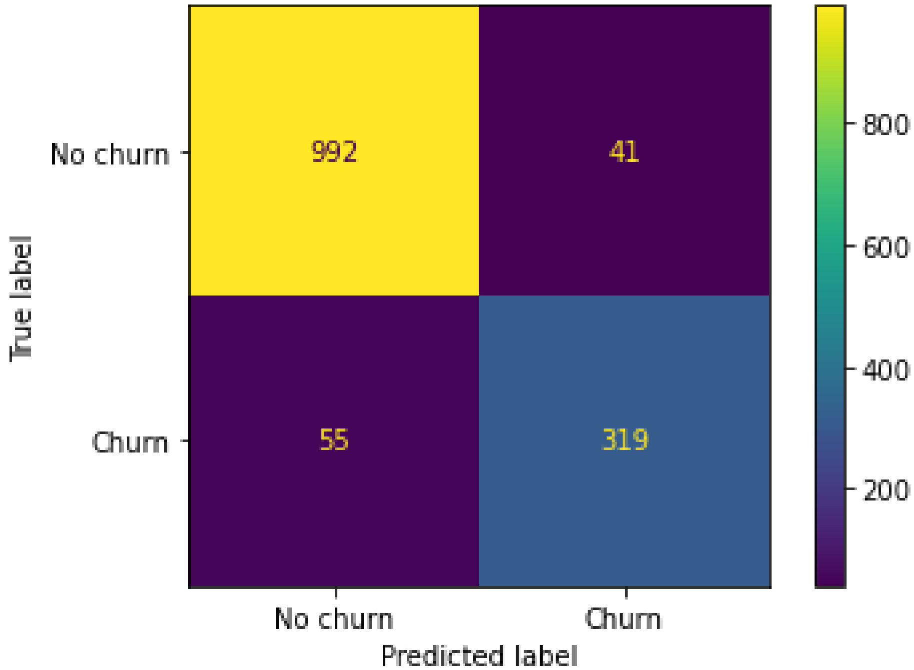
best parameters : {'C': 0.20202020202020202, 'max_iter': 100, 'penalty': 'l2', 'solver': 'liblinear'}

[633]:
```
final_modelL = LogisticRegression(random_state=0, C=0.20202, penalty = 'l2',␣
 ↪solver='liblinear', tol=0.2)

regressionL=evaluation(X_trainL,y_trainL,X_testL,y_testL,modelList={"Regression␣
 ↪Logistique":final_modelL},confusion=True,roc_cu=False,learning_cu=False)
```

```
modele Regression Logistique
              precision    recall  f1-score   support

    No churn   0.947469  0.960310  0.953846      1033
       Churn   0.886111  0.852941  0.869210       374

    accuracy                       0.931770      1407
   macro avg   0.916790  0.906625  0.911528      1407
```

```
weighted avg   0.931159  0.931770  0.931349      1407
```

```
<Figure size 432x288 with 0 Axes>
```



RANDOM FOREST CLASSIFIER

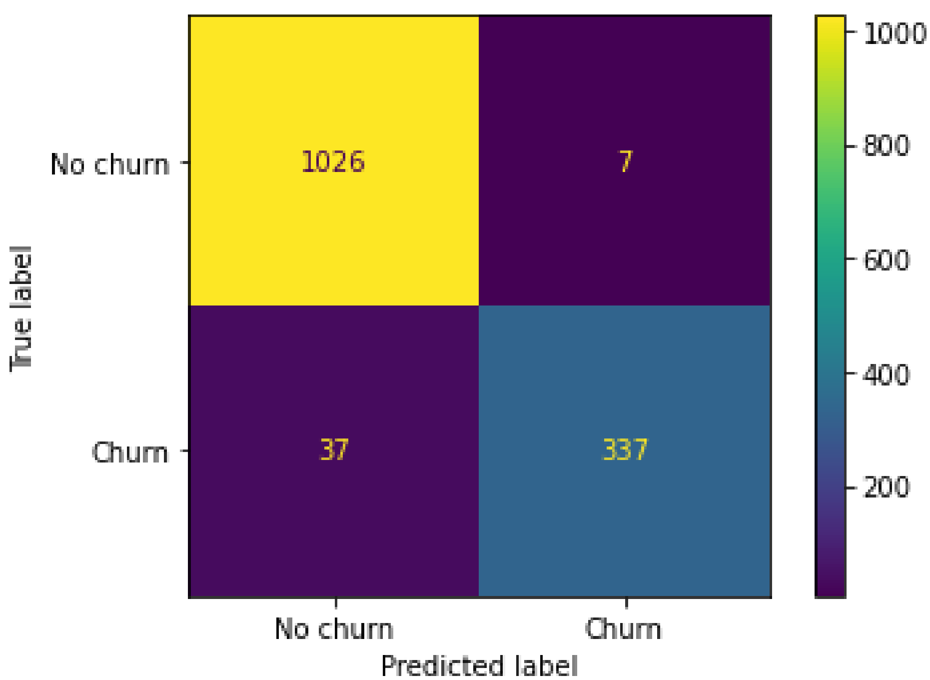
[634]:
```
# Drop useless columns
dfR = df.drop(['Count','Country','State','CustomerID','Lat Long','Churn␣
↪Label'],axis=1)
#One Hot Encoing using get_dummies method
dfR = pd.get_dummies(dfR, columns = ['Contract','Dependents','Device␣
↪Protection','Gender',
                                    'Internet Service','Multiple Lines','Online␣
↪Backup',
                                    'Online Security','Paperless␣
↪Billing','Partner',
                                    'Payment Method','Phone Service','Senior␣
↪Citizen',
                                    'Streaming Movies','Streaming TV','Tech␣
↪Support','City','Churn Reason'])
```

[635]:
```
#Create Feature variable X and target variable y
y = dfR['Churn Value']
X = dfR.drop(['Churn Value'], axis =1)
```

```
X_trainR, X_testR,y_trainR,y_testR = train_test_split(X, y, test_size = 0.20,␣
 ↪random_state = 0,stratify = y)

rfmodel = RandomForestClassifier(n_estimators=100 ,␣
 ↪criterion='gini',random_state=0) # par defaut au prémier lieux
                                                                              ␣
 ↪        #justifier les choix des paramètres
randomF=evaluation(X_trainR,y_trainR,X_testR,y_testR,modelList={"Random Forest":
 ↪rfmodel},confusion=True,roc_cu=False,learning_cu=False)
```

```
modele Random Forest
              precision    recall  f1-score   support

    No churn   0.965193  0.993224  0.979008      1033
       Churn   0.979651  0.901070  0.938719       374

    accuracy                       0.968728      1407
   macro avg   0.972422  0.947147  0.958863      1407
weighted avg   0.969036  0.968728  0.968298      1407
```

```
<Figure size 432x288 with 0 Axes>
```



Famille de Gradient Boosting

[636]:
```
df= pd.read_excel("Telco_customer_churn.csv.xlsx")
df["Total Charges"]=pd.to_numeric(df["Total Charges"],errors="coerce")
df["Total Charges"]=df["Total Charges"].astype(float)
df=df[df["Total Charges"].notna()]
```

Features engineering

Création de nouvelles variables dans le but d'ameliorer la prédiction

NB ces variables créées ont découlé de la compréhension du dataset

[637]:
```
#service_cols=["Phone Service","Multiple Lines","Internet Service","Online␣
↪Security","Online Backup","Device Protection","Tech Support","Streaming␣
↪TV","Streaming Movies","Paperless Billing"]

service_cols=['Phone Service', 'Internet Service', 'Online Security','Online␣
↪Backup', 'Device Protection', 'Tech Support','Streaming TV', 'Streaming␣
↪Movies']

execptions=["Gender",        "Senior␣
↪Citizen",        "Partner",        "Dependents",        "Phone␣
↪Service"        ,"Paperless Billing",        "Churn Label","Churn␣
↪Score",        "Engaged",        "ElectCheck",        "fiberopt",        "StreamNoInt",

df.loc[:,'Engaged'] = np.where(df['Contract'] != 'Month-to-month', 1,0) #␣
↪client avec un contrat month-to-month
df.loc[:,'YandNotE'] = np.where((df['Senior Citizen']==0) & (df['Engaged']==0),␣
↪1,0) # si client est jeune et n'a pas souscrit a un abonnement month-to-month
df.loc[:,'ElectCheck'] = np.where((df['Payment Method'] == 'Electronic check')␣
↪& (df['Engaged']==0), 1,0) #si le client a paye par chèque electronique et␣
↪n'a pas souscrit à un payement mensuel
df.loc[:,'fiberopt'] = np.where((df['Internet Service'] != 'Fiber optic'), 1,0)␣
↪# si le client utilise internet par fibre optique
df.loc[:,'StreamNoInt'] = np.where((df['Streaming TV'] != 'No internet␣
↪service'), 1,0) # si le client utilise le streaming TV sans internet
df.loc[:,'NoProt'] = np.where((df['Online Backup'] != 'No') | (df['Device␣
↪Protection'] != 'No') | (df['Tech Support'] != 'No'), 1,0) # si le client␣
↪n'utilise pas au moins un service supplémentaire au service internet

df['nb_subscriptions'] = (df[service_cols]== 'Yes').sum(axis=1) # le nombre de␣
↪service auxquels le client a souscrit

target_col=["Churn Label"]
cat_col=[]
binary_col=[]
multi_cat_col=[]
delete_col=["Churn Value","Count","Zip Code",'City','Zip Code','Lat␣
↪Long','Latitude','Longitude',"CustomerID","Churn Reason"]
```

```
cat_col=df.nunique()[df.nunique() < 10].keys().tolist()
cat_col=[x for x in cat_col if x not in delete_col]
binary_col = df.nunique()[df.nunique() == 2].keys().tolist()
multi_cat_col=[x for x in cat_col if x not in binary_col]
```

[638]:
```
print(delete_col)
df=df.drop(delete_col,axis=1,errors="ignore")
```

```
['Churn Value', 'Count', 'Zip Code', 'City', 'Zip Code', 'Lat Long', 'Latitude',
'Longitude', 'CustomerID', 'Churn Reason']
```

Transformation des colones multicatégories en unicatégorie

[639]:
```
df=pd.get_dummies(data=df,columns=multi_cat_col)
```

Encodage

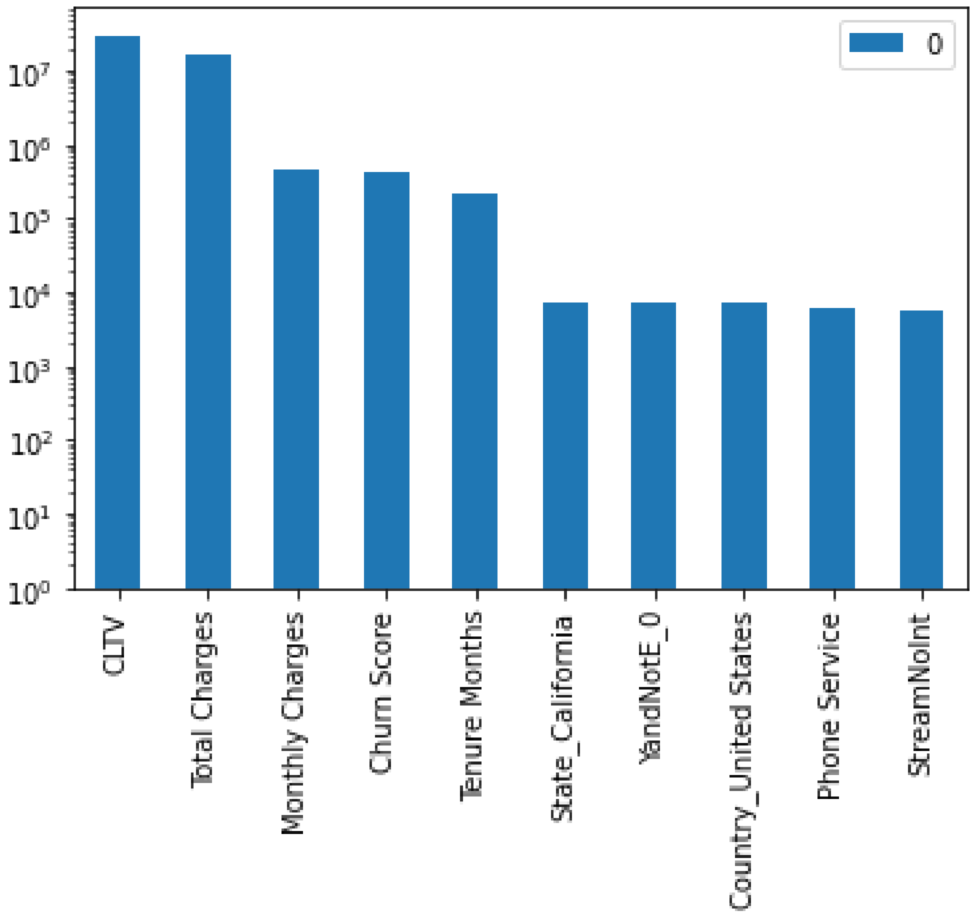
[640]:
```
#col_to_standardScale=[col for col in features.columns if col not in execptions]
#col_to_encode=[col for col in binary_col if col not in delete_col]
encoder = LabelEncoder()
for col in binary_col:
    if col not in delete_col:
        df[col]=encoder.fit_transform(df[col])
```

[641]:
```
features=df.drop(target_col,axis=1)
target=df[target_col].values.reshape(features.shape[0])
```

SelectKBest

[642]:
```
selector=SelectKBest(score_func=chi2,k=features.shape[1])
fit=selector.fit_transform(features,target)
dd=pd.DataFrame(fit.sum(axis=0))
dd.index =features.columns
dd.sort_values(dd.columns[0],ascending=False).iloc[:10].plot.bar(log=True)
```

[642]:
```
<AxesSubplot:>
```



Normalisation

[643]:
```
cols=[col for col in features.columns if col not in execptions]
scaler=StandardScaler()
data=scaler.fit_transform(df[cols])
right=pd.DataFrame(data,columns=cols)
left=pd.DataFrame(df[execptions],columns=execptions)
data=left.merge(right, left_index=True, right_index=True,how="left")
```

[644]:
```
data.dropna(axis=0,inplace=True)
features=data.drop(target_col,axis=1)
target=data[target_col].values.reshape(features.shape[0])
```

Division train_set et test_set

[645]:
```
x_train,x_test,y_train,y_test=train_test_split(features,target,test_size=0.
↪2,random_state=0,stratify=target)
```

1 Comparaison des modèles avec les paramètres de base

Les modèles sont classés de facon décroissante de performance par la courbe ROC

Les 3 modèles sont :

LGBMClassifier avec une aire de 0.9975258951124759

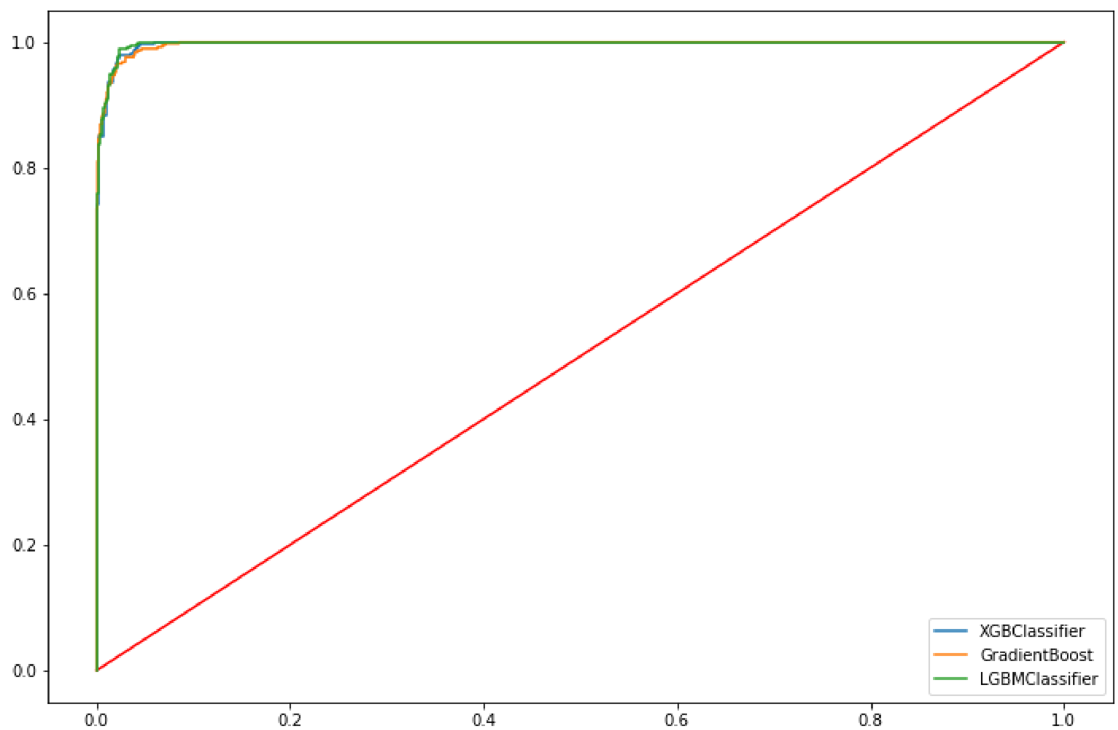
XGBClassifier avec une aire de 0.9971576321208317

GradientBoostingClassifier avec une aire de 0.9969670171216356

La courbe ROC est suivie du Classification repport, Confusion metric et Learning curve de chaque modèle

[646]:
```
xgb = XGBClassifier()
gboost=GradientBoostingClassifier()
lgbm=LGBMClassifier()
```

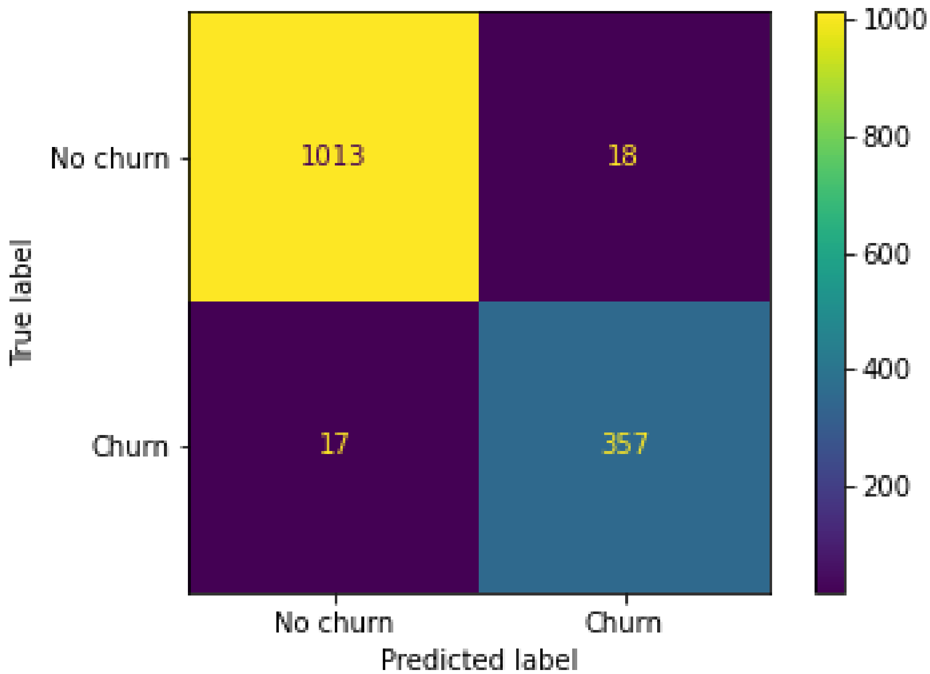
[438]:
```
modelList={"XGBClassifier":xgb,"GradientBoost":gboost,"LGBMClassifier":lgbm}
modelsBase=evaluation(x_train,y_train,x_test,y_test,modelList,True,learning_cu=True)
```




```
l aire du modele LGBMClassifier est 0.9975258951124759
l aire du modele XGBClassifier est 0.9971576321208317
l aire du modele GradientBoost est 0.9969670171216356
modele XGBClassifier
              precision    recall  f1-score   support
```

No churn	0.983495	0.982541	0.983018	1031
Churn	0.952000	0.954545	0.953271	374
accuracy			0.975089	1405
macro avg	0.967748	0.968543	0.968144	1405
weighted avg	0.975111	0.975089	0.975100	1405

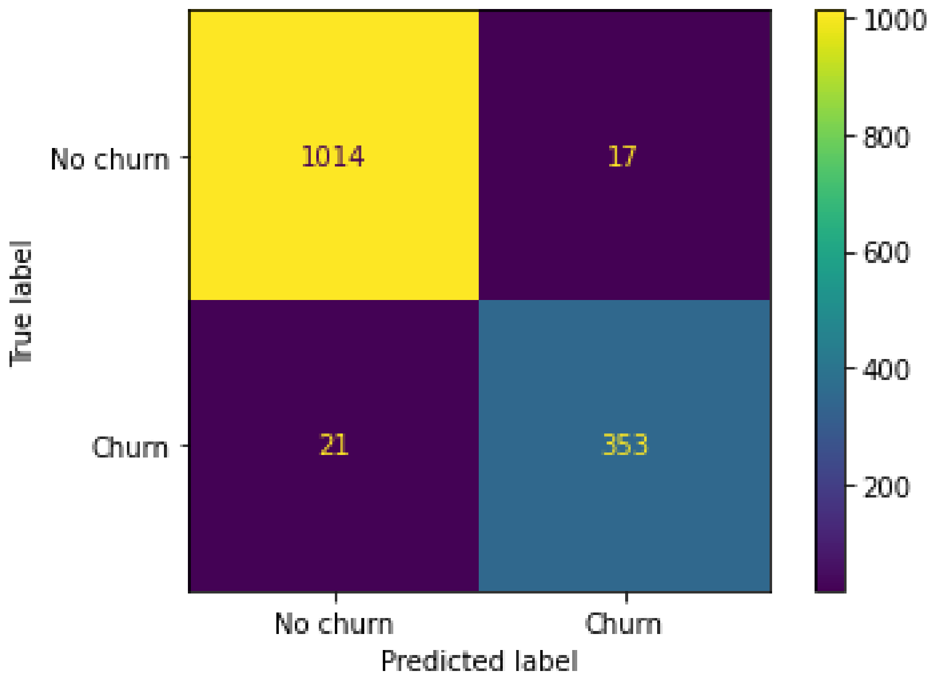
<Figure size 432x288 with 0 Axes>



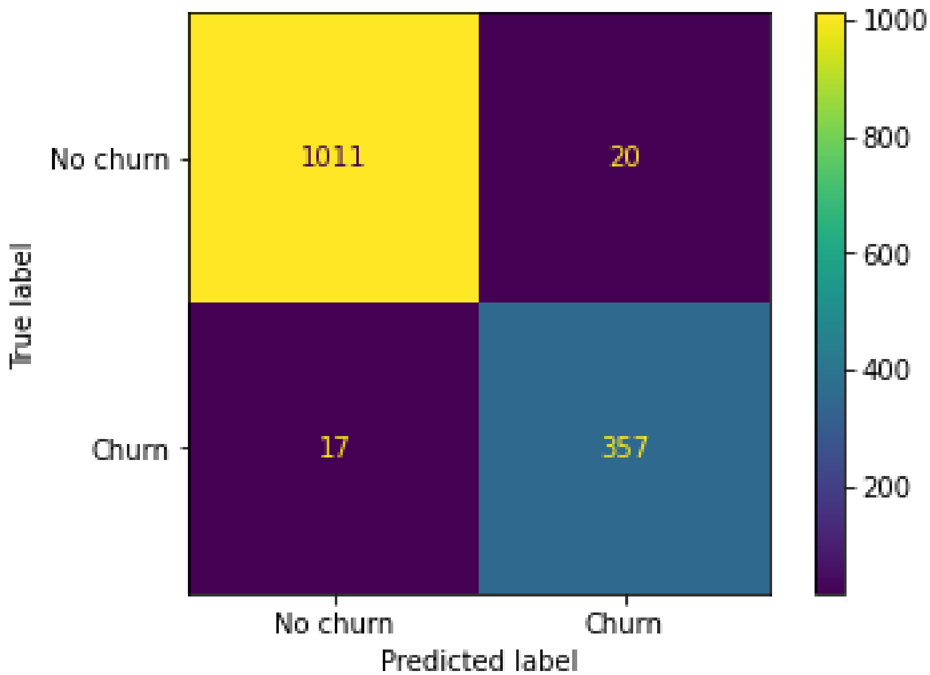
modele GradientBoost

	precision	recall	f1-score	support
No churn	0.979710	0.983511	0.981607	1031
Churn	0.954054	0.943850	0.948925	374
accuracy			0.972954	1405
macro avg	0.966882	0.963681	0.965266	1405
weighted avg	0.972881	0.972954	0.972907	1405

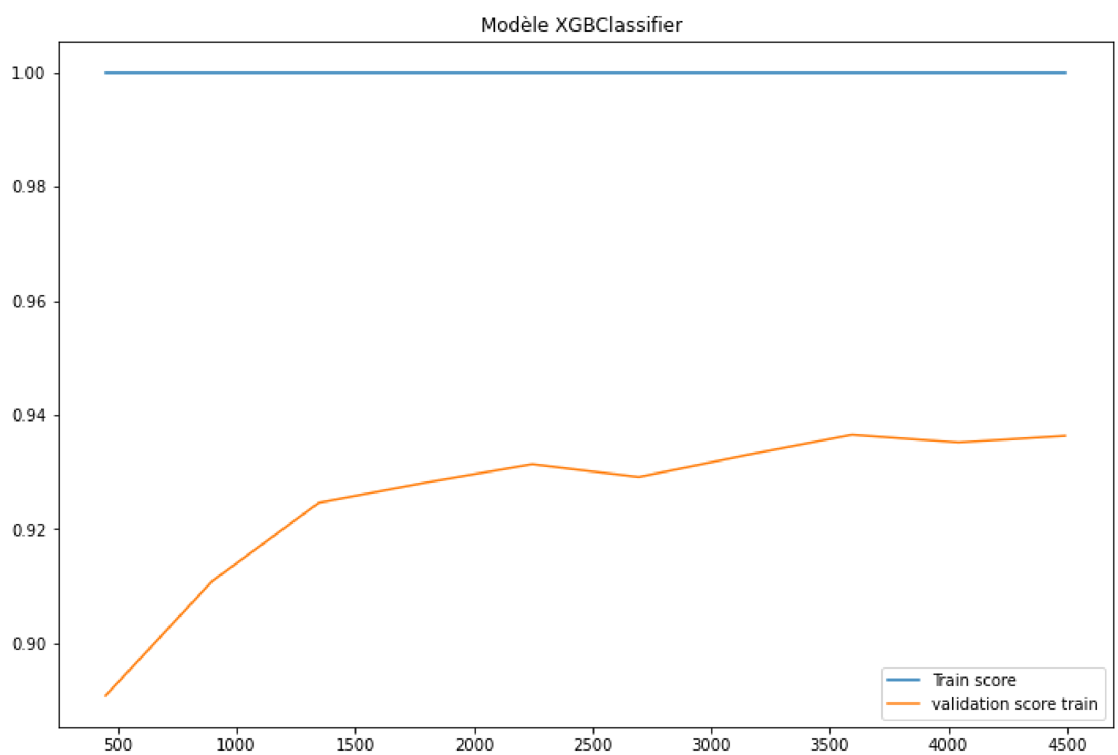
<Figure size 432x288 with 0 Axes>



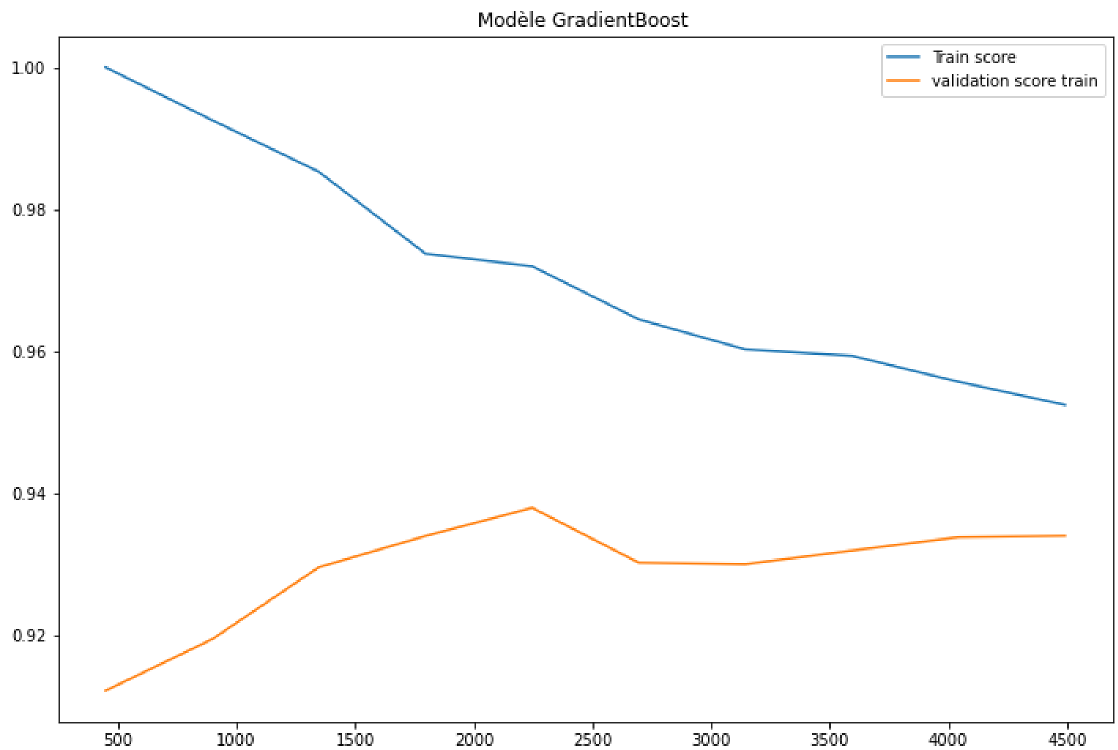

```
modele LGBMClassifier
              precision    recall  f1-score   support

    No churn   0.983463  0.980601  0.982030      1031
       Churn   0.946950  0.954545  0.950732       374

    accuracy                       0.973665      1405
   macro avg   0.965206  0.967573  0.966381      1405
weighted avg   0.973743  0.973665  0.973699      1405

<Figure size 432x288 with 0 Axes>
```

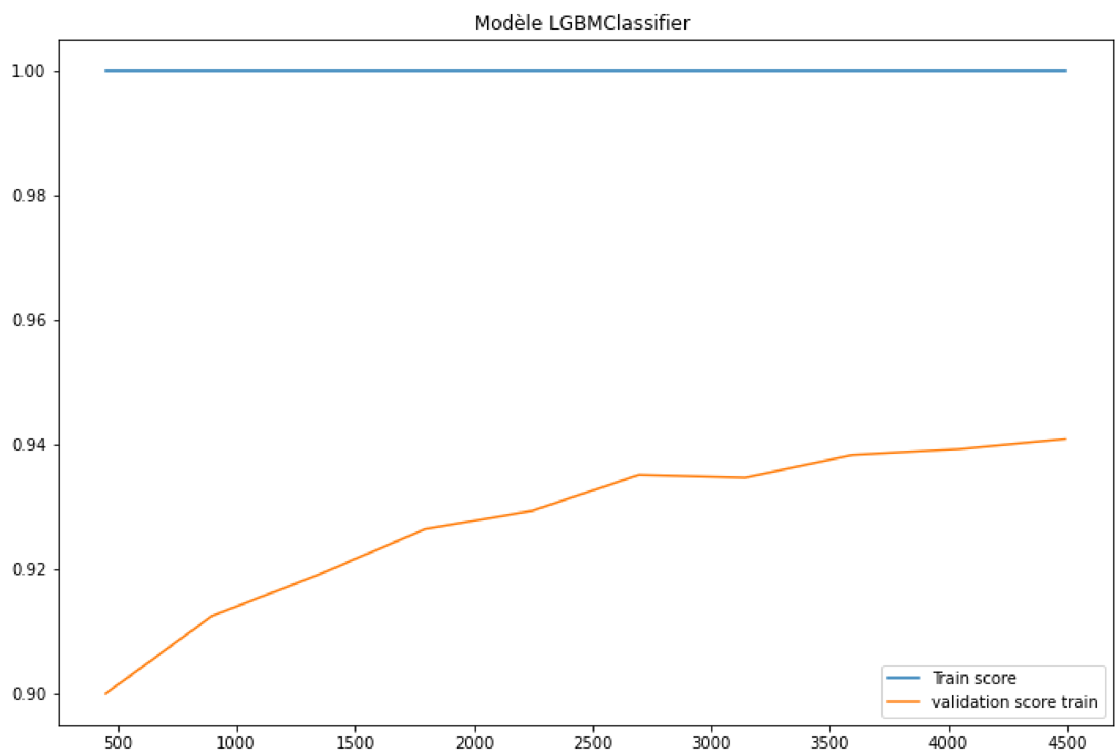


No churn
Churn
1011
20
17
357
True label
No churn
Churn
Predicted label
1000
800
600
400
200



Modèle XGBClassifier
1.00
0.98
0.96
0.94
0.92
0.90
500
1000
1500
2000
2500
3000
3500
4000
4500
Train score
validation score train


Modèle GradientBoost
Train score
validation score train
1.00
0.98
0.96
0.94
0.92
500
1000
1500
2000
2500
3000
3500
4000
4500


Modèle LGBMClassifier
Train score
validation score train
1.00
0.98
0.96
0.94
0.92
0.90
500
1000
1500
2000
2500
3000
3500
4000
4500

1.0.1 Example de réglage des hyper paramètres (cas du GradientBoostingClassifier)

réglage du n_estimators

[439]:
```
paramsGB1={"n_estimators":range(290,321,10)}
model=GradientBoostingClassifier(learning_rate=0.2,␣
↪min_samples_split=35,min_samples_leaf=3,max_depth=8,max_features='sqrt',subsample=0.
↪8,random_state=0)
grid1=GridSearchCV(model, param_grid = paramsGB1, scoring='recall',n_jobs=4,␣
↪cv=5)
grid1.fit(x_train,y_train)
print( grid1.best_params_, grid1.best_score_)
```

```
{'n_estimators': 310} 0.94247491638796
```

réglage du max_depth et min_samples_split

[440]:
```
paramsGB1={"max_depth":range(3,6,1),'min_samples_split':range(420,451,10)}
model=GradientBoostingClassifier(learning_rate=0.2,n_estimators=310,␣
↪min_samples_split=35,min_samples_leaf=3,max_features='sqrt',subsample=0.
↪8,random_state=0)
grid1=GridSearchCV(model, param_grid = paramsGB1, scoring='recall',n_jobs=4,␣
↪cv=5)
grid1.fit(x_train,y_train)
print( grid1.best_params_, grid1.best_score_)
```

```
{'max_depth': 5, 'min_samples_split': 430} 0.9511705685618729
```

réglage du min_samples_leaf et max_features

[441]:
```
paramsGB1={'min_samples_leaf':range(29,40,1),'max_features':range(7,12,2)}
model=GradientBoostingClassifier(learning_rate=0.2,n_estimators=310,␣
↪min_samples_split=430,max_features='sqrt',subsample=0.8,random_state=0)
grid1=GridSearchCV(model, param_grid = paramsGB1, scoring='recall',n_jobs=4,␣
↪cv=5)
grid1.fit(x_train,y_train)
print( grid1.best_params_, grid1.best_score_)
```

```
{'max_features': 9, 'min_samples_leaf': 31} 0.9464882943143813
```

réglage du subsample

[442]:
```
paramsGB1={'subsample':[0.6,0.7,0.75,0.8,0.85,0.9]}
model=GradientBoostingClassifier(learning_rate=0.
↪2,n_estimators=310,max_depth=5,min_samples_split=430,min_samples_leaf=32,max_features=8␣
↪,random_state=0)
grid1=GridSearchCV(model, param_grid = paramsGB1, scoring='recall',n_jobs=4,␣
↪cv=5)
grid1.fit(x_train,y_train)
print( grid1.best_params_, grid1.best_score_)
```

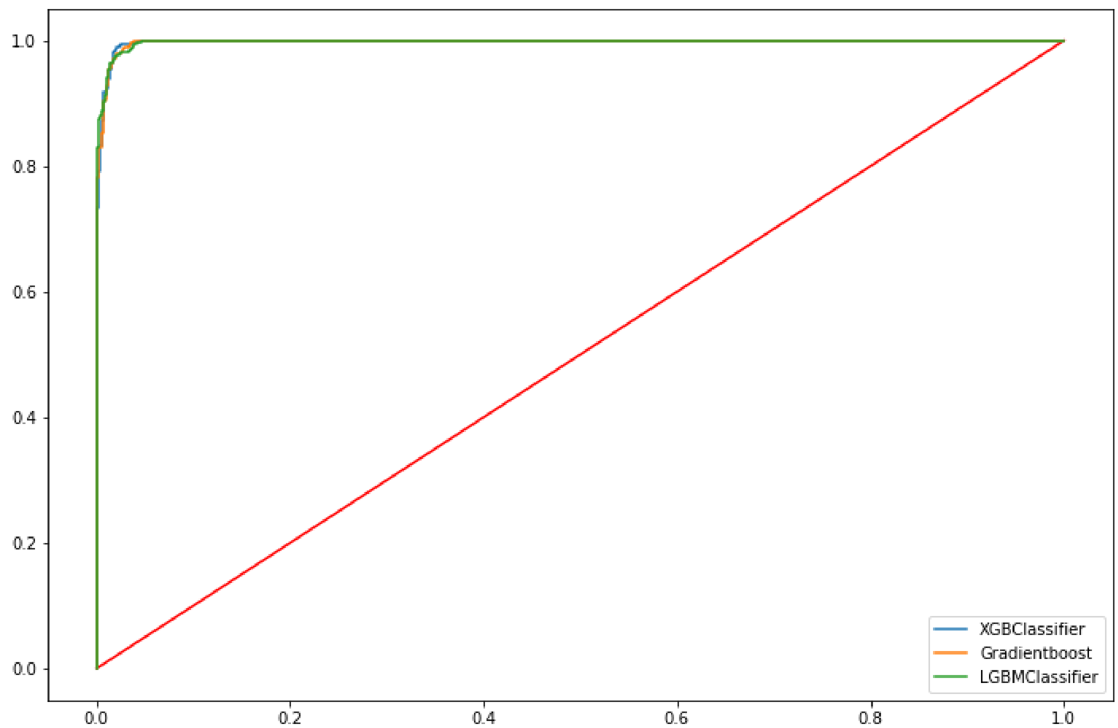
```
{'subsample': 0.8} 0.9478260869565218
```

Comparaison des modèles améliorés

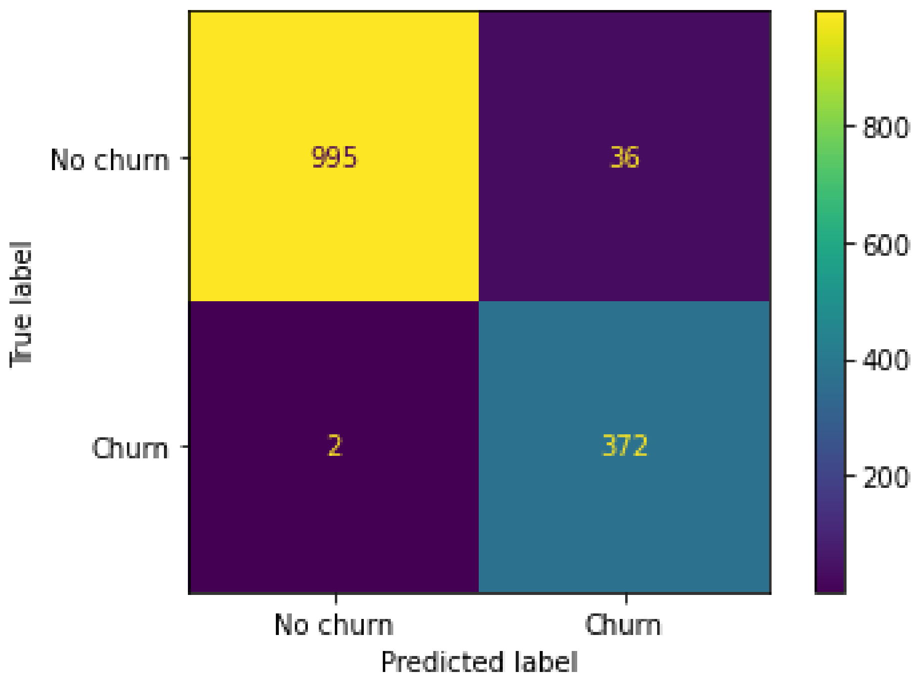
Nous remarquons que la plupart des modèles ont besoin de plus de données pour devenir plus performant

[443]:
```
modelLGBM = LGBMClassifier(num_leaves=17,max_depth=10,learning_rate=0.
↪1,n_estimators=32,subsample_for_bin=200000,min_split_gain=0.
↪0,min_child_weight=0.001,min_child_samples=1,subsample=0.
↪6,subsample_freq=0,colsample_bytree=1.
↪0,reg_alpha=1e-08,reg_lambda=1e-08,random_state=0)
modelGB = GradientBoostingClassifier(learning_rate=0.
↪2,n_estimators=310,max_depth=5,min_samples_split=430,min_samples_leaf=32,max_features=7␣
↪,random_state=0)
modelRF =␣
↪RandomForestClassifier(n_estimators=190,max_depth=7,min_samples_split=29,max_leaf_nodes=30,m
↪7,max_features=19,random_state=0)
modelXGB = XGBClassifier(n_estimators=250,learning_rate=0.
↪1,max_depth=5,min_child_weight=0.001,gamma = 0.
↪3,subsample=1,colsample_bytree = 0.83,reg_alpha=1e-5,scale_pos_weight=600)
```

[454]:
```
modelList={"XGBClassifier":modelXGB,"Gradientboost":modelGB,"LGBMClassifier":
↪modelLGBM}
models=evaluation(x_train,y_train,x_test,y_test,modelList,True,learning_cu=True)
```



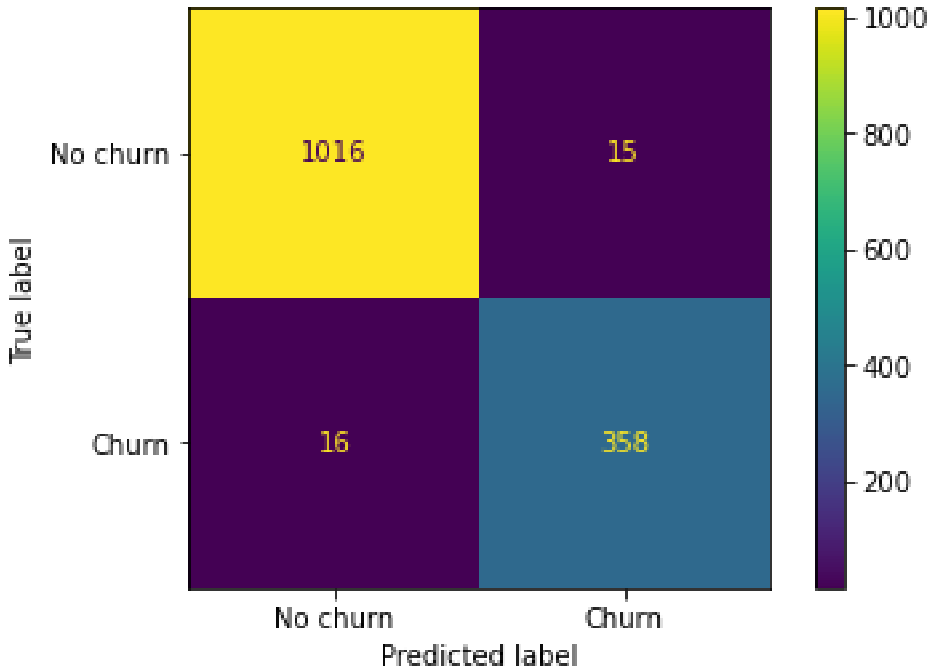
```
l aire du modele LGBMClassifier est 0.9978293230703797
l aire du modele XGBClassifier est 0.9977229936150459
l aire du modele Gradientboost est 0.997650378377257
modele XGBClassifier
              precision    recall  f1-score   support

    No churn   0.997994  0.965082  0.981262      1031
       Churn   0.911765  0.994652  0.951407       374

    accuracy                       0.972954      1405
   macro avg   0.954879  0.979867  0.966334      1405
weighted avg   0.975040  0.972954  0.973315      1405

<Figure size 432x288 with 0 Axes>
```




```
modele Gradientboost
              precision    recall  f1-score   support

    No churn   0.984496  0.985451  0.984973      1031
       Churn   0.959786  0.957219  0.958501       374

    accuracy                       0.977936      1405
   macro avg   0.972141  0.971335  0.971737      1405
```

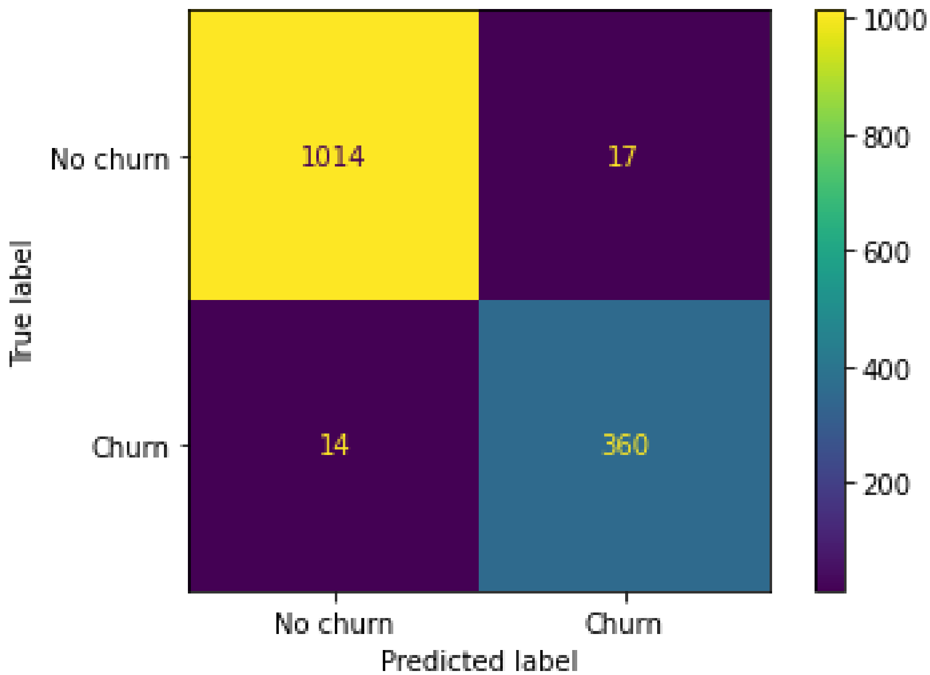
```
weighted avg   0.977918  0.977936  0.977927      1405
```

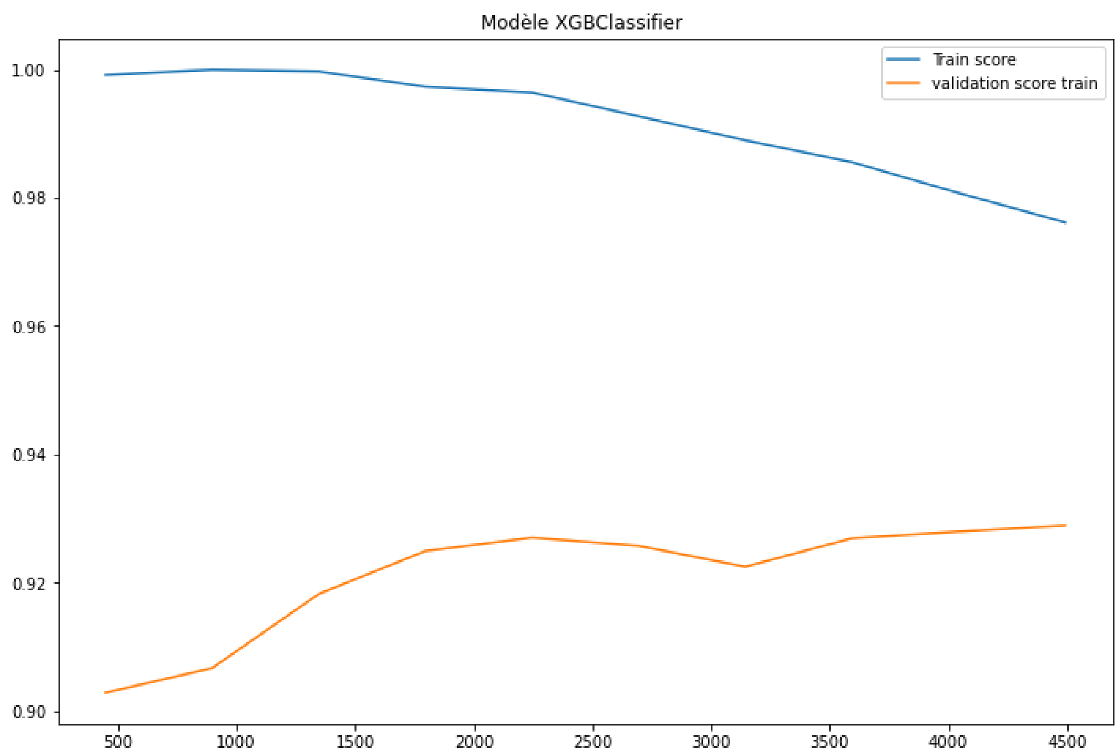
<Figure size 432x288 with 0 Axes>



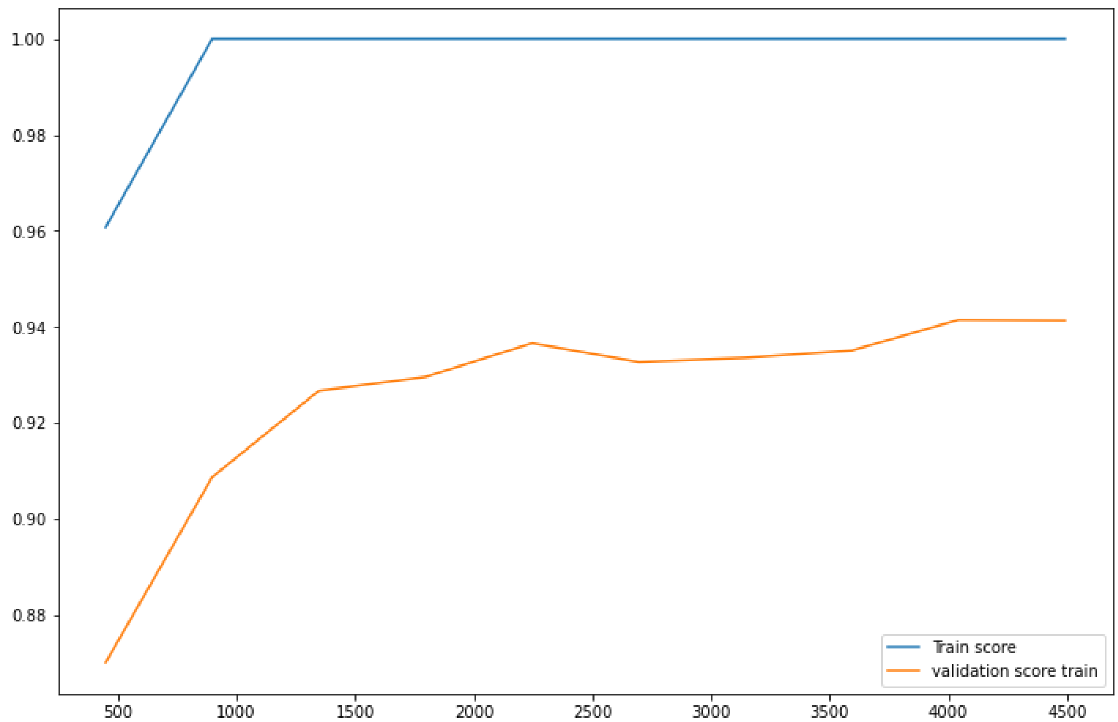

```
modele LGBMClassifier
              precision    recall  f1-score   support

    No churn   0.986381  0.983511  0.984944      1031
       Churn   0.954907  0.962567  0.958722       374

    accuracy                       0.977936      1405
   macro avg   0.970644  0.973039  0.971833      1405
weighted avg   0.978003  0.977936  0.977964      1405
```

<Figure size 432x288 with 0 Axes>

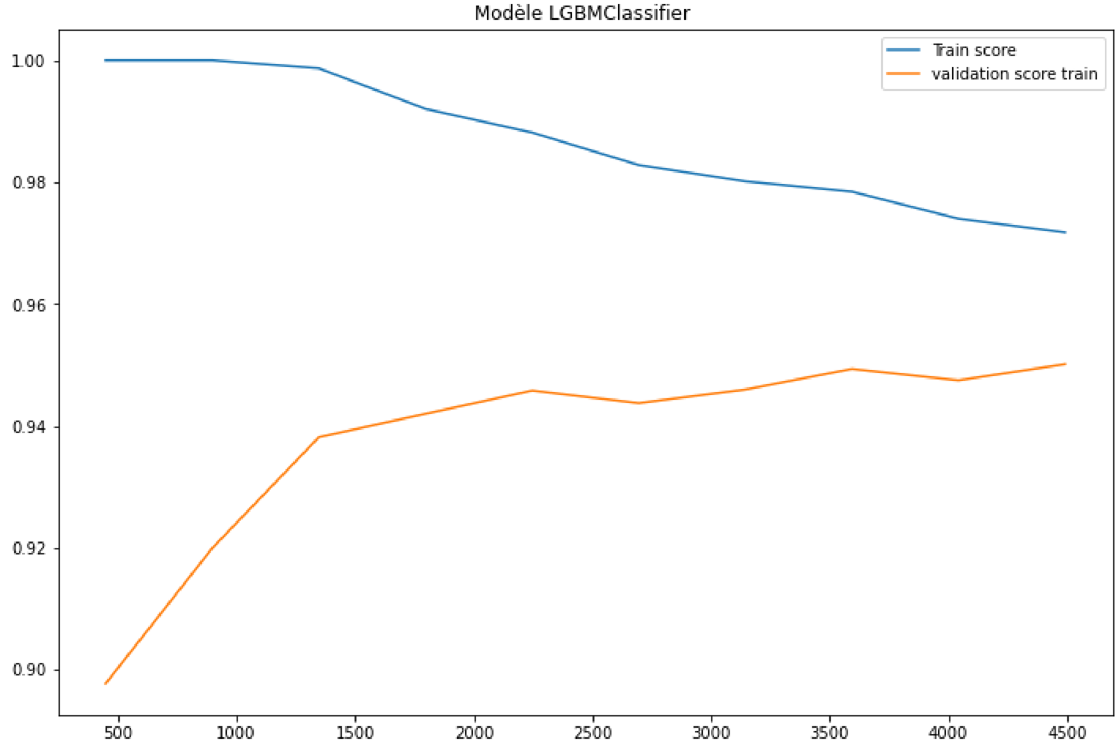

No churn
Churn
True label
1014
17
14
360
No churn
Churn
Predicted label
1000
800
600
400
200


Modèle XGBClassifier
Train score
validation score train
1.00
0.98
0.96
0.94
0.92
0.90
500
1000
1500
2000
2500
3000
3500
4000
4500

Modèle Gradientboost



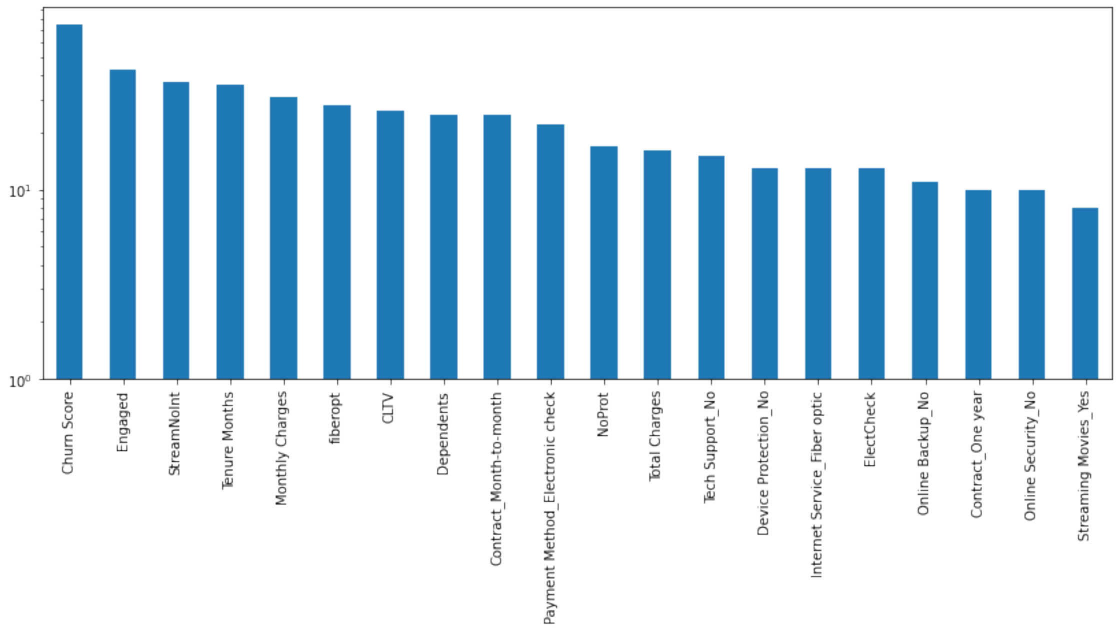
Modèle LGBMClassifier



[455]:
```
best_model=models["LGBMClassifier"]
```

Features importances du modéle LGBM

[456]:
```
featuresImportances(best_model,features.columns,20)
```

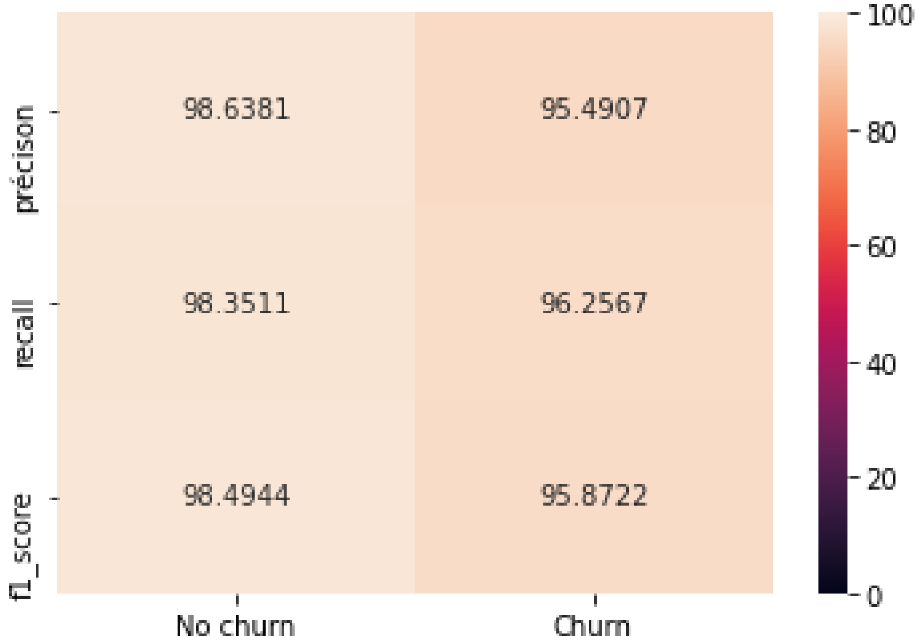


Résultats finaux

Les 2 meilleurs modèles sont LGBMClassifier et GradientBoostingClassifier

LGBMClassifier

[457]:
```
precision_recall_f1(best_model)
```



GradientBoostingClassifier

[458]:
```
precision_recall_f1(models["Gradientboost"])
```

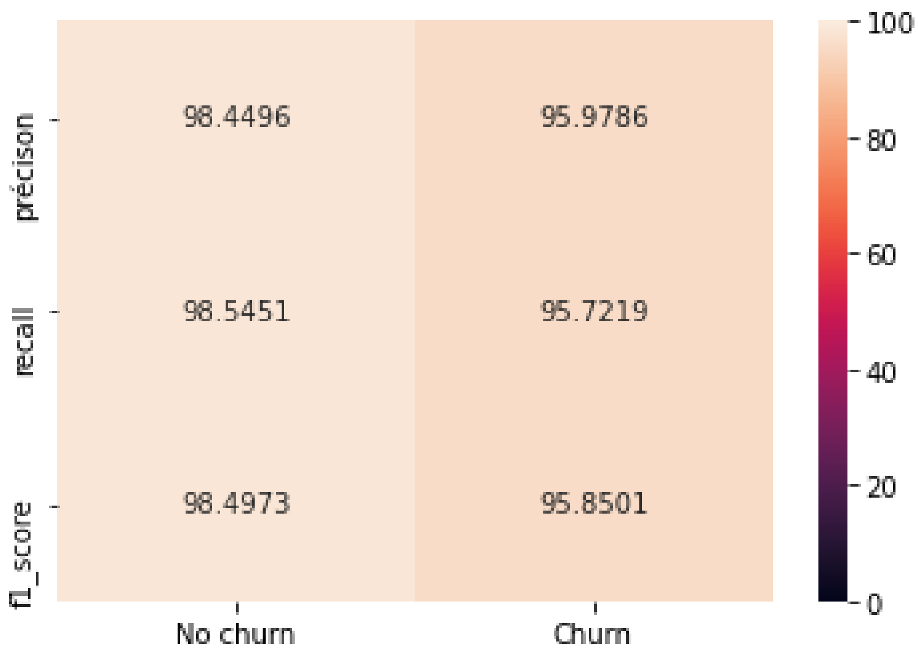


Tableau comparatif

[486]:
```
models["Regression Logist"]=regressionL["Regression Logistique"]
models["RamdomForest"]=randomF["Random Forest"]
models["NaiveBayes"]=NaiveBayes["NaiveBayes"]
models
```

[486]:
```
{'XGBClassifier': XGBClassifier(base_score=0.5, booster='gbtree',
colsample_bylevel=1,
               colsample_bynode=1, colsample_bytree=0.83, gamma=0.3, gpu_id=-1,
               importance_type='gain', interaction_constraints='',
               learning_rate=0.1, max_delta_step=0, max_depth=5,
               min_child_weight=0.001, missing=nan, monotone_constraints='()',
               n_estimators=250, n_jobs=0, num_parallel_tree=1, random_state=0,
               reg_alpha=1e-05, reg_lambda=1, scale_pos_weight=600, subsample=1,
               tree_method='exact', validate_parameters=1, verbosity=None),
 'Gradientboost': GradientBoostingClassifier(learning_rate=0.2, max_depth=5,
max_features=7,
                            min_samples_leaf=32, min_samples_split=430,
                            n_estimators=310, random_state=0),
 'LGBMClassifier': LGBMClassifier(max_depth=10, min_child_samples=1,
n_estimators=32,
                num_leaves=17, random_state=0, reg_alpha=1e-08,
reg_lambda=1e-08,
                subsample=0.6),
 'Regression Logist': LogisticRegression(C=0.20202, random_state=0,
solver='liblinear', tol=0.2),
 'RamdomForest': RandomForestClassifier(random_state=0),
 'NaiveBayes': GaussianNB()}
```

[658]:
```
test_set={
    "XGBClassifier":[x_test,y_test],
    "Gradientboost":[x_test,y_test],
    "LGBMClassifier":[x_test,y_test],
    "Regression Logist":[X_testL,y_testL],
    "RamdomForest":[X_testR,y_testR],
    "NaiveBayes":[X_testN,y_testN],
}
comparaison_models(models,test_set)
```

```
+-------------------+------------+----------+------------+------------+---------
----+----------+
| Modèles           |   précison |   recall |   f1_score |   accuracy |
test_size | Etat     |
+===================+============+==========+============+============+=========
====+==========+
| XGBClassifier     |    99.7994 |  96.5082 |    98.1262 |    97.2954 |
1031 | No churn |
```

XGBClassifier	91.1765	99.4652	95.1407	97.2954	374	Churn
Gradientboost	98.4496	98.5451	98.4973	97.7936	1031	No churn
Gradientboost	95.9786	95.7219	95.8501	97.7936	374	Churn
LGBMClassifier	98.6381	98.3511	98.4944	97.7936	1031	No churn
LGBMClassifier	95.4907	96.2567	95.8722	97.7936	374	Churn
Regression Logist	94.7469	96.031	95.3846	93.177	1033	No churn
Regression Logist	88.6111	85.2941	86.921	93.177	374	Churn
RamdomForest	96.5193	99.3224	97.9008	96.8728	1033	No churn
RamdomForest	97.9651	90.107	93.8719	96.8728	374	Churn
NaiveBayes	94.1468	91.8683	92.9936	89.8365	1033	No churn
NaiveBayes	78.9474	84.2246	81.5006	89.8365	374	Churn

```
[ ]:
```

```
[ ]:
```