Telecom Churn

Python Project

Contributor:
Mohamed Ibrahim

2024

TABLE OF CONTENTS

Introduction

This project uses the Telecom Churn Dataset. It contains parameters for subscribers of a telecommunications network.

The goal is to analyze network data and build a model that predicts which subscribers are likely to churn. These predictions can help the management of this network to make decisions, minimize the cause(s) of churn and increase the loyalty of subscribers. In general, the cost of acquiring a new subscriber is much more than the cost of keeping an existing one.

It is supervised learning with the response variable in the "Churn" column. The language used is Python.

The GitHub directory for this project is https://github.com/moibrahim2021/Telecom-churn.

Data Exploration

There are two data files:

- churn-bigml-80.csv that contains training data.
- churn-bigml-20.csv that contains the test data.

```
import pandas as pd
abonnes = pd.read_csv(r"/content/churn-bigml-80.csv")
abonnes.head().transpose()
```

	0	**1**	**2**	**3**	**4**
State	KS	OH	NJ	OH	OK
Account length	128	107	137	84	75
Area code	415	415	415	408	415
International plan	No	No	No	Yes	Yes
Voice mail plan	Yes	Yes	No	No	No
Number vmail messages	25	26	0	0	0
Total day minutes	265.1	161.6	243.4	299.4	166.7
Total day calls	110	123	114	71	113
Total day charge	45.07	27.47	41.38	50.9	28.34
Total eve minutes	197.4	195.5	121.2	61.9	148.3
Total eve calls	99	103	110	88	122
Total eve charge	16.78	16.62	10.3	5.26	12.61
Total night minutes	244.7	254.4	162.6	196.9	186.9
Total night calls	91	103	104	89	121
Total night charge	11.01	11.45	7.32	8.86	8.41
Total intl minutes	10.0	13.7	12.2	6.6	10.1
Total intl calls	3	3	5	7	3
Total intl charge	2.7	3.7	3.29	1.78	2.73

	0	1	2	3	4
Customer service calls	1	1	0	2	3
Churn	False	False	False	False	False

The above table is transposed for visibility purpose because it contains 20 attributes. In its original form, each row of the dataset contains the attributes of one subscriber.

Explanation of the attributes

1. State: the abbreviation of the name of the state (in the United States) where the subscriber resides
2. Account length: the number of days the account is active
3. Area code: the area code of the subscriber's telephone number
4. International plan: if the subscriber has an international calling plan
5. Voice mail plan: if the subscriber has the voicemail service
6. Number vmail messages: the average number of voicemails per month
7. Total day minutes: the total number of call minutes used during the day
8. Total day calls: the total number of calls made during the day
9. Total day charge: the cost of the day's calls
10. Total eve minutes: the total number of call minutes used during the evening
11. Total eve calls: the total number of calls made during the evening
12. Total eve charge: the cost for the evening's calls
13. Total night minutes: the total number of call minutes used during the night
14. Total night calls: the total number of calls made during the night
15. Total night charge: the cost of the night's calls
16. Total intl minutes: the total number of the international minutes
17. Total intl calls: the total number of the international calls
18. Total intl charge: the cost of the international calls
19. Customer service calls: the number of calls made to customer service
20. Churn: If the subscriber has churned.

The 20th column/attribute (Churn) is the response variable that must be predicted.

The problem and the objective

Orange Telecom needs to minimize the number of subscribers who churn or leave the network for another. The data provided by the company is that of about 3300 subscribers, some of whom have churned. To minimize the number of churns is to minimize the cost of increasing the number of subscribers and increasing the revenues.

The objective is to build a prediction model that can alert managers at Orange Telecom when one or more subscribers are at risk of churning. These managers will have a plan to execute to retain the maximum number of these subscribers.

Data import and preparation

```
import pandas as pd
# we combine the 2 files, and we create the abonnes object
abonnes = pd.concat(map(pd.read_csv, ["D:\Documents\Github\Telecom-
churn\churn-bigml-80.csv", "D:\Documents\Github\Telecom-churn\churn-bigml-
20.csv"]))
```

Data Exploration and Preparation

```
abonnes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3333 entries, 0 to 666
Data columns (total 20 columns):
#   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   State                  3333 non-null   object
 1   Account length         3333 non-null   int64
 2   Area code              3333 non-null   int64
 3   International plan     3333 non-null   object
 4   Voice mail plan        3333 non-null   object
 5   Number vmail messages  3333 non-null   int64
 6   Total day minutes      3333 non-null   float64
 7   Total day calls        3333 non-null   int64
 8   Total day charge       3333 non-null   float64
 9   Total eve minutes      3333 non-null   float64
 10  Total eve calls        3333 non-null   int64
 11  Total eve charge       3333 non-null   float64
 12  Total night minutes    3333 non-null   float64
 13  Total night calls      3333 non-null   int64
 14  Total night charge     3333 non-null   float64
 15  Total intl minutes     3333 non-null   float64
 16  Total intl calls       3333 non-null   int64
 17  Total intl charge      3333 non-null   float64
 18  Customer service calls 3333 non-null   int64
 19  Churn                  3333 non-null   bool
dtypes: bool(1), float64(8), int64(8), object(3)
memory usage: 524.0+ KB
```

The dataset contains 3333 rows (subscribers). The 20 attributes do not contain null values. The types of attributes are:

- 8 int64 attributes
- 8 float64 attributes
- 3 object attributes (text)
- 1 Boolean Attribute

```
print("Unique values:\n", abonnes.nunique())
```

```
# # # Unique values # # #
 State 51
Account length 212
Area code 3
International plan 2
```

```
Voice mail plan 2
Number vmail messages 46
Total day minutes 1667
Total day calls 119
Total day charge 1667
Total eve minutes 1611
Total eve calls 123
Total eve charge 1440
Total night minutes 1591
Total night calls 120
Total night charge 933
Total intl minutes 162
Total intl calls 21
Total intl charge 162
Customer service calls 10
Churn 2
dtype: int64
```

The table above shows each attribute contains how many unique values.

For statistical calculations, we transform 3 attributes whose types are not numeric (object or boolean) and which each has 2 unique values: International plan, Voice mail plan and Churn.

```
# Initial state of the 3 attributes
print("# # # Before transformation # # #\n")
print("International Calling Service:\n", abonnes['International plan'].value_counts(), "\n")
print("Voice Mail Service:\n", abonnes['Voice mail plan'].value_counts(), "\n")
print("Churn:\n", abonnes['Churn'].value_counts(), "\n")
```

```
# # # Before transformation # # #

International Calling Service:
 International plan
No. 3010
Yes 323
Name: count, dtype: int64

Voice Mail Service:
 Voice mail plan
No. 2411
Yes 922
Name: count, dtype: int64

Churn:
 Churn
False 2850
True 483
Name: count, dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder
# The col_bin variable contains the names of the 3 attributes
col_bin = abonnes.nunique()[abonnes.nunique() == 2].keys().tolist()

# Encode the 3 attributes in 0 or 1
encodeur = LabelEncoder()
for i in col_bin:
```

```
  abonnes[i] = encodeur.fit_transform(abonnes[i])

# Check the 3 attributes
print("# # # After transformation # # #\n")
print("International Calling Service:\n", abonnes['International plan'].value_counts(), "\n")
print("Voice Mail Service:\n", abonnes['Voice mail plan'].value_counts(), "\n")
print("Churn:\n", abonnes['Churn'].value_counts(), "\n")
print("abonnes.info() :")
print(abonnes.info())
```

```
# # # After transformation # # #

International Calling Service:
 International plan
0    3010
1     323
Name: count, dtype: int64

Voice Mail Service:
 Voice mail plan
0    2411
1     922
Name: count, dtype: int64

Churn:
 Churn
0    2850
1     483
Name: count, dtype: int64

abonnes.info() :
<class 'pandas.core.frame.DataFrame'>
Index: 3333 entries, 0 to 666
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   State                   3333 non-null   object
 1   Account length          3333 non-null   int64
 2   Area code               3333 non-null   int64
 3   International plan      3333 non-null   int32
 4   Voice mail plan         3333 non-null   int32
 5   Number vmail messages   3333 non-null   int64
 6   Total day minutes       3333 non-null   float64
 7   Total day calls         3333 non-null   int64
 8   Total day charge        3333 non-null   float64
 9   Total eve minutes       3333 non-null   float64
 10  Total eve calls         3333 non-null   int64
 11  Total eve charge        3333 non-null   float64
 12  Total night minutes     3333 non-null   float64
 13  Total night calls       3333 non-null   int64
 14  Total night charge      3333 non-null   float64
 15  Total intl minutes      3333 non-null   float64
 16  Total intl calls        3333 non-null   int64
 17  Total intl charge       3333 non-null   float64
 18  Customer service calls  3333 non-null   int64
 19  Churn                   3333 non-null   int64
dtypes: float64(8), int32(2), int64(9), object(1)
memory usage: 520.8+ KB
```

```
# Summary table of the 19 numeric columns (int or float)
sommaire = (abonnes[[i for i in abonnes.columns]].describe().transpose().
reset_index())

print("\n# # # Summary # # #\n", sommaire)
```

Summary # #

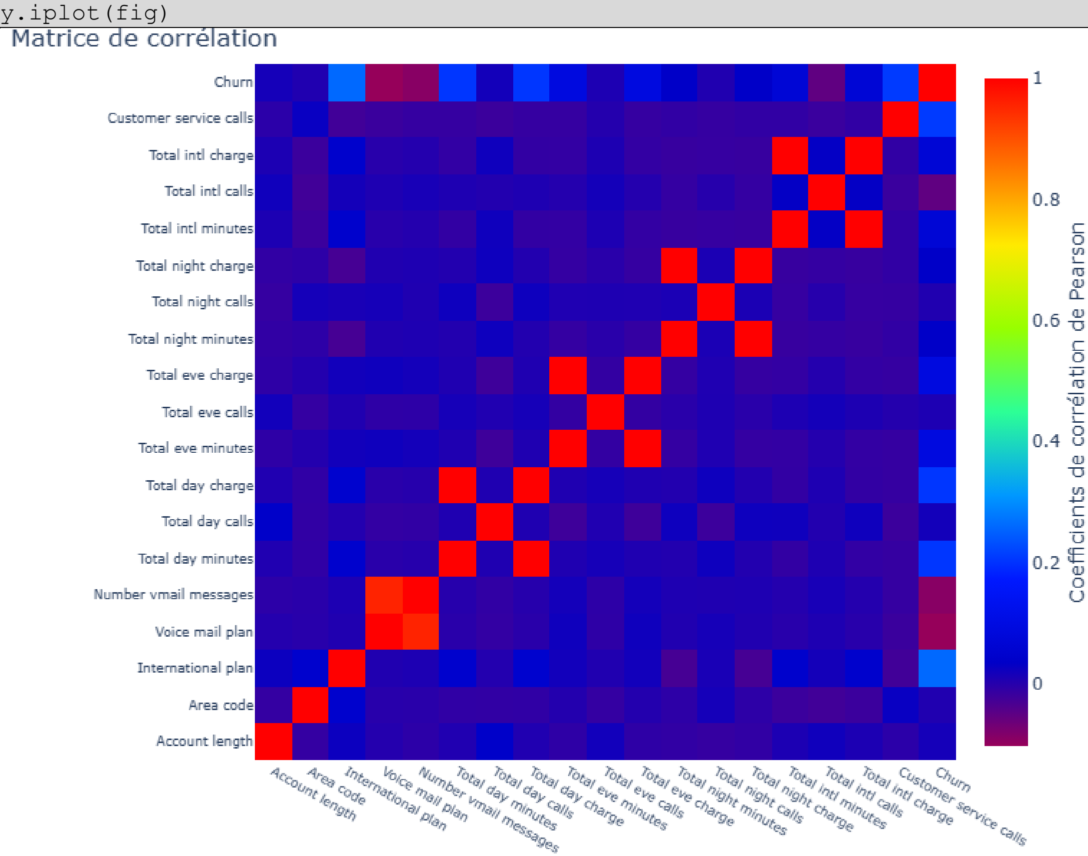
	index	count	mean	std	min	25%	50%	75%	max
0	Account length	3333.0	101.065	39.822	1.00	74.00	101.00	127.00	243.00
1	Area code	3333.0	437.182	42.371	408.00	408.00	415.00	510.00	510.00
2	International plan	3333.0	0.097	0.296	0.00	0.00	0.00	0.00	1.00
3	Voice mail plan	3333.0	0.277	0.447	0.00	0.00	0.00	1.00	1.00
4	Number vmail messages	3333.0	8.099	13.688	0.00	0.00	0.00	20.00	51.00
5	Total day minutes	3333.0	179.775	54.467	0.00	143.70	179.40	216.40	350.80
6	Total day calls	3333.0	100.436	20.069	0.00	87.00	101.00	114.00	165.00
7	Total day charge	3333.0	30.562	9.259	0.00	24.43	30.50	36.79	59.64
8	Total eve minutes	3333.0	200.980	50.714	0.00	166.60	201.40	235.30	363.70
9	Total eve calls	3333.0	100.114	19.923	0.00	87.00	100.00	114.00	170.00
10	Total eve charge	3333.0	17.084	4.311	0.00	14.16	17.12	20.00	30.91
11	Total night minutes	3333.0	200.872	50.574	23.20	167.00	201.20	235.30	395.00
12	Total night calls	3333.0	100.108	19.569	33.00	87.00	100.00	113.00	175.00
13	Total night charge	3333.0	9.039	2.276	1.04	7.52	9.05	10.59	17.77
14	Total intl minutes	3333.0	10.237	2.792	0.00	8.50	10.30	12.10	20.00
15	Total intl calls	3333.0	4.479	2.461	0.00	3.00	4.00	6.00	20.00
16	Total intl charge	3333.0	2.765	0.754	0.00	2.30	2.78	3.27	5.40
17	Customer service calls	3333.0	1.563	1.315	0.00	1.00	1.00	2.00	9.00
18	Churn	3333.0	0.145	0.352	0.00	0.00	0.00	0.00	1.00

The correlation between the attributes

The Pearson correlation matrix (heatmap)

```
import plotly.graph_objs as go
```

```
import plotly.offline as py
import numpy as np

numerics = ['int32', 'int64', 'float64']
abonnes_num = abonnes.select_dtypes(include=numerics)
correlation = abonnes_num.corr()
col_mat = correlation.columns.tolist()
corr_array = np.array(correlation) # convert to array
trace = go.Heatmap(z = corr_array,
                   x = col_mat,
                   y = col_mat,
                   colorscale = "rainbow",
                   colorbar = dict(title = "Coefficients de corrélation de
Pearson", titleside = "right"),
                  )
layout = go.Layout(dict(title = "Matrice de corrélation",
                        autosize = False,
                        height = 720,
                        width = 800,
                        margin = dict(r = 0, l = 210, t = 25, b = 210),
                        yaxis = dict(tickfont = dict(size = 9)),
                        xaxis = dict(tickfont = dict(size = 9))
                       )
                  )
data = [trace]
fig = go.Figure(data=data, layout=layout)
py.iplot(fig)
```



This correlation matrix shows that the attribute 'Churn' has a low correlation with all attributes.

It also shows that there are 5 pairs of attributes that are strongly correlated:

1. Total intl minutes and Total intl charge
2. Total night minutes and Total night charge
3. Total eve minutes and Total eve charged
4. Total day minutes and Total day charge
5. Voice mail plan and Number vmail messages

So, we keep the first attribute of each pair only during the modeling. We create the 'supprimer' list which contains the names of the attributes we are going to remove.

```
supprimer = ['Total intl charge', 'Total night charge', 'Total eve charge',
'Total day charge', 'Number vmail messages']
```

The Pearson correlation of 'Churn'

```
# Correlation between 'Churn' and other attributes
print("\n# # # # Pearson's correlation with Churn # # #")
abonnes_num = abonnes_num.drop(supprimer, axis=1)
abonnes.corrwith(abonnes.iloc[:,19])
```

```
# # # Pearson's correlation with Churn # # #
Account length            0.016541
Area code                 0.006174
International plan        0.259852
Voice mail plan          -0.102148
Total day minutes         0.205151
Total day calls           0.018459
Total eve minutes         0.092796
Total eve calls           0.009233
Total night minutes       0.035493
Total night calls         0.006141
Total intl minutes        0.068239
Total intl calls         -0.052844
Customer service calls    0.208750
Churn                     1.000000
dtype: float64
```

This table shows that there is a weak correlation between 'Churn' and all attributes.

The rank correlation of 'Churn' (Spearman)

```
print("\n# # # Rank correlation (Spearman) with Churn # # #")
abonnes.corrwith(abonnes.iloc[:,19], method='spearman')
```

```
# # # Rank correlation (Spearman) with Churn # # #
Account length            0.015583
Area code                 0.003257
International plan        0.259852
Voice mail plan          -0.102148
Total day minutes         0.170677
Total day calls           0.026311
Total eve minutes         0.088592
Total eve calls           0.008578
Total night minutes       0.034343
Total night calls         0.004694
Total intl minutes        0.060850
Total intl calls         -0.074758
```

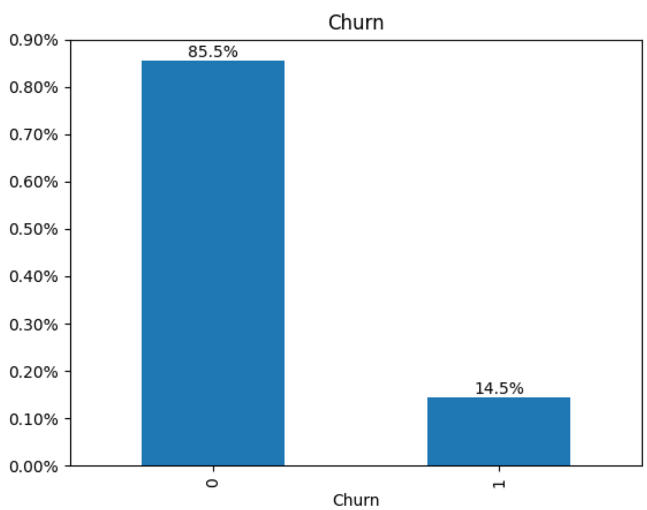
```
Customer service calls     0.136657
Churn                      1.000000
dtype: float64
```

Like Pearson's correlation, there is a weak correlation between 'Churn' and all attributes.

The 'Churn' response attribute

Viewing 'Churn'

```
import matplotlib.ticker as mtick

nbr_churn = abonnes['Churn'].value_counts()
nbr_churn[0] = round(nbr_churn[0]/abonnes.Churn.count(), 6)
nbr_churn[1] = round(nbr_churn[1]/abonnes.Churn.count(), 6)
ax = nbr_churn.plot.bar(x=nbr_churn, )
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.set_title('Churn')
ax.set_ylim((0,0.9))
for c in ax.containers:
  ratios = c.datavalues / c.datavalues.sum()
  ax.bar_label(c, labels=[f'{r:.1%}' for r in ratios])
```

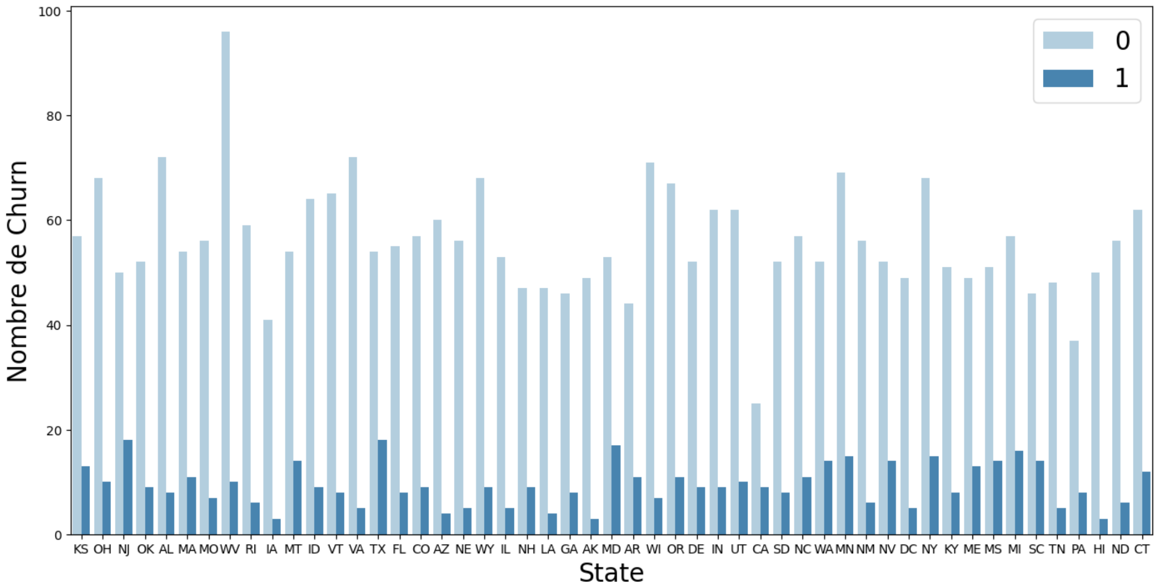


This figure shows that the 2 values of the 'Churn' attribute are not balanced.

The 'State' attribute

Visualization of 'State' vs 'Churn'

```
import matplotlib.pyplot as plt
import seaborn as sns
fig, axz = plt.subplots(figsize=(15,15))
axz = sns.countplot(x='State', hue='Churn', data=abonnes, palette='Blues')
axz.set_ylabel('Nombre de Churn', size=20)
axz.set_xlabel('State', size=20)
axz.legend(loc=0, fontsize=20);
```



Distribution of Churn=1 by 'State'

```
# Filter subscribers with Churn = 1 (True)
abonnes_churn = abonnes[abonnes.Churn == 1]
# Number of abonnes_churn per State
print("# # # Number of subscribers per State with Churn=1 # # #\n", abonnes_churn.State.value_counts())
```

```
# # # Number of subscribers per State with Churn=1 # # #
 State
NJ    18
TX    18
MD    17
MI    16
MN    15
NY    15
MT    14
MS    14
NV    14
SC    14
WA    14
ME    13
KS    13
CT    12
NC    11
MA    11
OR    11
AR    11
OH    10
WV    10
UT    10
CA     9
DE     9
CO     9
OK     9
NH     9
IN     9
ID     9
WY     9
```

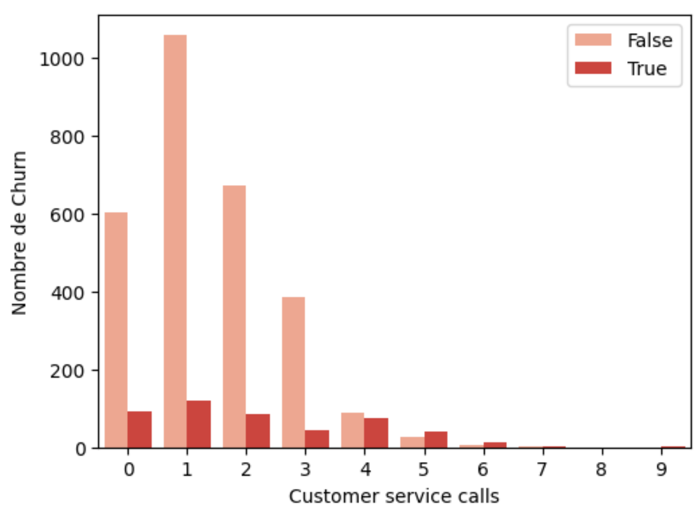
```
GA      8
AL      8
SD      8
FL      8
VT      8
KY      8
PA      8
WI      7
MO      7
NM      6
RI      6
ND      6
NE      5
IL      5
TN      5
VA      5
DC      5
LA      4
AZ      4
IA      3
AK      3
HI      3
Name: count, dtype: int64
```

The states with the highest numbers of churned subscribers are: New Jersey, Texas, Maryland, Michigan, Minnesota and New York.

The 'Customer service calls' attribute

Visualization of 'Customer service calls' vs 'Churn'

```
fig, axz = plt.subplots(figsize=(5,3))
axz = sns.countplot(x='Customer service calls', hue='Churn', data=abonnes, palette='Reds')
axz.set_ylabel('Nombre de Churn', size=10)
axz.set_xlabel('Customer service calls', size=10)
axz.legend(loc=0, fontsize=10, labels=['False', 'True']);
```



This figure shows that there are subscribers who have churned after a single call to customer service, and even there are some who have not called it at all.

Importing the 2 separate files

```
# Training file
train = pd.read_csv("D:\Documents\Github\Telecom-churn\churn-bigml-80.csv")
print("\n# # # Training # # #\n")
print(train.info())
print("\nUnique values:\n", train.nunique())
col_bin = train.nunique()[train.nunique() == 2].keys().tolist()
# Encode the 3 attributes in 0 or 1
encodeur = LabelEncoder()
for i in col_bin:
  train[i] = encodeur.fit_transform(train[i])
print("\nAfter the transformation of Training:")
print(train.info())
```

```
# # # Training # # #

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2666 entries, 0 to 2665
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   State                   2666 non-null   object
 1   Account length          2666 non-null   int64
 2   Area code               2666 non-null   int64
 3   International plan      2666 non-null   object
 4   Voice mail plan         2666 non-null   object
 5   Number vmail messages   2666 non-null   int64
 6   Total day minutes       2666 non-null   float64
 7   Total day calls         2666 non-null   int64
 8   Total day charge        2666 non-null   float64
 9   Total eve minutes       2666 non-null   float64
 10  Total eve calls         2666 non-null   int64
 11  Total eve charge        2666 non-null   float64
 12  Total night minutes     2666 non-null   float64
 13  Total night calls       2666 non-null   int64
 14  Total night charge      2666 non-null   float64
 15  Total intl minutes      2666 non-null   float64
 16  Total intl calls        2666 non-null   int64
 17  Total intl charge       2666 non-null   float64
 18  Customer service calls  2666 non-null   int64
 19  Churn                   2666 non-null   bool
dtypes: bool(1), float64(8), int64(8), object(3)
memory usage: 398.5+ KB
None

Unique values:
 State                      51
Account length            205
Area code                   3
International plan          2
Voice mail plan             2
Number vmail messages      42
Total day minutes        1489
Total day calls           115
```

```
Total day charge         1489
Total eve minutes        1442
Total eve calls           120
Total eve charge         1301
Total night minutes      1444
Total night calls         118
Total night charge        885
Total intl minutes        158
Total intl calls           21
Total intl charge         158
Customer service calls     10
Churn                       2
dtype: int64

After the transformation of Training:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2666 entries, 0 to 2665
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   State                   2666 non-null   object
 1   Account length          2666 non-null   int64
 2   Area code               2666 non-null   int64
 3   International plan      2666 non-null   int32
 4   Voice mail plan         2666 non-null   int32
 5   Number vmail messages   2666 non-null   int64
 6   Total day minutes       2666 non-null   float64
 7   Total day calls         2666 non-null   int64
 8   Total day charge        2666 non-null   float64
 9   Total eve minutes       2666 non-null   float64
 10  Total eve calls         2666 non-null   int64
 11  Total eve charge        2666 non-null   float64
 12  Total night minutes     2666 non-null   float64
 13  Total night calls       2666 non-null   int64
 14  Total night charge      2666 non-null   float64
 15  Total intl minutes      2666 non-null   float64
 16  Total intl calls        2666 non-null   int64
 17  Total intl charge       2666 non-null   float64
 18  Customer service calls  2666 non-null   int64
 19  Churn                   2666 non-null   int64
dtypes: float64(8), int32(2), int64(9), object(1)
memory usage: 395.9+ KB
None
```

The training object contains 2666 rows (subscribers). The 20 attributes do not contain null values. After the transformation, the types of the attributes are:

- 9 attributes are int64
- 8 attributes are float64
- 2 attributes are int32
- 1 attribute is object

```
# Test file
test = pd.read_csv("D:\Documents\Github\Telecom-churn\churn-bigml-20.csv")
print("\n# # # Test # # #\n")
```

```
print(test.info())
print("Unique values:\n", test.nunique())
col_bin = test.nunique()[test.nunique() == 2].keys().tolist()
# Encode the 3 attributes in 0 or 1
encodeur = LabelEncoder()
for i in col_bin:
  test[i] = encodeur.fit_transform(test[i])
print("\nAfter the Test Transformation:")
print(test.info())
```

```
# # # Test # # #

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 667 entries, 0 to 666
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   State                   667 non-null    object
 1   Account length          667 non-null    int64
 2   Area code               667 non-null    int64
 3   International plan      667 non-null    object
 4   Voice mail plan         667 non-null    object
 5   Number vmail messages   667 non-null    int64
 6   Total day minutes       667 non-null    float64
 7   Total day calls         667 non-null    int64
 8   Total day charge        667 non-null    float64
 9   Total eve minutes       667 non-null    float64
 10  Total eve calls         667 non-null    int64
 11  Total eve charge        667 non-null    float64
 12  Total night minutes     667 non-null    float64
 13  Total night calls       667 non-null    int64
 14  Total night charge      667 non-null    float64
 15  Total intl minutes      667 non-null    float64
 16  Total intl calls        667 non-null    int64
 17  Total intl charge       667 non-null    float64
 18  Customer service calls  667 non-null    int64
 19  Churn                   667 non-null    bool
dtypes: bool(1), float64(8), int64(8), object(3)
memory usage: 99.8+ KB
None
Unique values:
 State                     51
Account length           179
Area code                  3
International plan         2
Voice mail plan            2
Number vmail messages     37
Total day minutes        562
Total day calls          100
Total day charge         562
Total eve minutes        557
Total eve calls           94
Total eve charge         528
Total night minutes      568
Total night calls         96
Total night charge       453
Total intl minutes       132
Total intl calls          17
Total intl charge        132
Customer service calls     9
```

```
Churn                      2
dtype: int64

After the transformation of Test:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 667 entries, 0 to 666
Data columns (total 20 columns):
#   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   State                   667 non-null    object
 1   Account length          667 non-null    int64
 2   Area code               667 non-null    int64
 3   International plan      667 non-null    int32
 4   Voice mail plan         667 non-null    int32
 5   Number vmail messages   667 non-null    int64
 6   Total day minutes       667 non-null    float64
 7   Total day calls         667 non-null    int64
 8   Total day charge        667 non-null    float64
 9   Total eve minutes       667 non-null    float64
 10  Total eve calls         667 non-null    int64
 11  Total eve charge        667 non-null    float64
 12  Total night minutes     667 non-null    float64
 13  Total night calls       667 non-null    int64
 14  Total night charge      667 non-null    float64
 15  Total intl minutes      667 non-null    float64
 16  Total intl calls        667 non-null    int64
 17  Total intl charge       667 non-null    float64
 18  Customer service calls  667 non-null    int64
 19  Churn                   667 non-null    int64
dtypes: float64(8), int32(2), int64(9), object(1)
memory usage: 99.1+ KB
None
```

The test object contains 667 rows (subscribers). The 20 attributes do not contain null values. After the transformation, the types of the attributes are:

- 9 attributes are int64
- 8 attributes are float64
- 2 attributes are int32
- 1 attribute is object

Learning, prediction, and evaluation

To build the prediction model, four algorithms are used to choose the one that gives the best classification results.

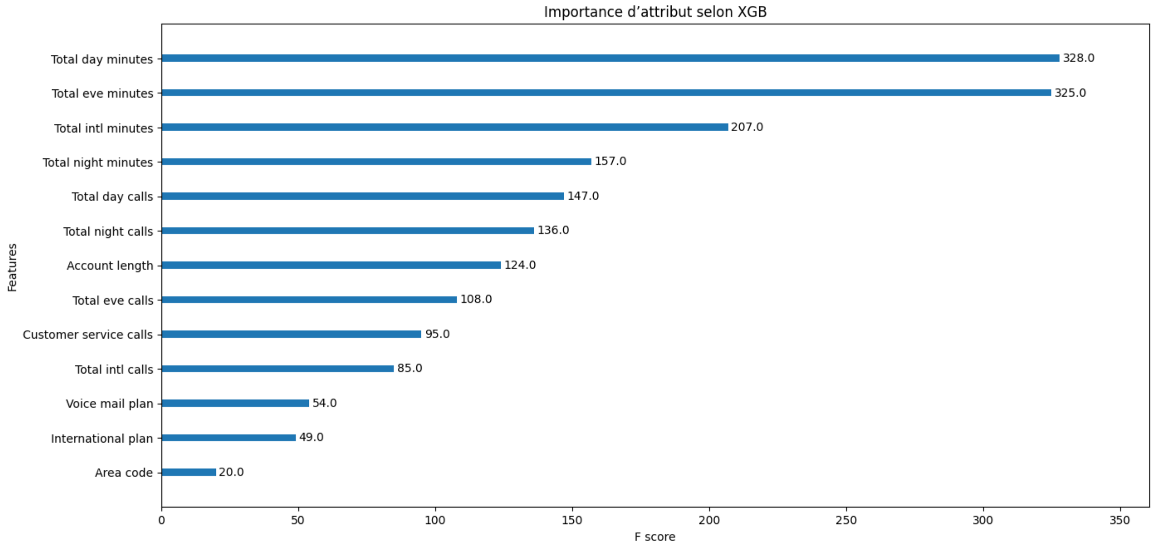
Create a function to call the classification algorithms

```
def classement(data, data_test):
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.ensemble import RandomForestClassifier
    from xgboost import XGBClassifier, plot_importance
    from sklearn.metrics import accuracy_score, precision_score, f1_score,
recall_score, confusion_matrix

    # Data preparation
    # Training data
    data = data[data["Churn"].notnull()] # delete rows that have missing
Churn (NA)
    x, y = data.drop("Churn", axis = 1), data[["Churn"]]
    supprimer.append('State')
    #print("supprimer: ", supprimer)
    x = x.drop(supprimer, axis = 1) # delete columns with names in the
'supprimer' list

    # Test Data
    data_test = data_test[data_test["Churn"].notnull()]
    x_test, y_test = data_test.drop("Churn", axis = 1), data_test[["Churn"]]
    x_test = x_test.drop(supprimer, axis = 1)

    # Algorithms
    KNC = KNeighborsClassifier()
    DTC = DecisionTreeClassifier()
    RFC = RandomForestClassifier()
    XGB = XGBClassifier()

    algorithms = [KNC, DTC, RFC, XGB]
    noms_algo = ['KNeighborsClassifier', 'DecisionTreeClassifier',
'RandomForestClassifier', 'XGBClassifier']

    # Metrics
    score_accuracy = []
    score_precision = []
    score_recall = []
    score_f1 = []

    # Learning, prediction, and evaluation
    for item in algorithmes:
        item.fit(x, y)
        prediction = item.predict(x_test)
        score_accuracy.append(accuracy_score(y_test, prediction))
        score_precision.append(precision_score(y_test, prediction))
        score_recall.append(recall_score(y_test, prediction))
        score_f1.append(f1_score(y_test, prediction))

    # Importance of Attributes
    plot_importance(XGB)
    plt.title('Importance d'attribut selon XGB')
    plt.grid(False)

    '''
```

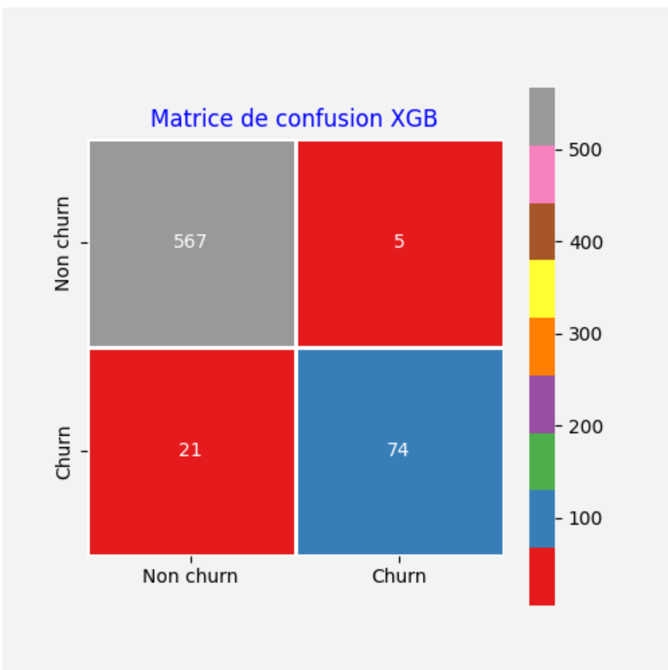
```
    apprenti = RFC.fit(x,y)
    feature_importance = np.array(apprenti.feature_importances_).tolist()
    feature_names = list(x.columns)
    fi_df = pd.DataFrame({'feature_names':feature_names,
'feature_importance':feature_importance})
    fi_df.sort_values(by=['feature_importance'], ascending = False, inplace =
True)
    plt.figure(figsize=(10,8))
    sns.barplot(x=fi_df['feature_importance'], y=fi_df['feature_names'])
    plt.title('Importance d'attribut selon Random Forest')
    plt.xlabel('F score')
    plt.ylabel('Attributs')
    '''
    #plt.show()

    # create confusion matrix for XGB
    fig = plt.figure(figsize=(5, 5))
    fig.set_facecolor("#F3F3F3")
    mat = confusion_matrix(y_test, XGB.predict(x_test))
    sns.heatmap(mat, annot=True, fmt = "d", square = True,
                xticklabels=["Non churn", "Churn"],
                yticklabels=["Non churn", "Churn"],
                linewidths = 2, linecolor = "w", cmap = "Set1")
    plt.title('Matrice de confusion XGB', color = "b")
    plt.subplots_adjust(wspace = .3, hspace = .3)

    '''
    # create confusion matrix for RandomForest
    fig = plt.figure(figsize=(5, 5))
    fig.set_facecolor("#F3F3F3")
    mat = confusion_matrix(y_test, RFC.predict(x_test))
    sns.heatmap(mat, annot=True, fmt = "d", square = True,
                xticklabels=["Non churn", "Churn"],
                yticklabels=["Non churn", "Churn"],
                linewidths = 2, linecolor = "w", cmap = "Set1")
    plt.title('Matrice de confusion RFC', color = "b")
    plt.subplots_adjust(wspace = .3, hspace = .3)
    '''
    plt.show()


    # Create dataframe of results
    result = pd.DataFrame(columns = ['score_accuracy','score_f1',
'score_recall', 'score_precision'], index = noms_algo)
    result['score_accuracy'] = score_accuracy
    result['score_f1'] = score_f1
    result['score_recall'] = score_recall
    result['score_precision'] = score_precision
    print(result.sort_values('score_accuracy', ascending = False))
    return result.sort_values('score_accuracy', ascending = False)

# Calling the previous function
classement(train, test)
```

	score_accuracy	score_f1	score_recall	score_precision
XGBClassifier	**0.961019**	**0.850575**	**0.778947**	**0.936709**
RandomForestClassifier	0.943028	0.759494	0.631579	0.952381
DecisionTreeClassifier	0.922039	0.729167	0.736842	0.721649
KNeighborsClassifier	0.877061	0.388060	0.273684	0.666667

The table above shows that the XGBClassifier algorithm gives the model with the best results.



The figure above shows that the most important attributes in the churn prediction (according to the XGB algorithm) are 'Account length', 'Voice mail plan' and 'International plan', excluding the attributes of calls and call minutes.



The XGB confusion matrix shows that a minority of cases are in the False Positive or False Negative squares (the red squares).

Observations and conclusions

The best model is the one created by the XGBoost algorithm, which gives 96% accuracy.

Based on the analysis, tables and graphs generated in this document, the following points can be concluded to minimize churn and predict which subscribers are likely to churn to switch networks:

- It is advisable to optimize and/or implement a network infrastructure that gives a better quality of service to subscribers, especially in states with the highest churn cases:
  - New Jersey,
  - Texas,
  - Maryland,
  - Michigan,
  - Minnesota and
  - New York.

- We see that there are cases of churn after making a single call to customer service. There are also some who churn without calling customer service.
  It is recommended to improve customer service to increase the First Call Resolution metric.

- According to the attribute importance chart based on the XGB algorithm, the most important attributes that impact churn (excluding the number of minutes and calls attributes) are:
  - Account length (the number of days the account is active): it is recommended to offer attractive promotions to subscribers who have their accounts active for a certain minimum period of time (to be defined by the network management).
  - Voice mail plan (if the subscriber has the voicemail service): it is recommended to optimize this service and to add offers that are of interest to subscribers and preferably that do not exist in other networks (for example a large number of messages that can be recorded per subscriber).
  - International plan (if the subscriber has an international calling plan): It is recommended to optimize international calling plans to increase the loyalty of subscribers who use this service.

- It is recommended to make the churn prediction regularly, using the XGBoost algorithm. It's so much better to take preventive measures rather than corrective measures. The former has a greater effect on subscriber’s loyalty.