

RAPPORT PROJET D'ETUDE

(KHALID OURO-ADOYI)

Master 1 IA

PLAN DU RAPPORT

- Contexte
- Identification des facteurs de désabonnement.
- Explorer la corrélation entre les variables.
- Construction de modèle de prédiction
- Evaluation de la performance du modèle
- Construction d'application churn_app

PROJET D'ETUDE

CONTEXTE

Analyse d'un Jeu de Données sur le Désabonnement des clients d'une entreprise de télécommunication en Iran.

Date de Collecte des Données : 4 août 2020

Source des Données : Les données sont collectées aléatoirement à partir de la base de données d'une entreprise de télécommunications en Iran sur une période de 12 mois.

Objectif :

L'objectif de ce projet est d'analyser un ensemble de données concernant le désabonnement des clients d'une entreprise de télécommunications en Iran. Les données sont collectées pour mieux comprendre les raisons potentielles du désabonnement et prédire si un client se désabonnera ou non.

Caractéristiques du Jeu de Données :

Nature : Multi variable (plusieurs attributs pour chaque observation).

Type d'Attribut : Entier (valeurs numériques).

Taille du Jeu de Données :

- Nombre d'Observations : 3150 (chaque ligne représente un client).
- Nombre de Variables : 14 (attributs).

Liste des Variables :

1. ID du client anonyme.
2. Échecs d'appel : Nombre d'échecs d'appel.
3. Plaintes : Binaire (0 : pas de plainte, 1 : plainte).
4. Durée de l'abonnement : Nombre total de mois d'abonnement.

5. Montant des frais : Attribut ordinal (0 : montant le plus bas, 9 : montant le plus élevé).
6. Secondes d'utilisation : Nombre total de secondes d'appel.
7. Fréquence d'utilisation : Nombre total d'appels.
8. Fréquence des SMS : Nombre total de messages texte.
9. Numéros appelés distincts : Nombre total d'appels téléphoniques distincts.
10. Groupe d'âge : Attribut ordinal (1 : plus jeune, 5 : plus âgé).
11. Plan tarifaire : Binaire (1 : Pay as you go, 2 : contractuel).
12. Statut : Binaire (1 : actif, 2 : non actif).
13. Désabonnement : Binaire (1 : désabonnement, 0 : non désabonnement) - Étiquette de classe.
14. Valeur du client : Valeur calculée du client.

Informations Supplémentaires : Les données représentent des informations agrégées sur une période de 9 mois, à l'exception de l'attribut "Désabonnement" qui représente l'état des clients à la fin des 12 mois.

Méthodologie du travail :

- Identifier les facteurs qui influencent le désabonnement des clients.
- Construire un modèle de classification pour prédire si un client se désabonne.
- Évaluer la performance du modèle en utilisant des métriques appropriées (précision, rappel, F1-score, etc.).
- Explorer la corrélation entre les variables pour mieux comprendre les relations entre elles.
- Prédire la valeur du client en utilisant des méthodes de régression.

Ce projet vise à fournir des informations et des prédictions utiles à l'entreprise de télécommunications pour mieux comprendre et gérer les taux de désabonnement de ses clients.

Double-cliquez (ou appuyez sur Entrée) pour modifier

```
# Importation de librairies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from numpy.lib.function_base import median
#Pour rendre le graphique interactif, on importe les éléments de la librairie ipywidgets
import ipywidgets as widgets
from ipywidgets import interact, interactive, fixed, interact_manual

# Les différents algorithmes nécessaires :
from sklearn.linear_model import LogisticRegression # regression logistic
from sklearn.ensemble import RandomForestClassifier # Forêt aléatoire
from sklearn.neural_network import MLPClassifier # Réseau de neurone artificielle
from sklearn.svm import SVC #support vector machine : le classificateur pour trouver
# le meilleur hyperplan de la variable cible

from sklearn.model_selection import GridSearchCV # chercher les meilleurs hyper-paramètres pour nos algorithmes
from sklearn.metrics import classification_report # Qui stock plusieurs métriques comme la précision du modèle, le F1 score
from sklearn.feature_selection import RFE # Sélection de variables prédictives importante : réduire la dimensionnalité
# de notre jeu de données tout en préservant la performance des algorithmes
```

Ce code met en place un environnement de machine learning en utilisant Python, en important diverses bibliothèques et en configurant des algorithmes de classification. Détaillons :

1. Importation de bibliothèques :

- `import numpy as np` : Une bibliothèque pour effectuer des opérations mathématiques et de manipulation sur des tableaux de données.
- `import pandas as pd` : Une bibliothèque pour la manipulation et l'analyse de données tabulaires.
- `import matplotlib.pyplot as plt` : Une bibliothèque pour créer des graphiques et visualiser les données.
- `import seaborn as sns` : Une bibliothèque pour la visualisation de données basée sur matplotlib, facilitant la création de graphiques esthétiques.
- `ipywidgets` : Une bibliothèque pour créer des éléments d'interface utilisateur interactifs.
- `interact, interactive, fixed, interact_manual` : Fonctions de pywidgets pour créer des interfaces interactives.

2. Algorithmes de classification :

- `LogisticRegression` : Un algorithme de régression logistique, utilisé pour résoudre des problèmes de classification.
- `RandomForestClassifier` : Un algorithme de forêt aléatoire, qui construit plusieurs arbres de décision pour améliorer les performances de classification.

- MLPClassifier : Un classificateur basé sur un réseau de neurones artificiels, capable de gérer des problèmes de classification complexes.

- SVC (Support Vector Classifier) : Un classificateur basé sur les machines à vecteurs de support, utilisé pour trouver le meilleur hyperplan pour séparer les classes.

3. Recherche d'hyper-paramètres :

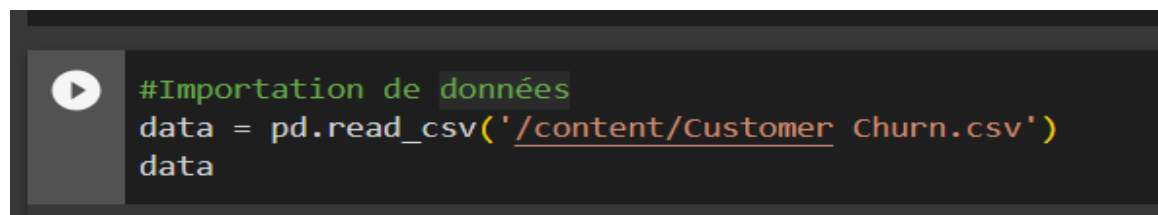
- GridSearchCV : Une technique de recherche d'hyper-paramètres qui essaie différentes combinaisons de paramètres pour obtenir les meilleures performances d'algorithmes.

4. Autres fonctionnalités :

- classification_report : Une fonction pour générer un rapport de classification avec différentes métriques telles que la précision, le rappel et le F1-score.

- RFE (Recursive Feature Elimination) : Une technique de sélection de caractéristiques qui élimine les variables moins importantes pour réduire la dimensionnalité des données tout en conservant les performances de l'algorithme.

En utilisant ces bibliothèques et algorithmes, vous pouvez construire des modèles de machine learning, effectuer des analyses et des prédictions sur des ensembles de données. Les widgets interactifs d'`ipywidgets` permettent de manipuler et d'observer les résultats de manière interactive dans un environnement Jupyter Notebook.

A screenshot of a Jupyter Notebook cell with a dark background. On the left, there is a play button icon. To its right, the following code is written in a monospaced font:

```
#Importation de données
data = pd.read_csv('/content/Customer Churn.csv')
data
```

Ce code concerne l'importation de données à partir d'un fichier CSV et leur stockage dans un objet appelé `data` à l'aide de la bibliothèque `pandas`. Voici une explication détaillée :

1. Importation de données :

- `pd.read_csv('/content/Customer Churn.csv')` : Cette ligne de code utilise la fonction `read_csv` de la bibliothèque `pandas` pour lire les données d'un fichier CSV. Le chemin `/content/Customer Churn.csv` est l'emplacement du fichier CSV que vous souhaitez lire.

2. Stockage des données :

- Une fois que les données sont lues à partir du fichier CSV, elles sont stockées dans un objet appelé `data`. Cet objet est une structure de données appelée `DataFrame`, spécifique à la bibliothèque `pandas`. Un `DataFrame` est une manière organisée et tabulaire de stocker des données, semblable à une feuille de calcul.

	Call Failure	Complains	Subscription Length	Charge Amount	Seconds of Use	Frequency of use	Frequency of SMS	Distinct Called Numbers	Age Group	Tariff Plan	Status	Age	Customer Value	Churn
0	8	0	38	0	4370	71	5	17	3	1	1	30	197.640	0
1	0	0	39	0	318	5	7	4	2	1	2	25	46.035	0
2	10	0	37	0	2453	60	359	24	3	1	1	30	1536.520	0
3	10	0	38	0	4198	66	1	35	1	1	1	15	240.020	0
4	3	0	38	0	2393	58	2	33	1	1	1	15	145.805	0
...
3145	21	0	19	2	6697	147	92	44	2	2	1	25	721.980	0
3146	17	0	17	1	9237	177	80	42	5	1	1	55	261.210	0
3147	13	0	18	4	3157	51	38	21	3	1	1	30	280.320	0
3148	7	0	11	2	4695	46	222	12	3	1	1	30	1077.640	0
3149	8	1	11	2	1792	25	7	9	3	1	1	30	100.680	1

3150 rows x 14 columns

ANALYSE EXPLORATOIRE DE NOS DONNEES : elle permet de mieux comprendre les données et de comprendre le type de prétraitement de données à effectuer.

`data.shape` : Cette ligne de code renvoie les dimensions (nombre de lignes et de colonnes) de votre `DataFrame` `data`. Cela peut vous donner une idée du nombre de points de données et du nombre de caractéristiques (colonnes) dans votre ensemble de données.

```
# Voir l'étendue de notre base de données
data.shape

(3150, 14)
```

`data.info()` : Cette méthode affiche des informations sur le `DataFrame` `data`, notamment le nombre total d'entrées non nulles par colonne et le type de données de chaque colonne. Cela peut vous aider à comprendre quelles

colonnes contiennent des données manquantes et quelles colonnes sont de quel type.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3150 entries, 0 to 3149
Data columns (total 14 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Call Failure                          3150 non-null   int64
1   Complains                             3150 non-null   int64
2   Subscription Length                  3150 non-null   int64
3   Charge Amount                        3150 non-null   int64
4   Seconds of Use                       3150 non-null   int64
5   Frequency of use                     3150 non-null   int64
6   Frequency of SMS                     3150 non-null   int64
7   Distinct Called Numbers              3150 non-null   int64
8   Age Group                            3150 non-null   int64
9   Tariff Plan                          3150 non-null   int64
10  Status                               3150 non-null   int64
11  Age                                  3150 non-null   int64
12  Customer Value                       3150 non-null   float64
13  Churn                                3150 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 344.7 KB
```

`data.isna().sum()` : Cette ligne de code calcule la somme des valeurs manquantes (NaN) pour chaque colonne dans le DataFrame data. Cela peut vous aider à identifier les colonnes avec des données manquantes.


```
#Voir le nombre de variables manquantes par variables
data.isna().sum()
```

```
Call Failure      0
Complains         0
Subscription Length 0
Charge Amount     0
Seconds of Use    0
Frequency of use   0
Frequency of SMS   0
Distinct Called Numbers 0
Age Group         0
Tariff Plan       0
Status           0
Age              0
Customer Value    0
Churn            0
dtype: int64
```

`data.head()` : Cette méthode affiche les premières lignes du DataFrame `data`, par défaut les 5 premières lignes. Cela vous permet de jeter un coup d'œil rapide aux premières observations de votre ensemble de données.

```
[ ] data.head()
```

	Call Failure	Complains	Subscription Length	Charge Amount	Seconds of Use	Frequency of use	Frequency of SMS	Distinct Called Numbers	Age Group	Tariff Plan	Status	Age	Customer Value	Churn
0	8	0	38	0	4370	71	5	17	3	1	1	30	197.640	0
1	0	0	39	0	318	5	7	4	2	1	2	25	46.035	0
2	10	0	37	0	2453	60	359	24	3	1	1	30	1536.520	0
3	10	0	38	0	4198	66	1	35	1	1	1	15	240.020	0
4	3	0	38	0	2393	58	2	33	1	1	1	15	145.805	0

`data.nunique()` : Cette méthode renvoie le nombre de valeurs uniques dans chaque colonne du DataFrame `data`. Cela peut vous aider à comprendre la variabilité et la diversité des données dans chaque colonne.

```
#Pour voir le nombre de valeur unique par colonne : elle nous donne le nombre
#de valeurs distinctes au niveau de chaque colonne
data.nunique()
```

```
Call Failure      37
Complains         2
Subscription Length 45
Charge Amount     11
Seconds of Use    1756
Frequency of use   242
Frequency of SMS   405
Distinct Called Numbers 92
Age Group         5
Tariff Plan       2
Status           2
Age              5
Customer Value    2654
Churn            2
dtype: int64
```

STATISTIQUE DESCRIPTIVE

Calcul des moyennes pour les colonnes spécifiées

```
# Calcul des moyennes pour les colonnes spécifiées
columns_to_calculate_mean = ['Seconds of Use', 'Frequency of use', 'Frequency of SMS', 'Age', 'Customer Value']
mean_values = data[columns_to_calculate_mean].mean()

print("Moyennes :\n", mean_values)
```

```
Moyennes :
Seconds of Use    4472.459683
Frequency of use   69.460635
Frequency of SMS   73.174921
Age               30.998413
Customer Value    470.972916
dtype: float64
```

#Calcul de mediane

```
#Calcul de mediane
columns_to_calculate_median= ['Seconds of Use', 'Frequency of use', 'Frequency of SMS', 'Age', 'Customer Value']
median_values = data[columns_to_calculate_median].median()

print("Medianes:\n", median_values)
```

```
Medianes:
Seconds of Use    2990.00
Frequency of use   54.00
Frequency of SMS   21.00
Age               30.00
Customer Value    228.48
dtype: float64
```

#Calcul de l'écart type

```
#Calcul de l'écart type
column_ecart_types = ['Seconds of Use','Frequency of use','Frequency of SMS','Age','Customer Value']
ecart_type_values = data[column_ecart_types].std()

print("Ecart_types:\n", ecart_type_values)
```

```
Ecart_types:
Seconds of Use      4197.908687
Frequency of use    57.413308
Frequency of SMS    112.237560
Age                 8.831095
Customer Value      517.015433
dtype: float64
```

Calcul des quantiles pour certaines colonnes :

```
# Colonnes pour lesquelles vous voulez calculer les quantiles
columns_to_calculate_quantiles = ['Seconds of Use', 'Frequency of use', 'Frequency of SMS', 'Age', 'Customer Value']

# Calcul des quantiles pour les colonnes spécifiées
quantiles = data[columns_to_calculate_quantiles].quantile([0.25, 0.5, 0.75])

print("Quantiles :\n", quantiles)
```

```
Quantiles :
Seconds of Use  Frequency of use  Frequency of SMS  Age  Customer Value
0.25           1391.25           27.0           6.0  25.0           113.80125
0.50           2990.00           54.0           21.0  30.0           228.48000
0.75           6478.25           95.0           87.0  30.0           788.38875
```

RESUMES STATISTIQUES

```
# data.describe()
```

	Call Failure	Complains	Subscription Length	Charge Amount	Seconds of Use	Frequency of use	Frequency of SMS	Distinct Called Numbers	Age Group	Tariff Plan	Status
count	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000	3150.000000
mean	7.627937	0.076508	32.541905	0.942857	4472.459683	69.460635	73.174921	23.509841	2.826032	1.077778	1.24825
std	7.263886	0.265851	8.573482	1.521072	4197.908687	57.413308	112.237560	17.217337	0.892555	0.267864	0.43200
min	0.000000	0.000000	3.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.00000
25%	1.000000	0.000000	30.000000	0.000000	1391.250000	27.000000	6.000000	10.000000	2.000000	1.000000	1.00000
50%	6.000000	0.000000	35.000000	0.000000	2990.000000	54.000000	21.000000	21.000000	3.000000	1.000000	1.00000
75%	12.000000	0.000000	38.000000	1.000000	6478.250000	95.000000	87.000000	34.000000	3.000000	1.000000	1.00000
max	36.000000	1.000000	47.000000	10.000000	17090.000000	255.000000	522.000000	97.000000	5.000000	2.000000	2.00000

VISUALISATIONS

#Variables catégorielles : celles ayant moins de 5 modalités

to_list() pour convertir les noms de colonnes en une liste

```
[ ] #Variables catégorielles : celles ayant moins de 5 modalités
#NB: to_list() pour convertir les noms de colonnes en une liste
categorical_columns = data.nunique()[data.nunique()<6].keys().to_list()
categorical_columns

['Complains', 'Age Group', 'Tariff Plan', 'Status', 'Age', 'Churn']
```



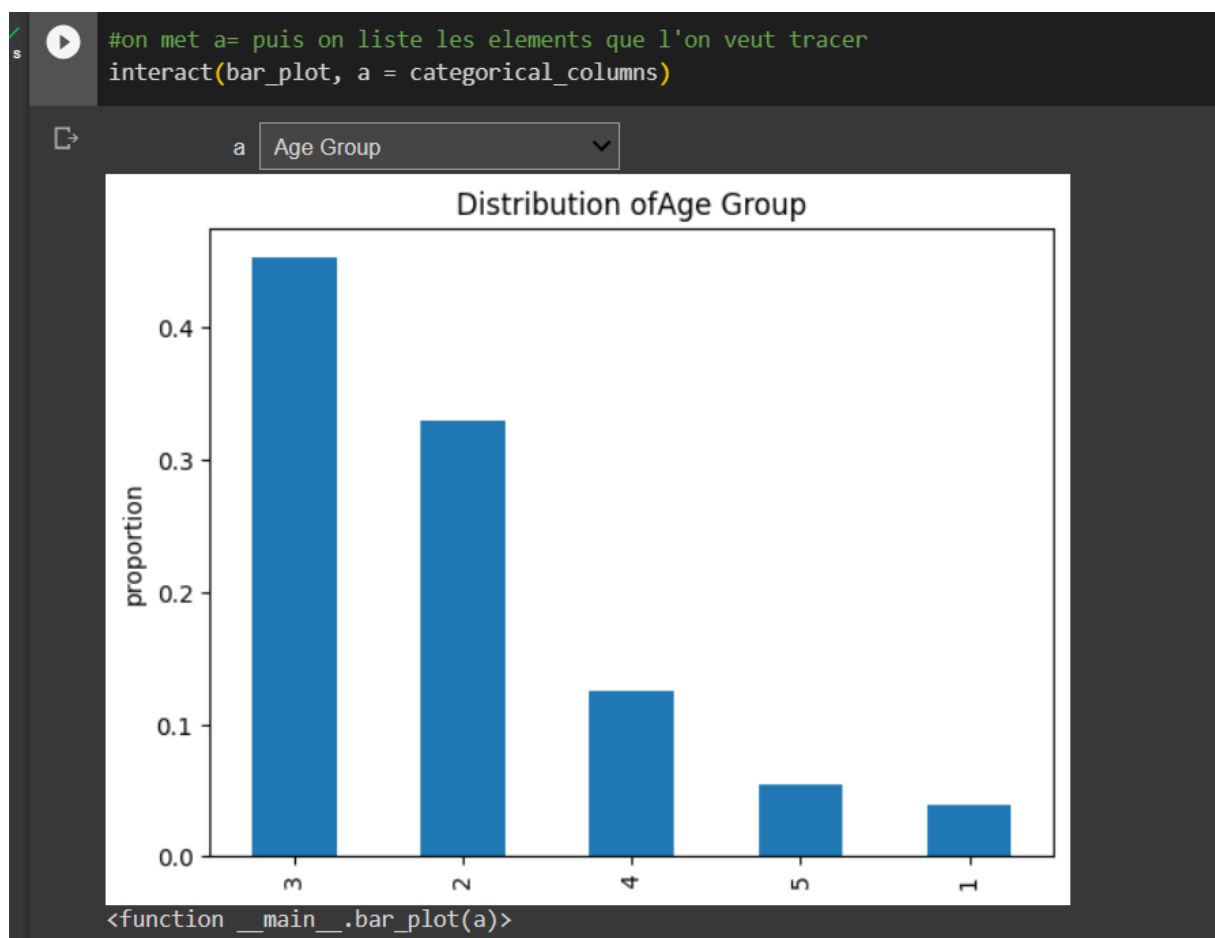
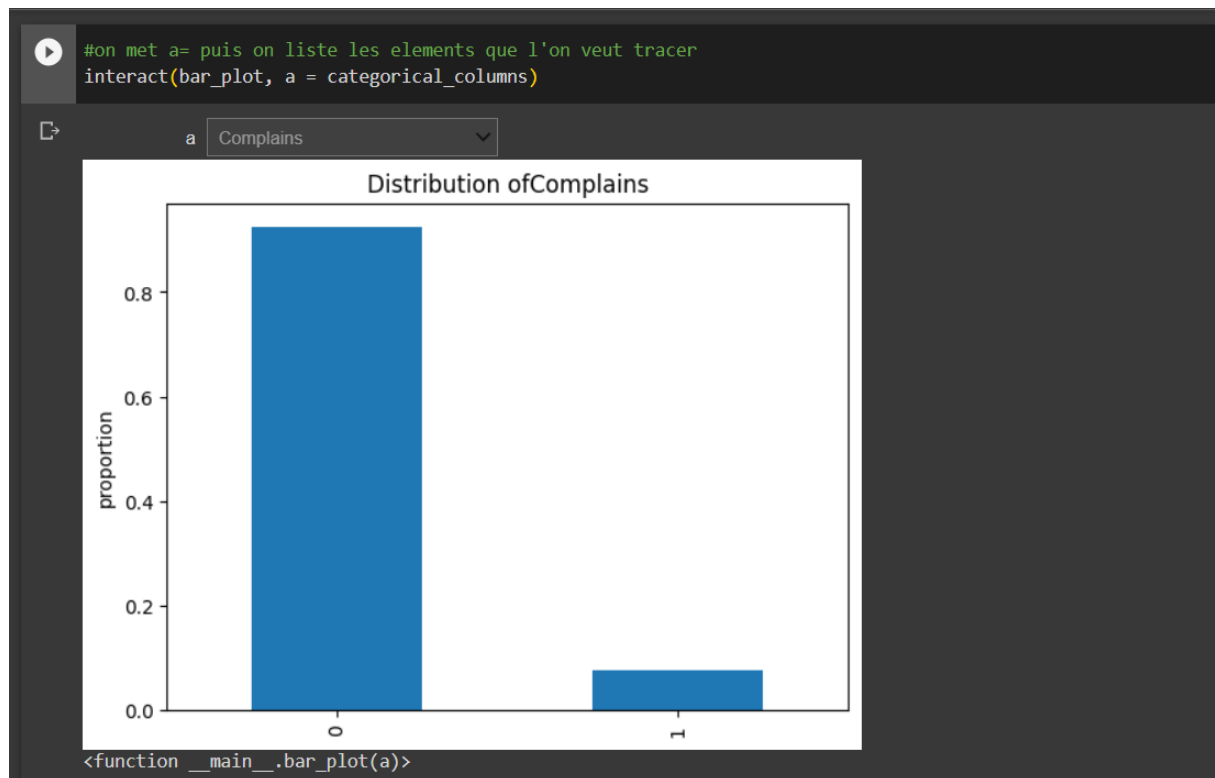
```
#Présenter les variables sous forme de colonne :  
for column in categorical_columns:  
    print(column)
```

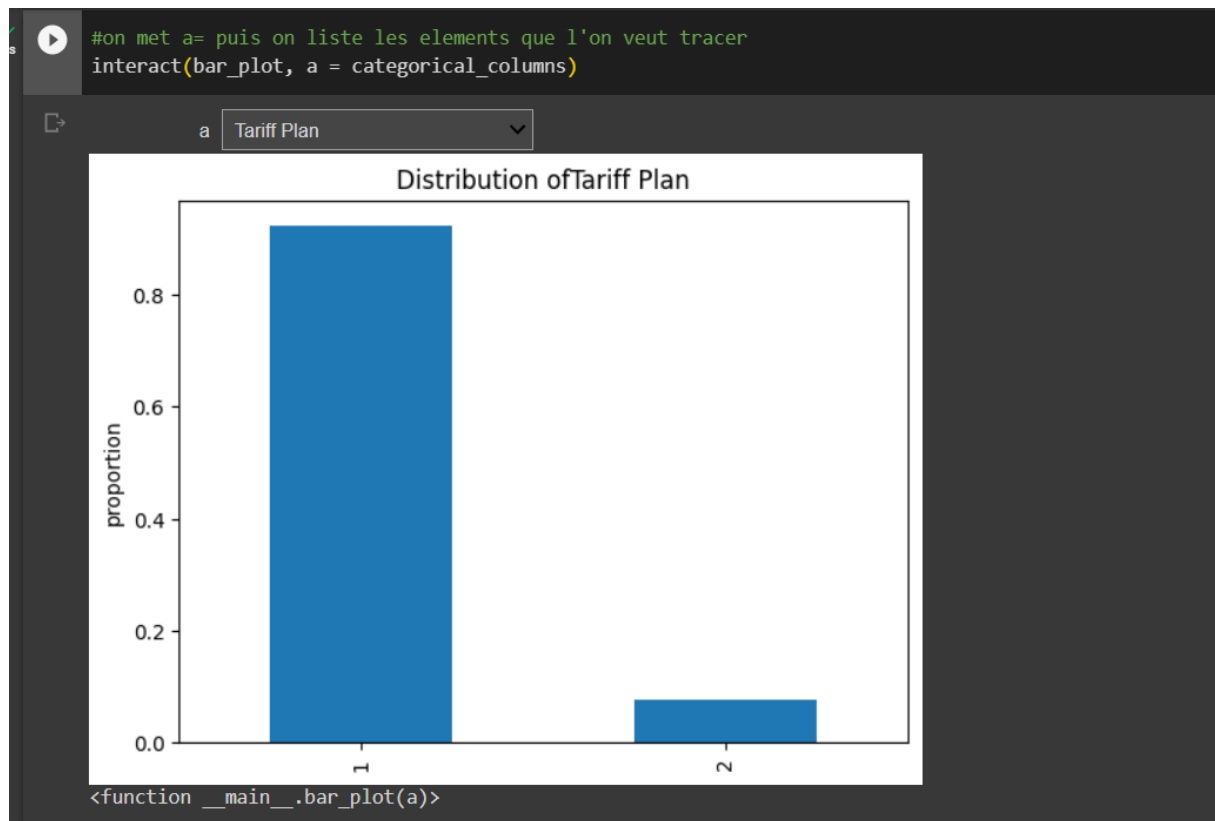


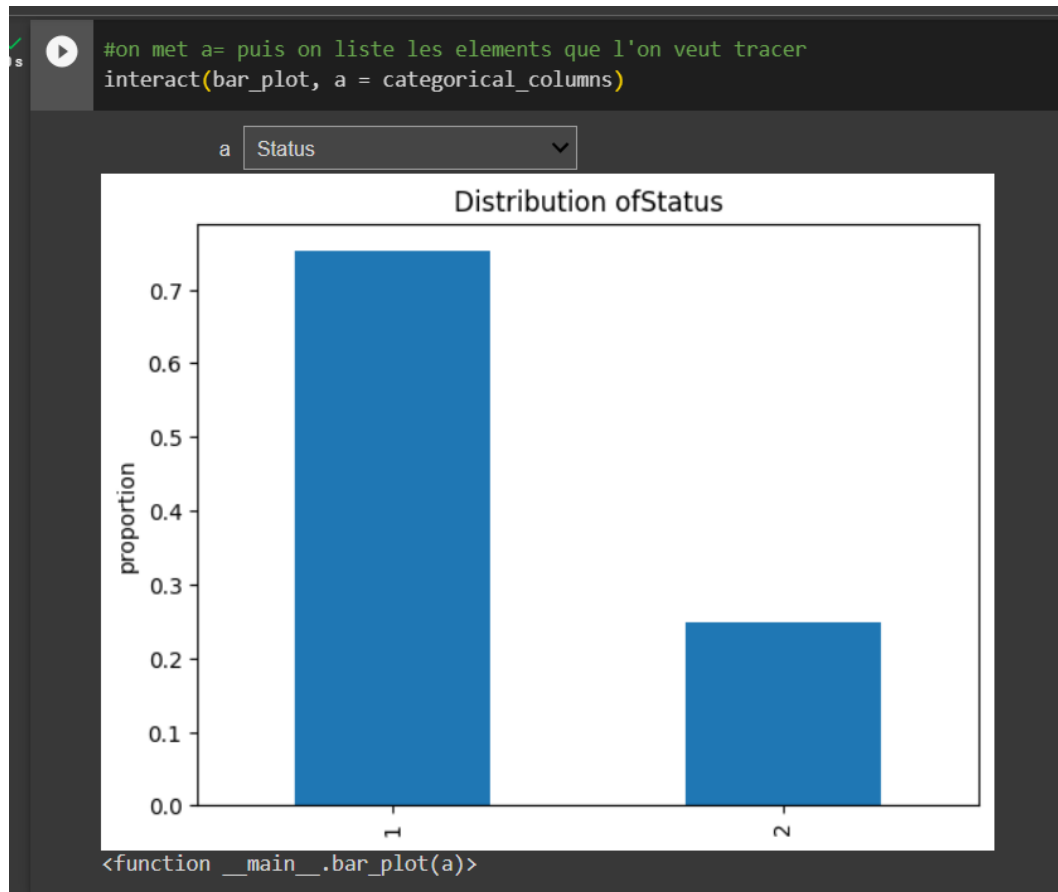
```
Complains  
Age Group  
Tariff Plan  
Status  
Age  
Churn
```

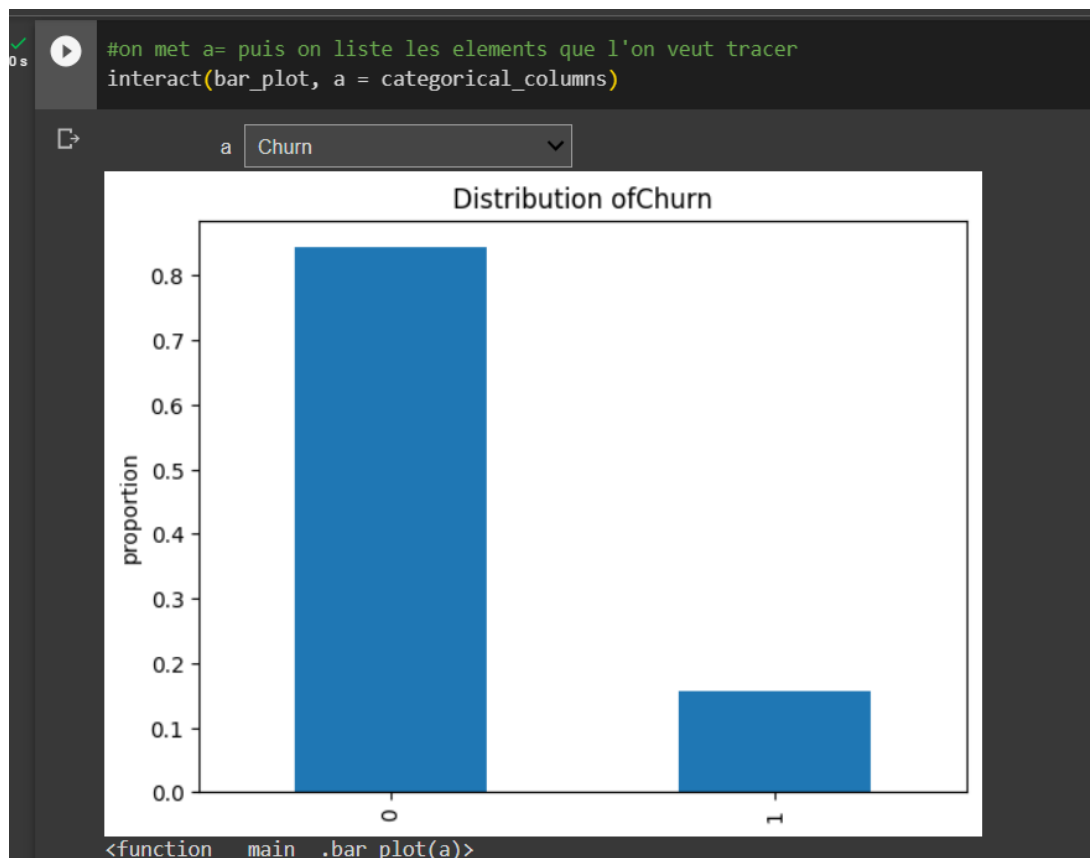
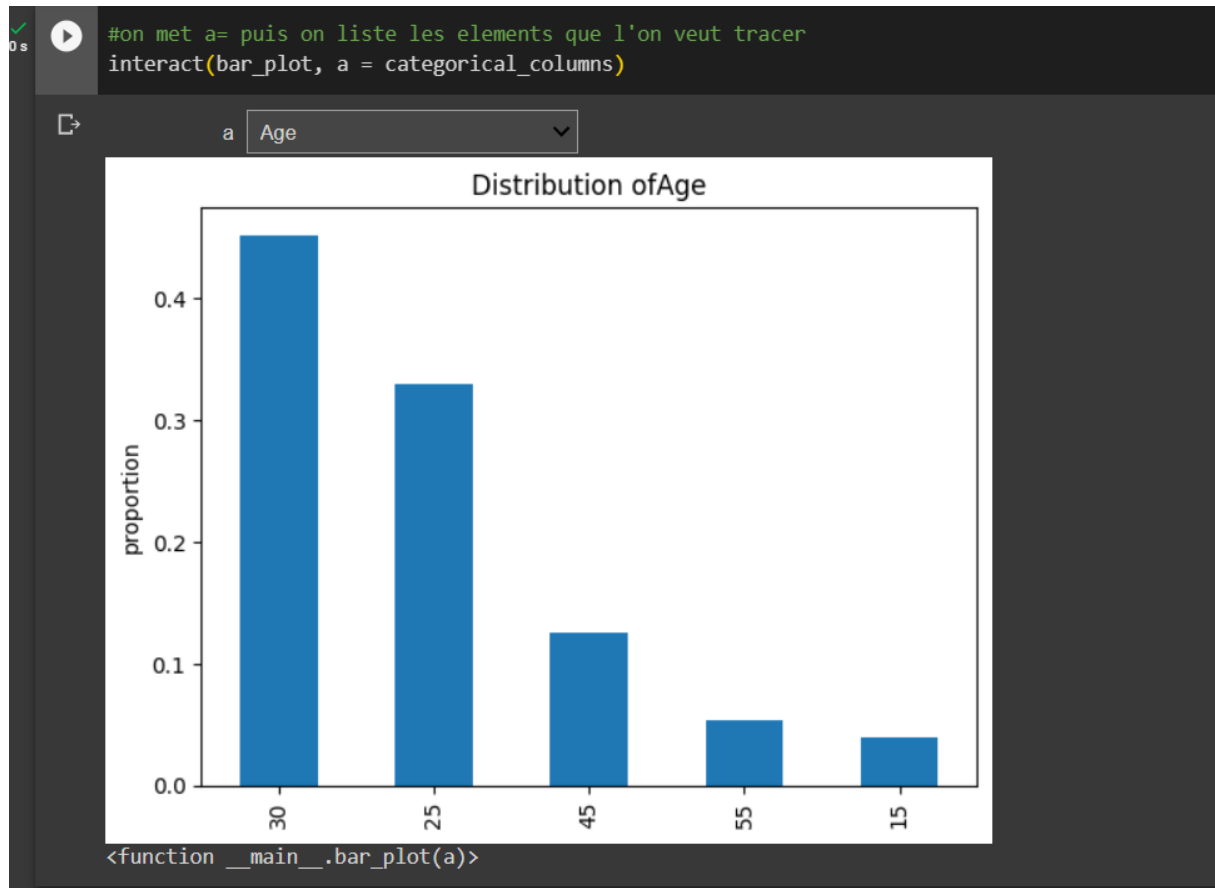
Utiliser la fonction interact pour créer nos graphes en barres interactifs

```
# Pour réaliser des graphiques  
import seaborn as sns  
import matplotlib.pyplot as plt  
# Pour inclure les graphes à notre notebook et ne pas  
les avoirs séparées  
%matplotlib inline  
  
#Pour rendre le graphique interactif, on importe les  
éléments de la librairie ipywidgets  
import ipywidgets as widgets  
from ipywidgets import interact, interactive, fixed,  
interact_manual
```









Remarque : La distribution de la variable churn montre que le pourcentage de clients fidèles est supérieur à celui de clients non fidèles (plus de 85% contre moins de 15%).

Aussi on remarque qu'au niveau de notre target (churn) il ya un problème de déséquilibre de classe qu'il faut résoudre car cela peut affecter négativement notre modèle.

ANALYSE UNIVARIEE DE VARIABLES QUANTITATIVES

```
#Variables quantitatives
numerical_columns = ['Frequency of use', 'Frequency of SMS', 'Customer Value', 'Seconds of Use']

# Création d'une fonction de construction d'histogramme de manière interactive
def hist_plot (b):
    sns.distplot(data[b], kde=False)
    plt.title('Histogram of '+str(b))
    return plt.show()

#Avant d'afficher l'histogramme, il faut convertir le type de la variable 'TotalCharges' en float
#data['Customer Value'] = data['Customer Value'].replace(' ', np.nan).astype(float)
```

#Voir le nombre de variables manquantes par variables

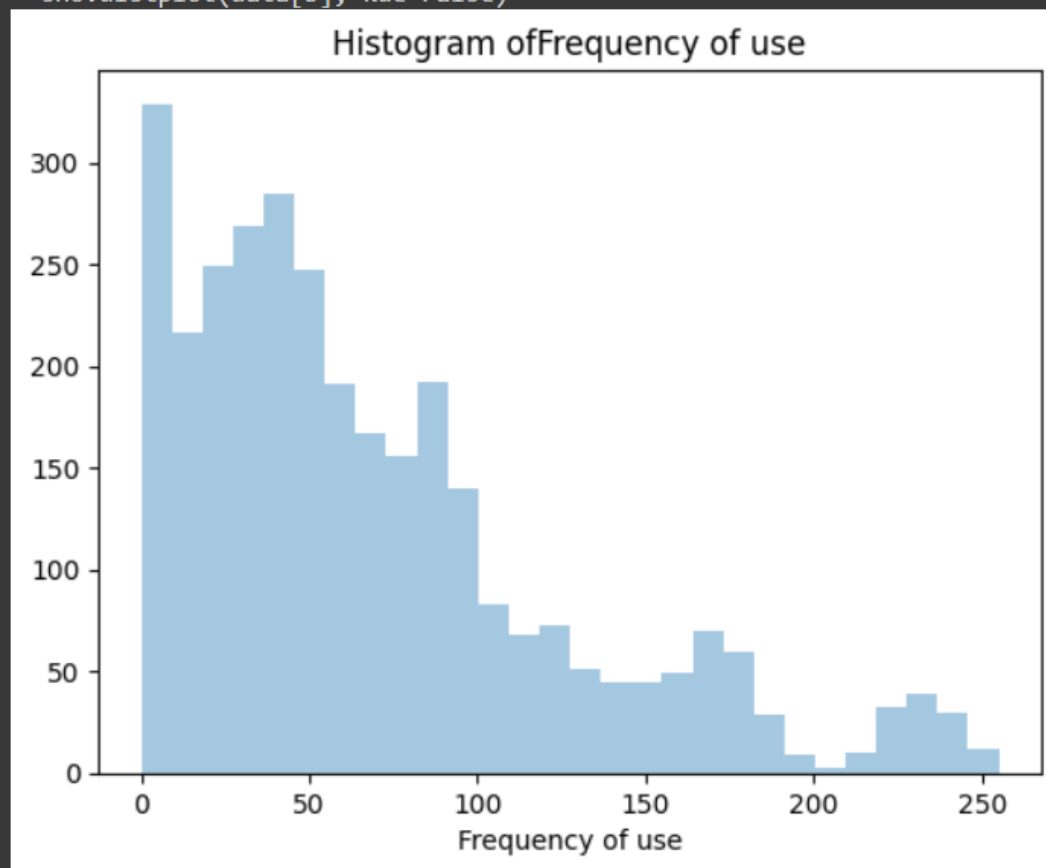
```
#Voir le nombre de variables manquantes par variables
data.isna().sum()
```

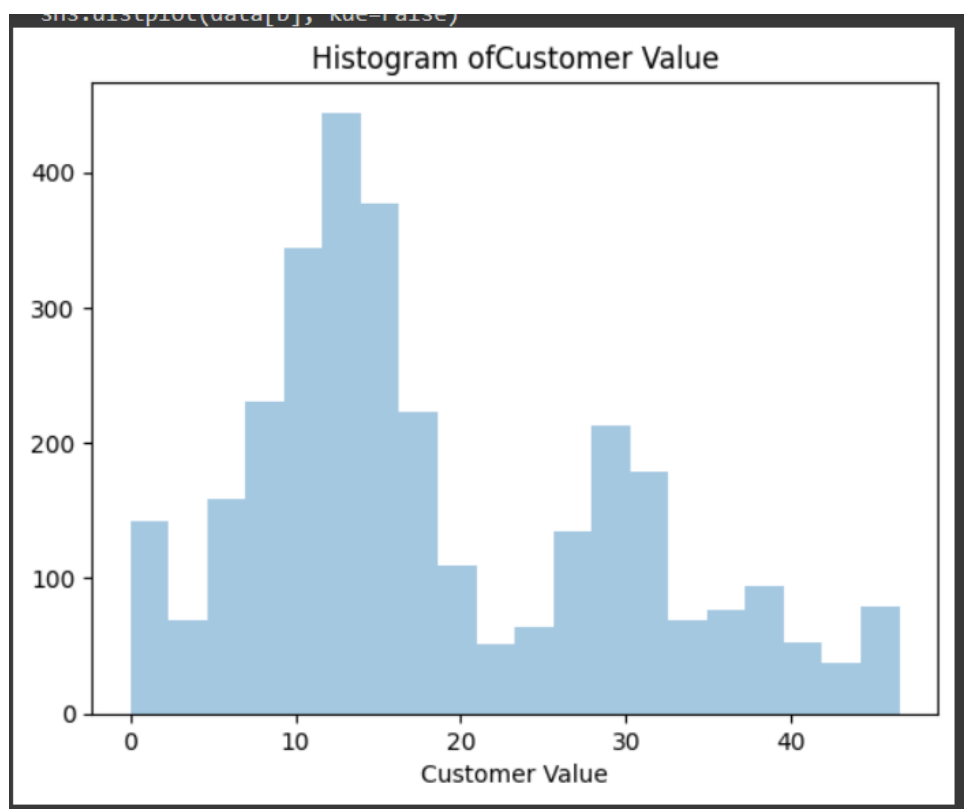
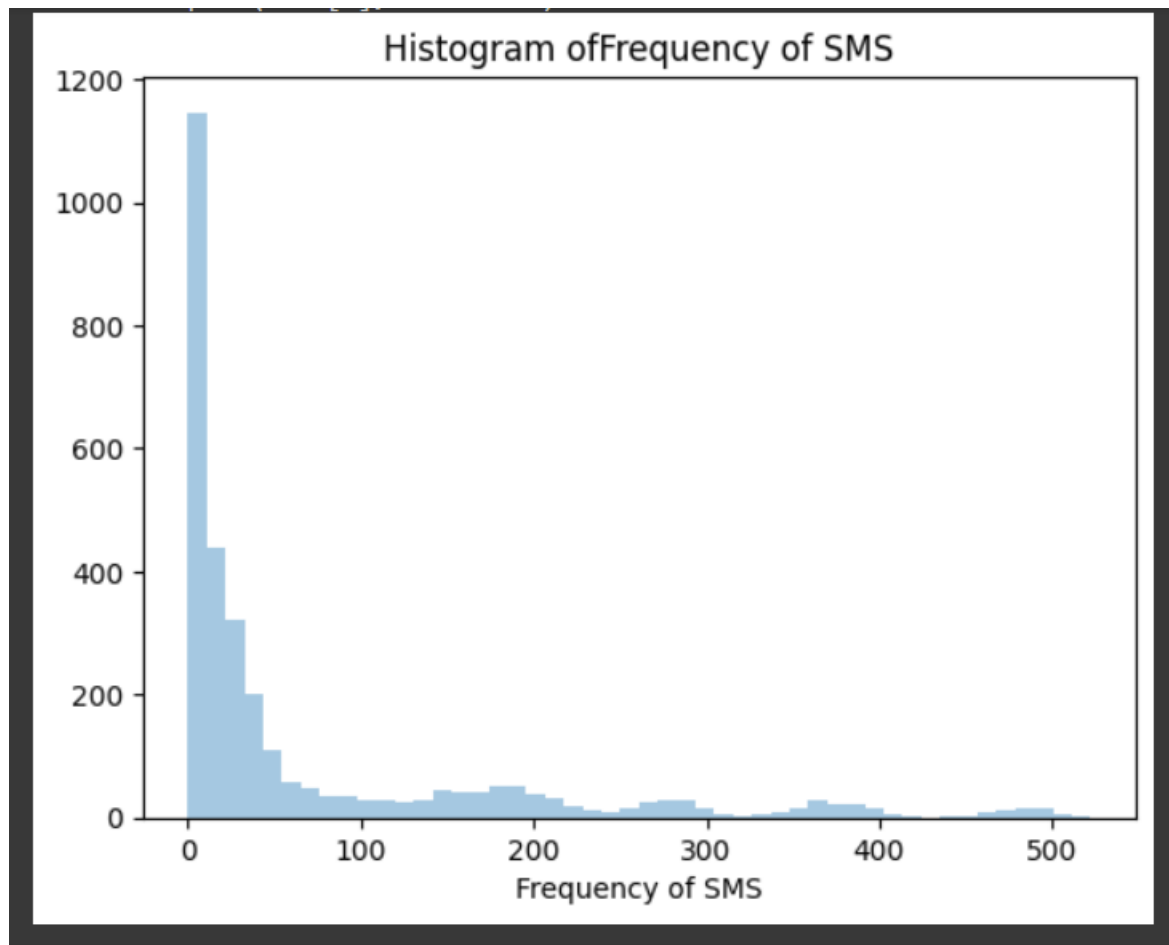
Call Failure	0
Complains	0
Subscription Length	0
Charge Amount	0
Seconds of Use	0
Frequency of use	0
Frequency of SMS	0
Distinct Called Numbers	0
Age Group	0
Tariff Plan	0
Status	0
Age	0
Customer Value	0
Churn	0
dtype: int64	

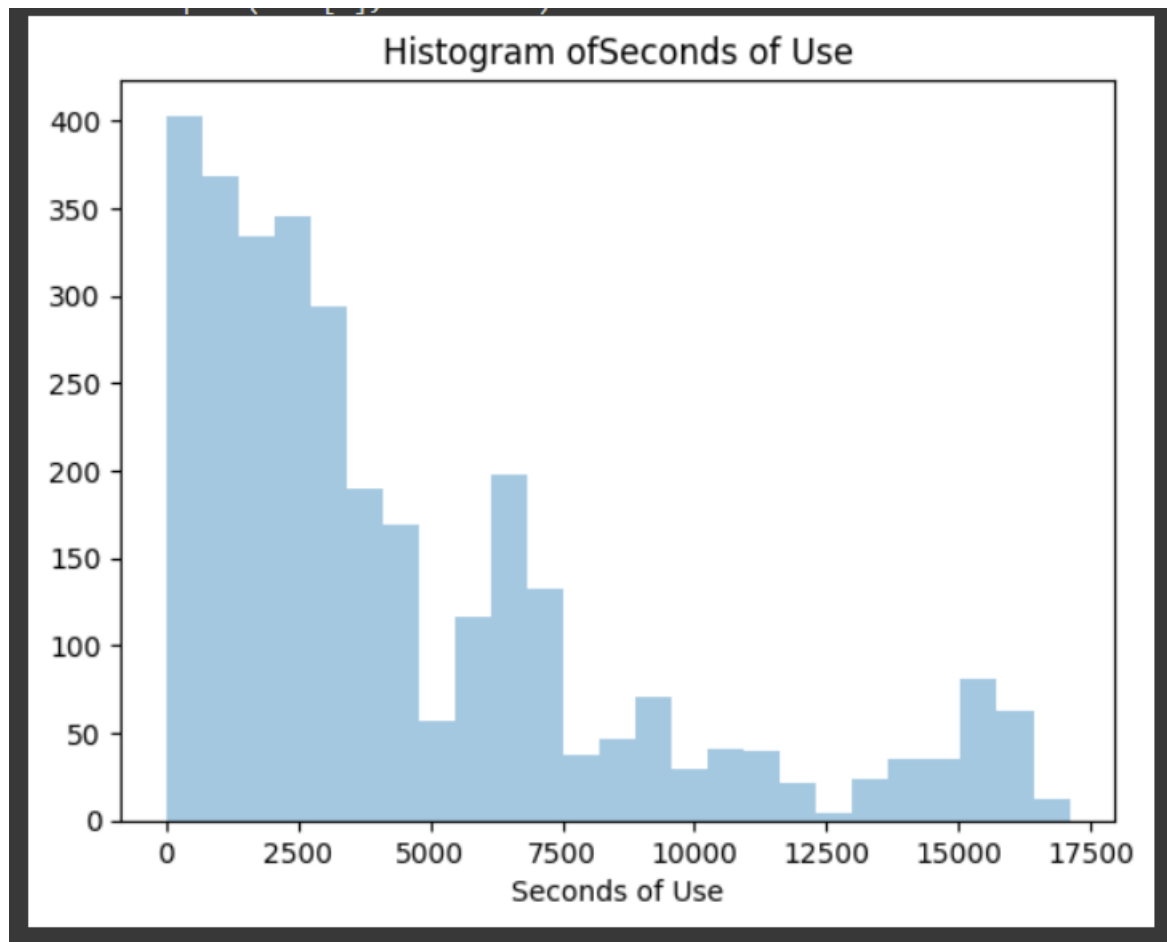
#Afficher les histogrammes avec la fonction (interact)

```
interact(hist_plot, b = numerical_columns);
```

```
sns.distplot(data[b], kde=False)
```







#Coefficient d'asymétrie des variables

```
#Coefficient d'asymetrie de la variable Seconds of Use
data['Seconds of Use'].skew()
# On voit que ce coef est très grand(tend vers 1) donc Asymétrique
#il faut transformer la variable afin de la rendre plus symétrique (Dans le prétraitement)
```

1.3219428645354965

```
[19] #Coefficient d'asymetrie de la variable Frequency of SMS
data['Frequency of SMS'].skew()
# On voit que ce coef est très grand(tend vers 1) donc Asymétrique
#il faut transformer la variable afin de la rendre plus symétrique (Dans le prétraitement)
```

1.9741417670137227

```
[20] #Coefficient d'asymetrie de la variable Frequency of use
data['Frequency of use'].skew()
# On voit que ce coef est très grand(tend vers 1) donc Asymétrique
#il faut transformer la variable afin de la rendre plus symétrique (Dans le prétraitement)
```

1.1441664249623964

```
[21] #Coefficient d'asymetrie de la variable customer Value
data['Customer Value'].skew()
# On voit que ce coef est très grand(tend vers 1) donc Asymétrique
#il faut transformer la variable afin de la rendre plus symétrique (Dans le prétraitement)
```

1.4272916100327098

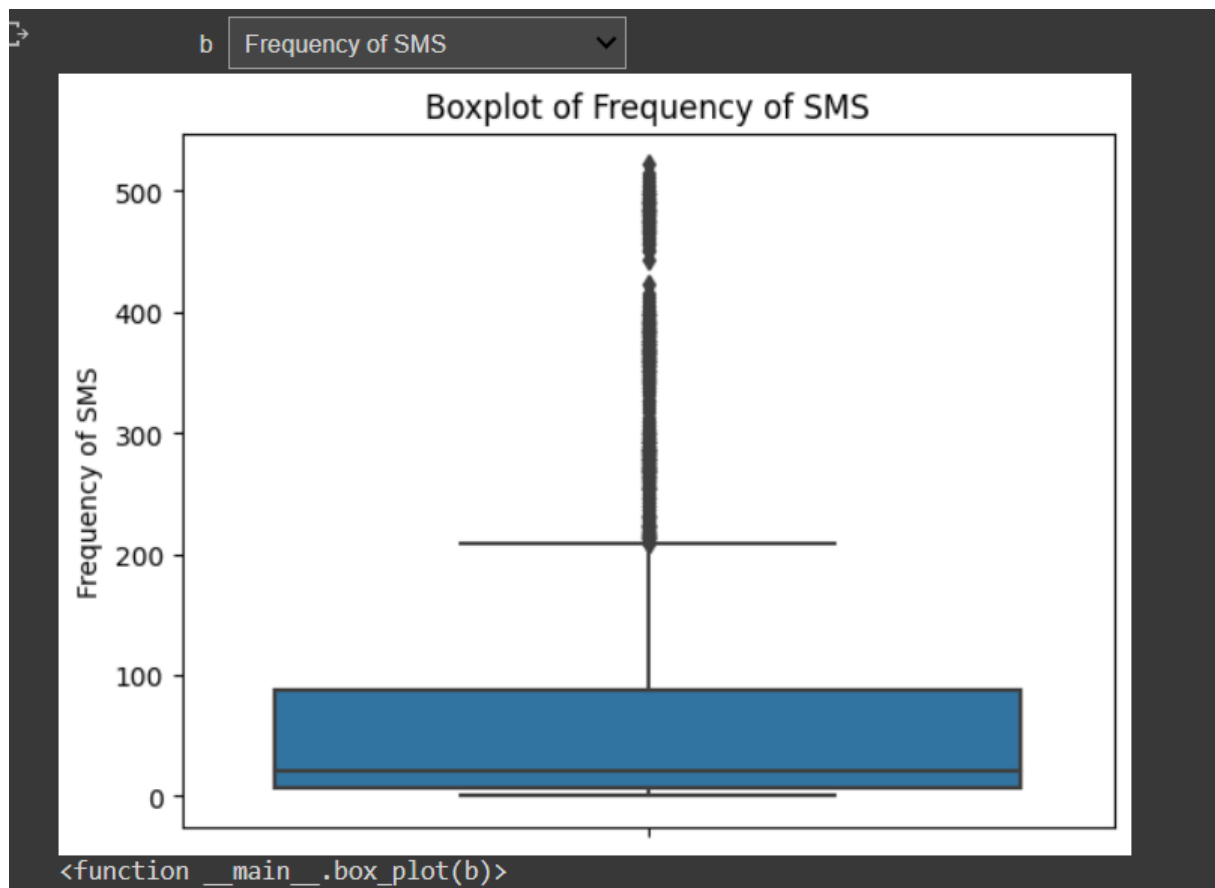
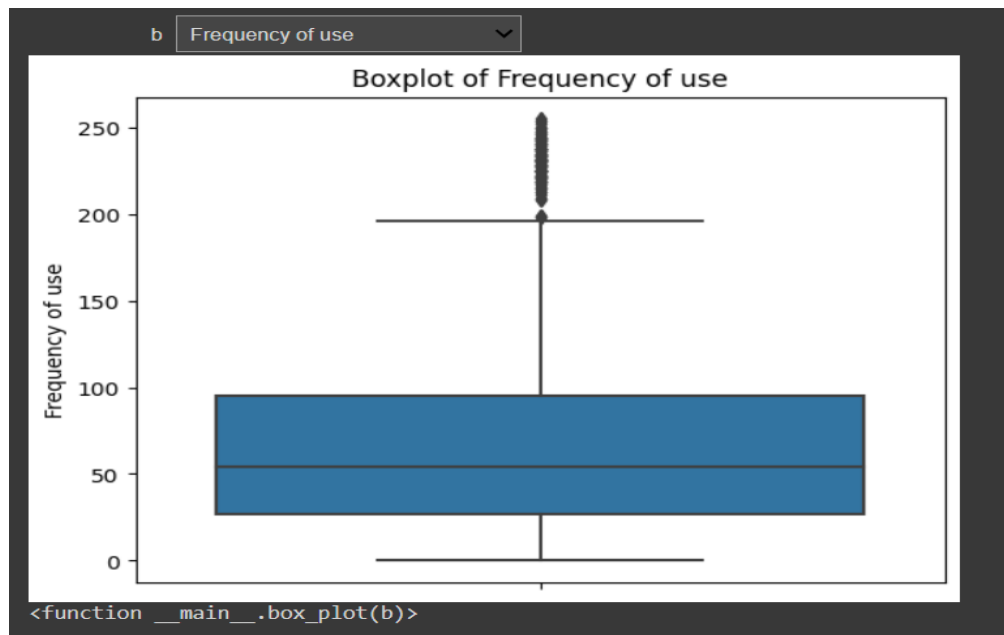
#Vérifions à présent l'existence ou non de valeurs aberrantes grâce aux boîtes à moustaches.

```
import seaborn as sns
import matplotlib.pyplot as plt

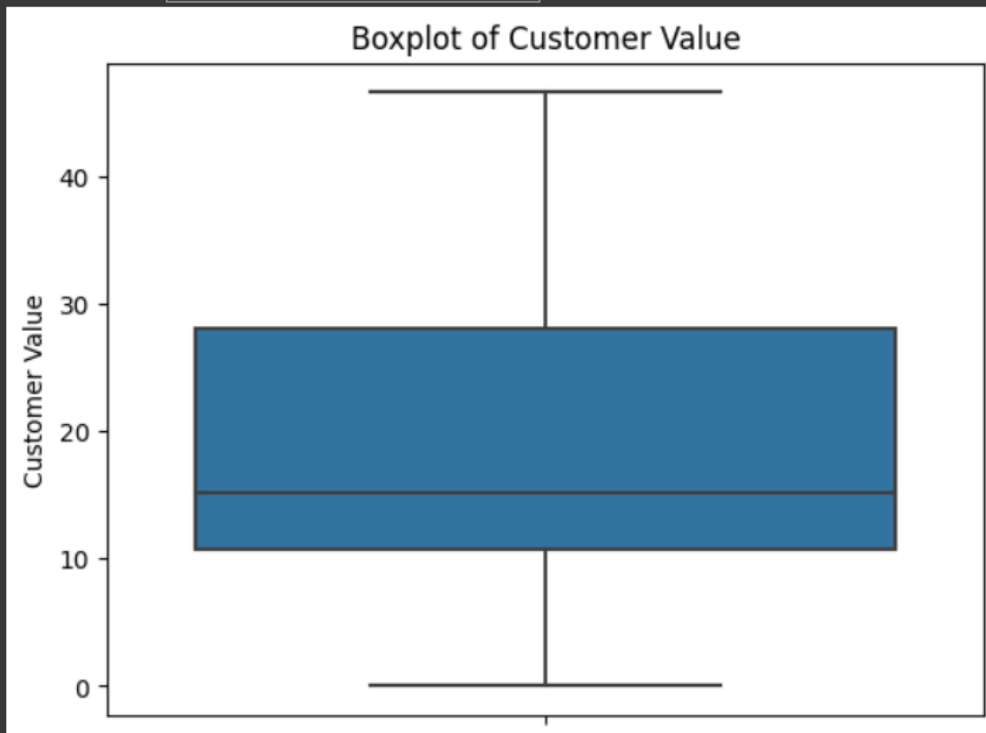
# Construction de la fonction de création de boîtes à moustaches
def box_plot(b):
    sns.boxplot(y=data[b])
    plt.title('Boxplot of ' + str(b))
    plt.show() # Affiche le graphique au lieu d'utiliser plt.shox()

# Interact
interact(box_plot, b=numerical_columns)
```

b Frequency of use

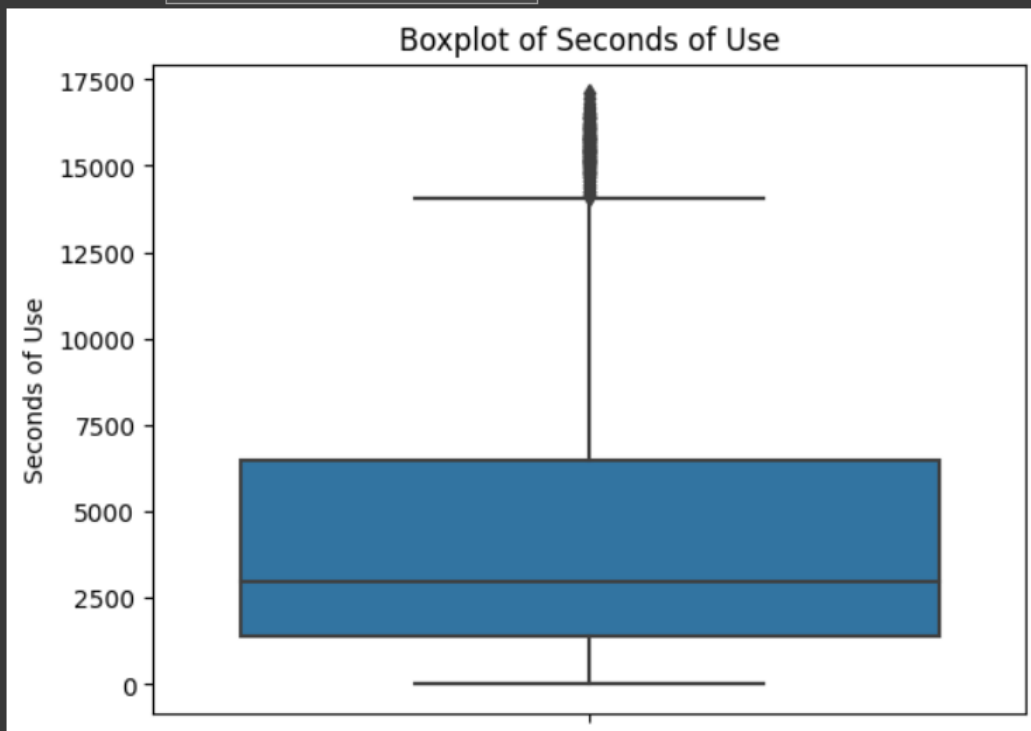


b Customer Value



```
<function __main__.box_plot(b)>
```

b Seconds of Use



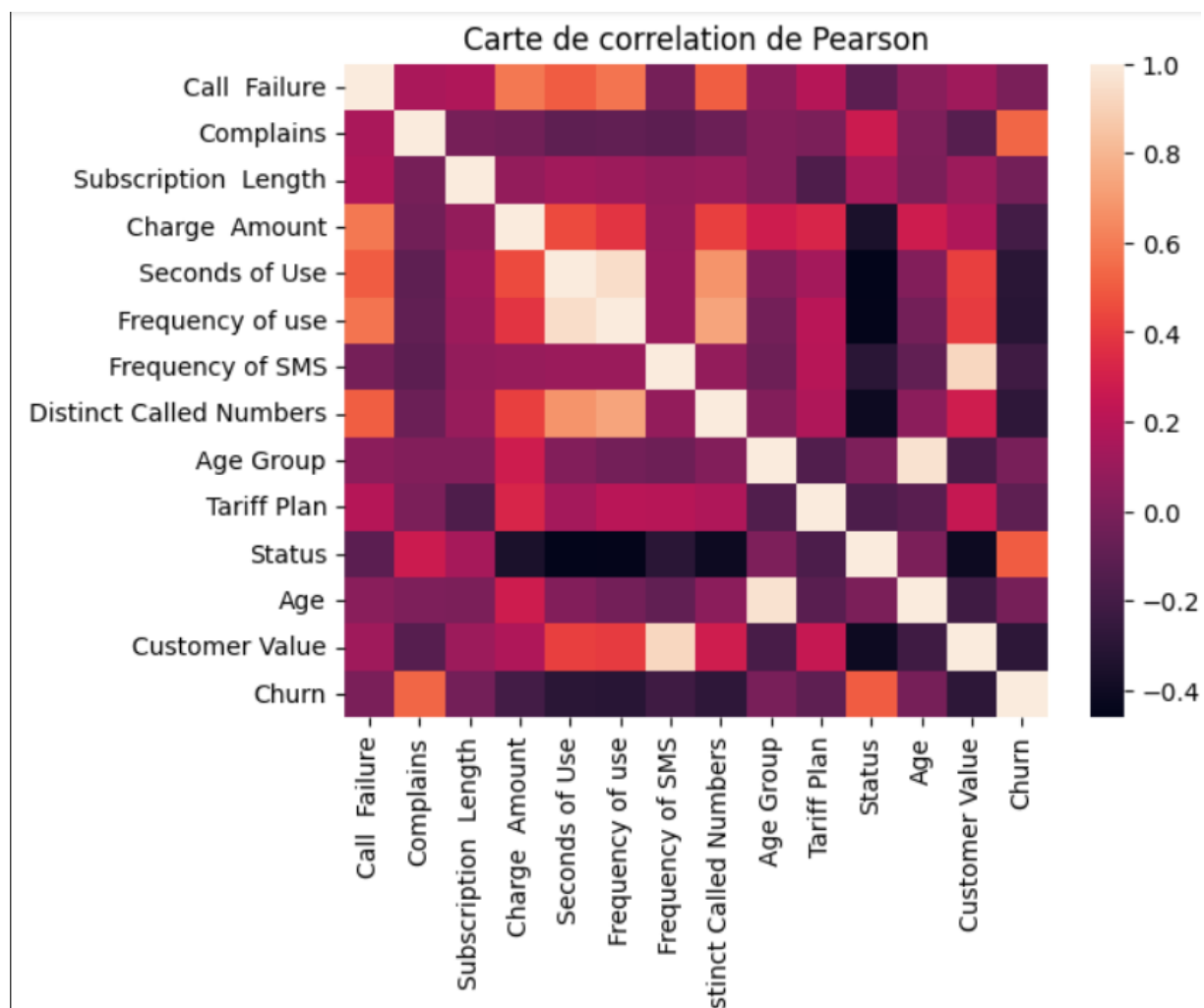
```
<function __main__.box_plot(b)>
```

On constate la présence de valeurs aberrantes d'où la nécessité de faire une normalisation.

MATRICE DE CORRELATION

#Matrice de corrélation : plus la couleur est proche de 1, plus les variables ont une forte corrélation.

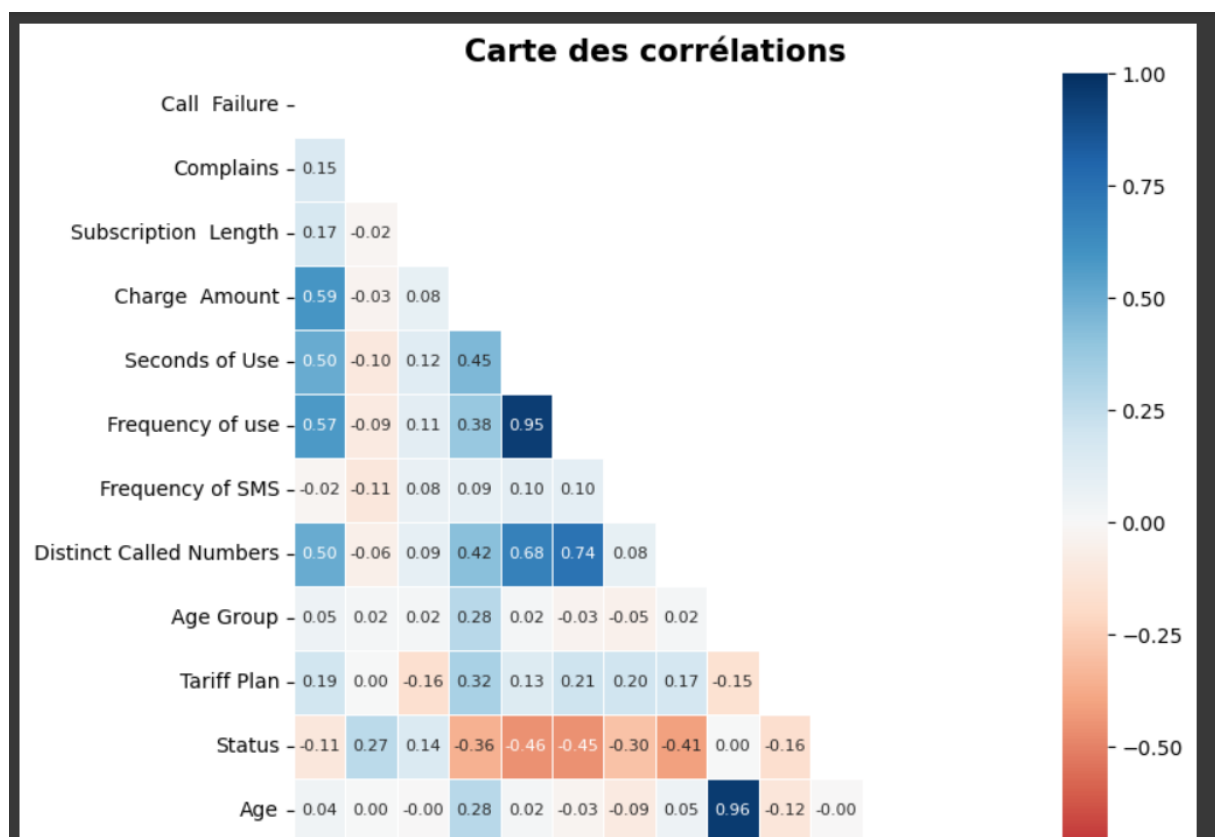
```
sns.heatmap(data.corr()).set_title('Carte de  
correlation de Pearson');
```

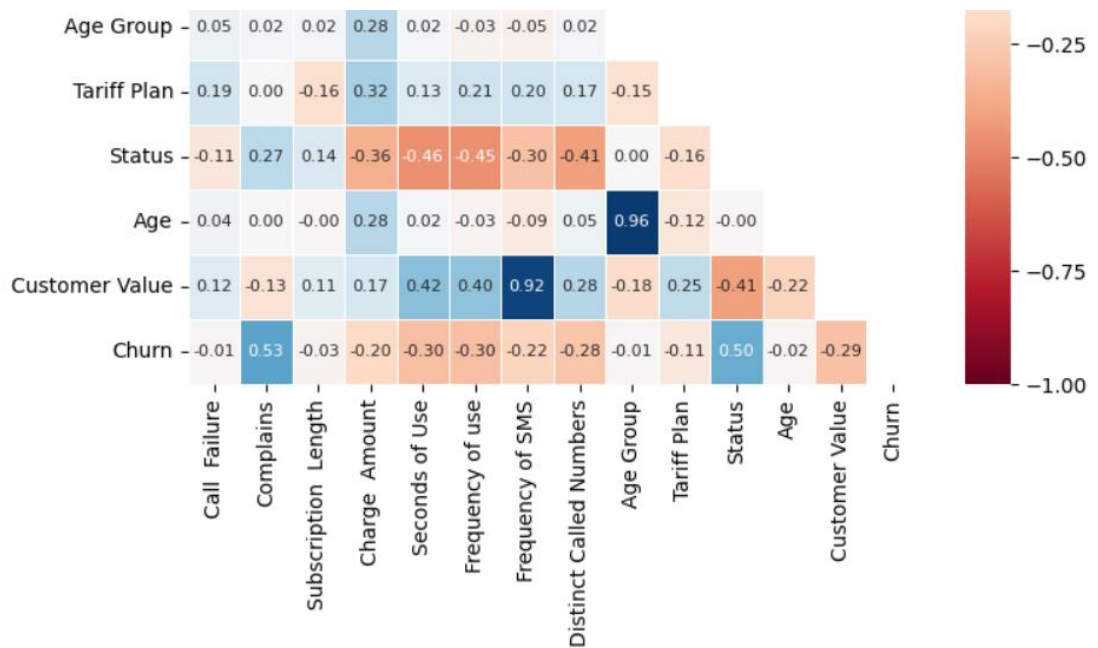


#La matrice optimale : masquer la seconde moitié de la matrice, créez un masque pour masquer le triangle inférieur


```
# La matrice optimale : masquer la seconde moitié de la matrice
# Créez un masque pour masquer le triangle inférieur
mask = np.triu(np.ones_like(data.corr(), dtype=bool))

# Créez la carte de chaleur avec des annotations ajustées
plt.figure(figsize=(8, 8))
sns.heatmap(data.corr(), mask=mask, center=0, cmap='RdBu',
            linewidths=0.5, annot=True, fmt=".2f", vmin=-1, vmax=1,
            annot_kws={"size": 8}) # Ajustez la taille de la police ici
plt.title('Carte des corrélations', fontsize=15, fontweight='bold')
plt.show()
```

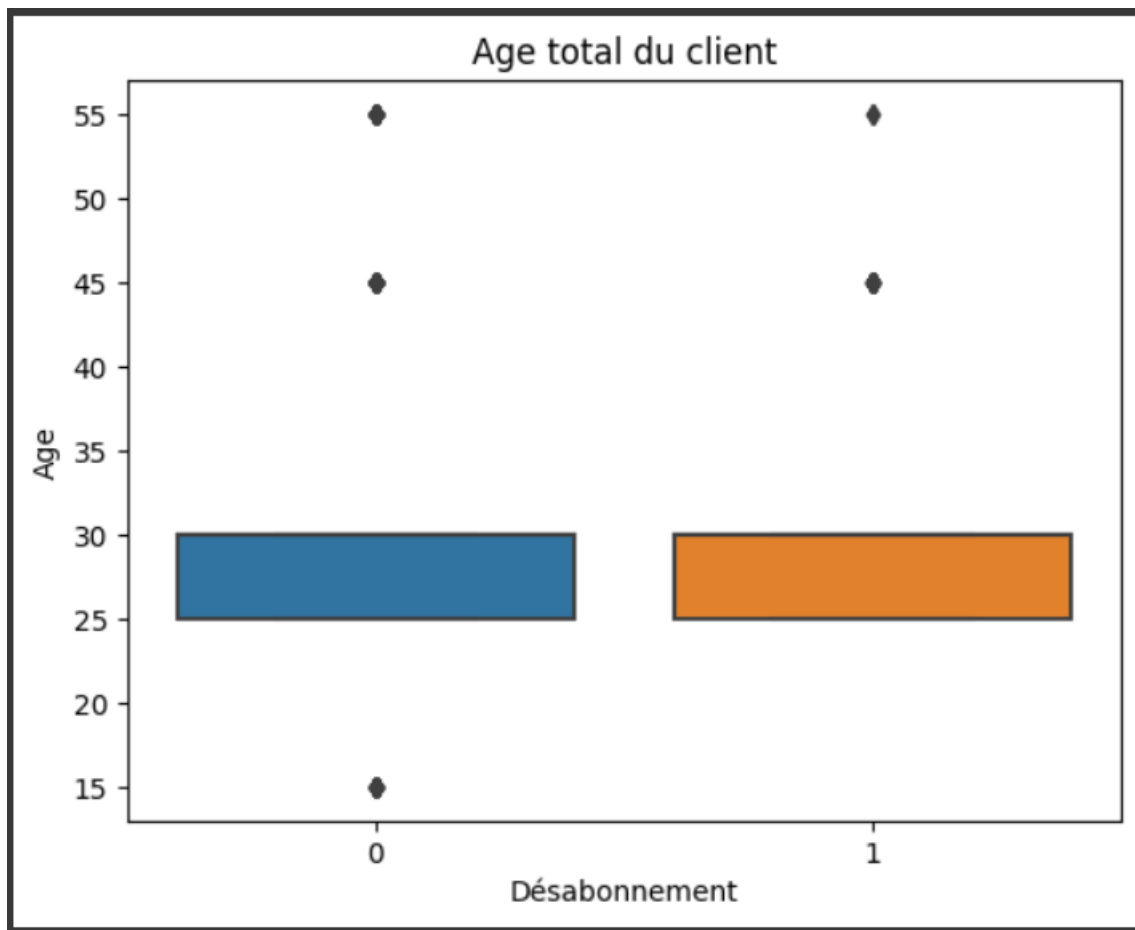




ANALYSE BIVARIEES

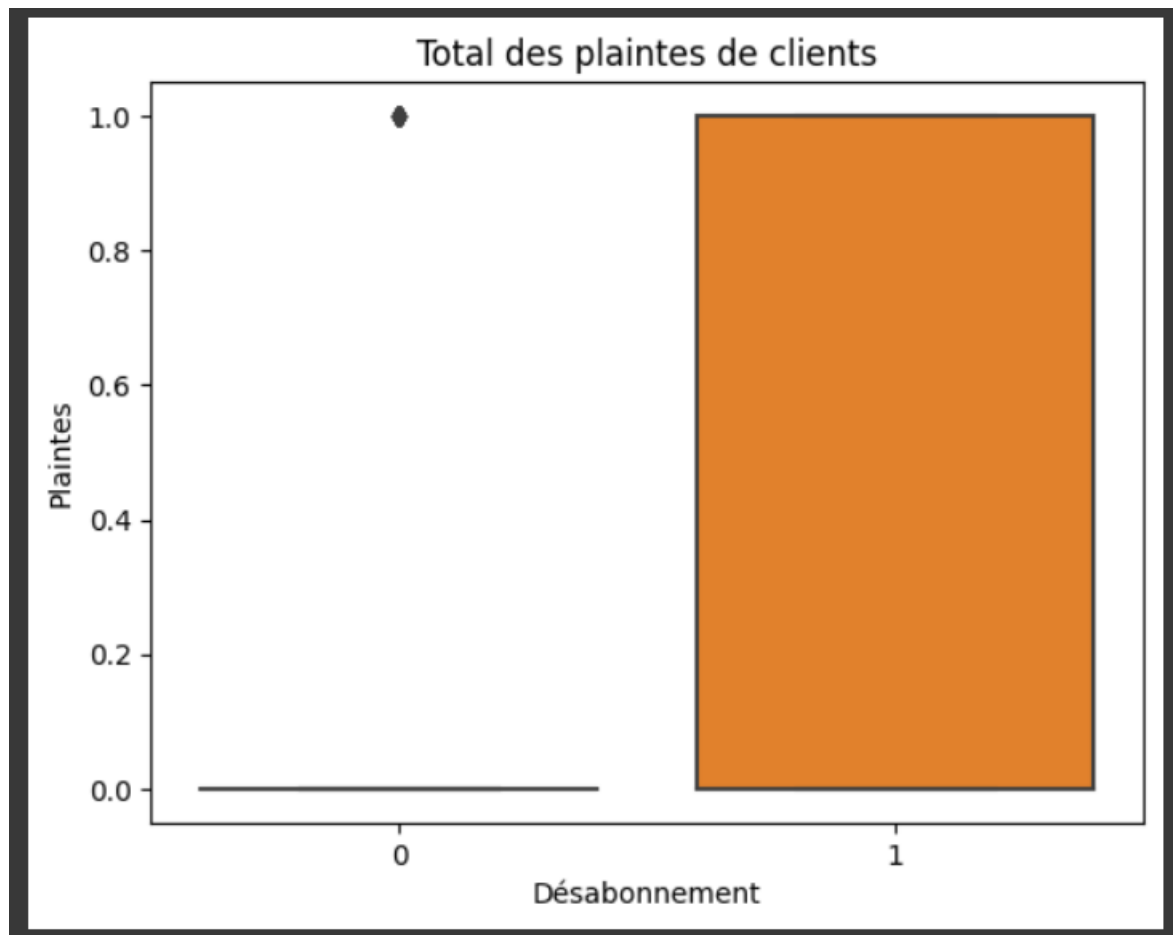
```
# Age vs Churn : on voit que l'age n'a aucun impact sur le desabonnement ou non du client.

sns.boxplot(x= 'Churn', y= 'Age', data=data)
plt.xlabel('Désabonnement')
plt.ylabel('Age')
plt.title('Age total du client')
plt.show()
```

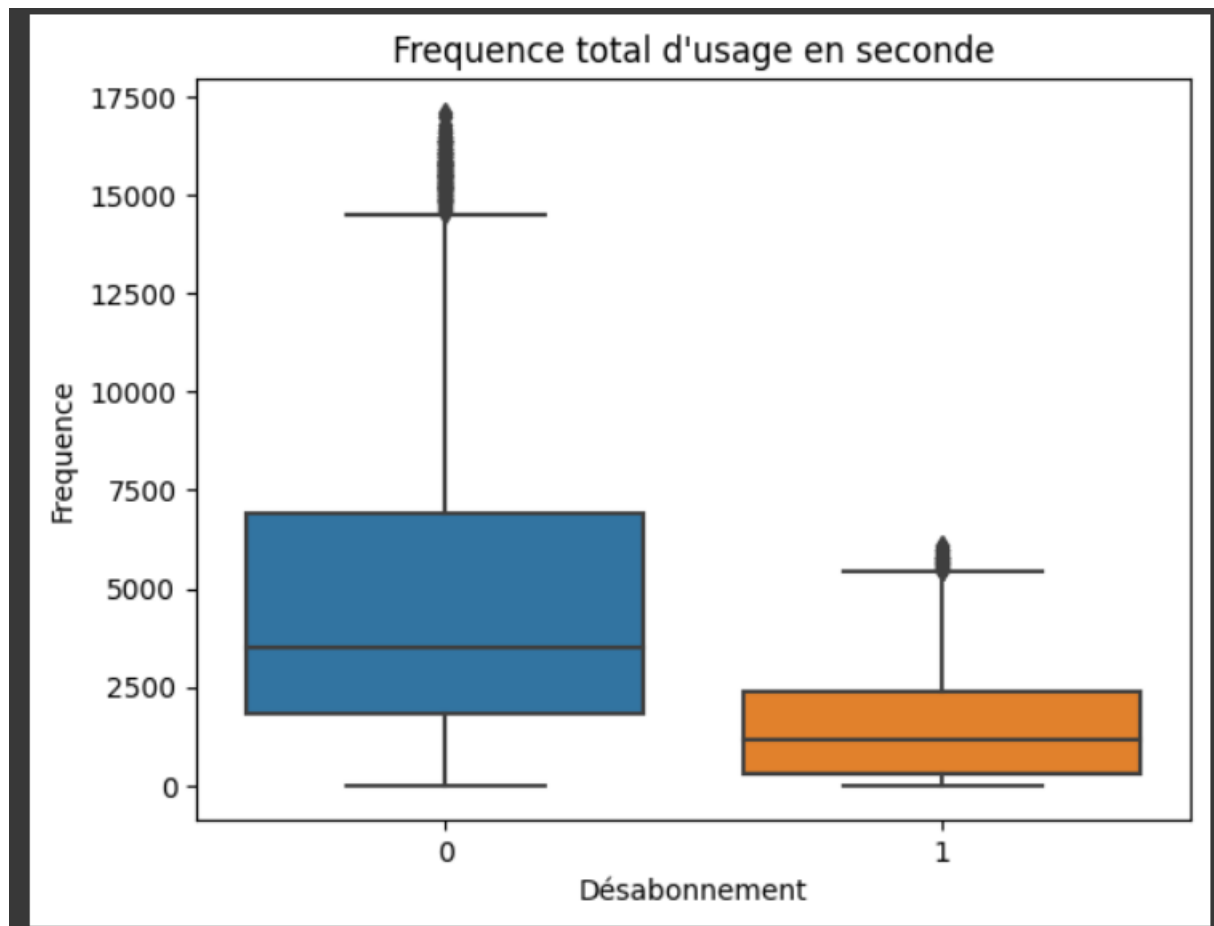


[28] # Complains vs Churn : on remarque que seul les client qui se sont plaints se sont desabonné.

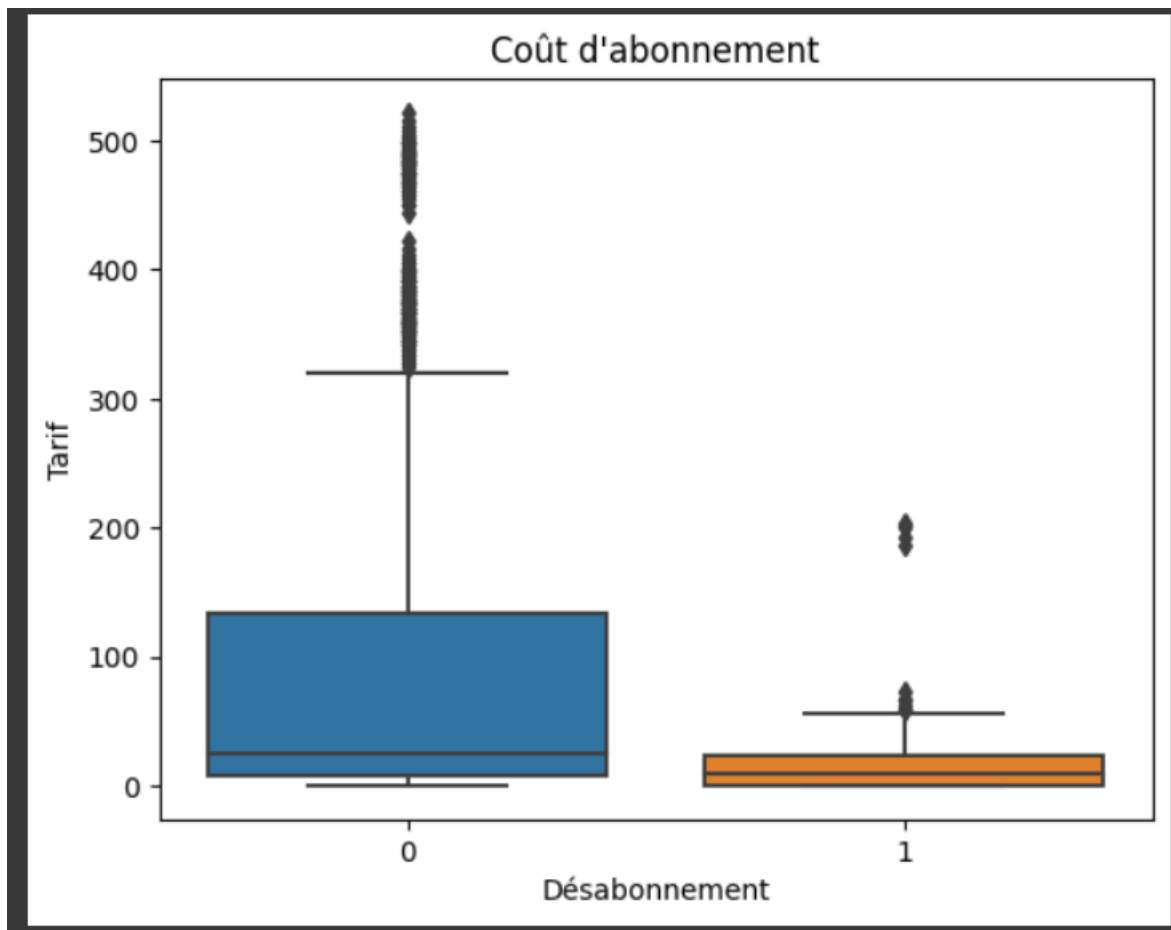
```
sns.boxplot(x= 'Churn', y= 'Complains', data=data)
plt.xlabel('Désabonnement')
plt.ylabel('Plaintes')
plt.title('Total des plaintes de clients')
plt.show()
```



```
[29] # Customer Value vs churn : les clients qui se désabonnent sont ceux qui ont une valeur moindre
sns.boxplot(x='Churn', y='Customer Value', data=data)
plt.xlabel('Désabonnement')
plt.ylabel('Valeur')
plt.title('Valeur du client')
plt.show()
```

```
# Frequency of SMS vs Churn : meme conclusion que l'analyse précédente
sns.boxplot(x='Churn', y='Frequency of SMS', data=data)
plt.xlabel('Désabonnement')
plt.ylabel('Tarif')
plt.title('Coût d\'abonnement')
plt.show()
```



A la suite de ces analyses, on relève 2 problèmes majeurs : Asymétrie des données et présence de valeurs aberrantes.

L'asymétrie dans les données peut affecter la performance des modèles, en particulier des modèles basés sur des hypothèses de normalité. Pour traiter l'asymétrie, vous pouvez envisager : La transformation et la standardisation des données.

Les valeurs aberrantes peuvent fausser les résultats et la robustesse du modèle. Voici quelques étapes pour traiter les valeurs aberrantes :

- Identification : Utilisez des méthodes comme les diagrammes de boîte à moustaches (box plots), les diagrammes de dispersion ou des méthodes statistiques pour identifier les valeurs aberrantes dans vos données.

- **Traitement** : Vous pouvez choisir de supprimer les valeurs aberrantes si elles sont vraiment des erreurs ou si elles ne représentent pas bien le comportement général. Sinon, vous pouvez utiliser des techniques de transformation comme la troncature ou la capping pour réduire leur impact.
- **Utilisation de modèles robustes** : Certains modèles sont moins sensibles aux valeurs aberrantes que d'autres. Par exemple, les modèles basés sur les arbres de décision et les modèles de forêt aléatoire peuvent mieux gérer les valeurs aberrantes car ils effectuent des coupures binaires répétées.

PRETRAITEMENT DE DONNEES

Faisons appel à MinMaxScaler est une fonction de normalisation de données

```
from sklearn.preprocessing import MinMaxScaler
```

##1-Gestion de valeurs manquantes : on remarque une absence de valeur manquantes

```
##1-Gestion de valeurs manquantes

#Créons une copie de notre base de données afin de la dataframe originelle (data) intacte .
data_copie = data.copy()
#Vérification d'absence ou non de valeur abérante
data_copie.isna().sum()
```

Call Failure	0
Complains	0
Subscription Length	0
Charge Amount	0
Seconds of Use	0
Frequency of use	0
Frequency of SMS	0
Distinct Called Numbers	0
Age Group	0
Tariff Plan	0
Status	0
Age	0
Customer Value	0
Churn	0
dtype: int64	

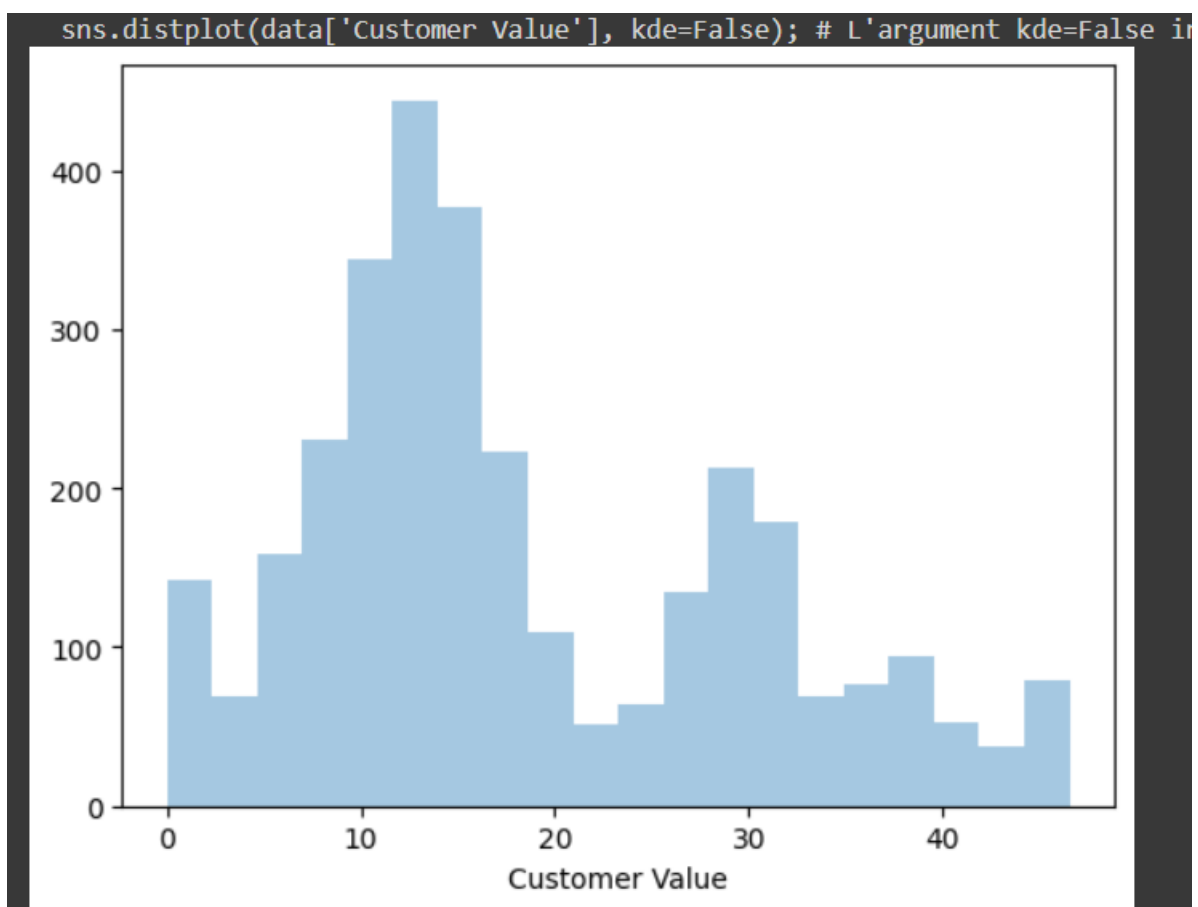
2-ENCODAGE DES VARIABLES : On remarque que toutes les variables contiennent uniquement des valeurs numériques et non des valeurs chaines de caractères. Donc on n'a pas besoin d'encoder quoique ce soit.

TRANSFORMATION DE VARIABLES

La variable Customer Value étant asymétrique, utilisons la transformation racine carrée pour réduire l'asymétrie.

```
# La variable Customer Value étant asymétrique , utilisons la transformation racine carrée pour réduire l'asymétrie .
data['Customer Value'] = np.sqrt(data['Customer Value'])
##Histogramme de la variable transformée : on remarque qu'il est moins asymétrique
sns.distplot(data['Customer Value'], kde=False); # L'argument kde=False indique que vous ne souhaitez pas tracer la fonction de densité estimée
```

Après la transformation de la variable customer Value, on obtient un histogramme moins asymétrique :



vérifions cela en calculant le coefficient d'asymétrie de cette variable

```
[37] # vérifions cela en calculant le coefficient d'asymétrie de cette variable
data['Customer Value'].skew() # on a passé de 1.4 à 0.6 donc une réduction de presque la moitié

0.6237200901031904
```

DIVISION DE NOS DONNEES : entraînement, validation et test.

```
[38] from sklearn.model_selection import train_test_split # Pour diviser notre dataset en 3 : Training , Validation et test
      from sklearn.utils import resample                 # rééchantillonner les données
```

#Données d'entraînement (60%) , de validation (20%) et de test (20%) : Double vérification ou double sécurité

#Les données de validation : pour pouvoir sélectionner le meilleur modèle sur lequel on va enfin tester

#Données d'entraînement : pour entraîner le modèle, Validation : pour sélectionner le meilleur modèle,

test : pour évaluer le modèle sélectionné. Donc il s'agit d'une double vérification.

```
x = data.drop('Churn', axis = 1) # Notre variable endogène ou indépendante (qui est la dataframe sans churn)
y = data['Churn']                # Notre variable indépendante ou expliquée (qui est churn)

seed=111 #La méthode seed() est utilisée pour initialiser le générateur de nombres aléatoires.

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.4, random_state = seed, stratify = y)

#stratify = y permet de s'assurer qu'on a la meme proportion de classe au niveau de la variable cible.

#Divisons le données de test créées en 2 parties égales pour former les données de validation et de test.

x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size = 0.5, random_state = seed, stratify = y_test)
```

NORMALISATION DES VARIABLES : Résolvons le problème de déséquilibre

Attaquons-nous au problème de déséquilibre de classes dans les données. Lorsqu'il ya une très grande différence entre le nombre d'observations dans chaque catégorie de la variable target, cela peut entraîner des erreurs de modélisation. Ici il ya plus de 80% des personnes qui n'ont pas résiliés leur abonnement contre 15% qui l'on fait. Il ya donc un très grand déséquilibre de classes. On peut donc utiliser le ré échantillonnage pour rééquilibrer les classes. Soit on fait du sur échantillonnage dans la classe minoritaire(modalité1), soit on fait du sous-échantillonnage dans la classe majoritaire(modalité2).

```

#Méthode de suréchantillonnage de la classe minoritaire.

X2 = x_train          #Création de la copie de nos données d'entraînement

X2['Churn'] = y_train.values #Ajout la variable cible à cette copie

minority = X2[X2.Churn == 1] #Filtrage pour obtenir uniquement les observations minoritaires
majority = X2[X2.Churn == 0] #Filtrage pour obtenir uniquement les observations majoritaires

minority_upsampled = resample(minority, replace = True, n_samples = len(majority), random_state = seed) # Application du suréchantillonnage
upsampled = pd.concat([majority, minority_upsampled]) # On concatene les DataFrames majority et minority_upsampled
#Nous créons un nouvel ensemble de données "upsampled" où les classes sont maintenant équilibrées

```

	Call Failure	Complains	Subscription Length	Charge Amount	Seconds of Use	Frequency of use	Frequency of SMS	Distinct Called Numbers	Age Group	Tariff Plan	Status	Age	Customer Value	Churn
2041	0	0	24	0	520	4	14	1	4	1	2	45	6.935416	0
964	9	0	37	0	15485	182	150	30	2	1	1	25	37.148553	0
1600	3	0	37	0	4180	71	30	19	3	1	1	30	17.030561	0
1697	3	0	13	1	2433	33	0	13	3	1	1	30	9.931767	0
163	5	0	42	2	3175	29	493	12	3	1	1	30	45.827503	0
...
349	0	1	3	0	1228	9	0	1	2	1	1	25	7.460898	1
2078	0	1	34	0	0	0	0	0	3	1	2	30	0.000000	1
2074	8	0	34	0	5513	92	19	22	2	1	1	25	18.377296	1
2777	16	1	38	0	2763	51	50	36	4	1	2	45	13.976766	1
2999	14	1	12	3	2502	36	15	9	3	1	1	30	12.709052	1

3186 rows x 14 columns

Vérification de la résolution du problème de déséquilibre

```

upsampled['Churn'].value_counts(normalize = True) # Equilibre parfait avec 50/50

```

```

0    0.5
1    0.5
Name: Churn, dtype: float64

```

Données d'entrainement sur la base de méthode de suréchantillonnage de la classe minoritaire

```

# Données d'entrainement sur la base de methode de surechantillonnage de la classe minoritaire
x_train_up = upsampled.drop('Churn', axis=1)
y_train_up = upsampled['Churn']

```

```

# Résolution du prob de desequilibre méthode de sous-échantillonnage de la classe majoritaire.
majority_downsampled = resample(majority, replace = False, n_samples = len(minority), random_state = seed)
downsampled = pd.concat([minority, majority_downsampled])
downsampled

#NB: replace = False signifie tirage sans remise : on aura moins d'observations

```

	Call Failure	Complains	Subscription Length	Charge Amount	Seconds of Use	Frequency of use	Frequency of SMS	Distinct Called Numbers	Age Group	Tariff Plan	Status	Age	Customer Value	Churn
2477	16	1	41	0	2753	58	52	38	4	1	2	45	14.151855	1
2122	28	0	35	0	1260	53	23	19	3	1	2	30	12.021647	1
1068	0	0	35	0	0	0	0	0	2	1	2	25	0.000000	1
672	25	1	33	0	1145	48	25	20	2	1	2	25	12.891276	1
1028	0	0	36	0	133	2	0	2	3	1	2	30	2.323790	1
...
883	31	1	32	4	16785	249	8	80	2	1	1	25	28.328960	0
125	0	0	32	0	2780	19	0	12	3	1	1	30	10.581115	0
2188	12	0	31	3	1680	45	205	15	3	2	1	30	29.816103	0
1147	6	0	16	3	2608	42	29	19	3	1	1	30	14.899664	0
225	0	0	30	0	2760	20	0	10	3	1	1	30	10.545141	0

594 rows x 14 columns

#Vérification

```

#Vérification
downsampled['Churn'].value_counts(normalize=True)

1    0.5
0    0.5
Name: Churn, dtype: float64

```

Données d'entrainement sur la base de méthode de sous échantillonnage de la classe majoritaire

```

[49] # Données d'entrainement sur la base de methode de souséchantillonnage de la classe majoritaire
x_train_down = downsampled.drop('Churn', axis=1)
y_train_down = downsampled['Churn']

Redéfinissons le X_train et le y_train

```

```

[50] y_train = x_train['Churn'] # Variable expliquée
x_train = x_train.drop('Churn', axis=1) #Variable explicative sans churn

```

```
#Définition des données d'entraînement
#Choix possibles de données : (X_train, y_train), (X_train_up, y_train_up) et (X_train_down, y_train_down)
#Choix numéro 1 : les données de surechantillonnage

train_features = x_train_up
train_labels = y_train_up

#Pour avoir la possibilité de pouvoir utiliser le type de données qu'on veut en changeant juste le (train_features et train_labels)
```

NORMALISATION DES DONNEES :

Après le problème de déséquilibre de classes, passons à la prochaine étape qui est celle de normalisation des données.

La standardisation consiste à remplacer les valeurs réelles par le **z_score** = **((Valeur_de_la_variable - Moyenne_de_la_variable)/(ecart_type_de_la_variable))**

```
#Normalisation des variables independantes des differents ensembles de données
# On peut choisir de standardiser au lieu de normaliser
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler() # Appel à la fonction MinMaxScaler pour la normalisation

#Entrainons l'algo à partir des données d'entraînement

mod_scaler = scaler.fit(train_features)

train_features = mod_scaler.transform(train_features) # Transformation des différents ensembles de données
x_val = mod_scaler.transform(x_val) # Pareil pour mes données de validation
x_test = mod_scaler.transform(x_test) # Pareil pour mes données de test

# Retransformation en dataframe

train_features = pd.DataFrame(train_features, columns = x.columns)
x_val= pd.DataFrame(x_val, columns = x.columns)
x_test= pd.DataFrame(x_test, columns = x.columns)
```

#Vérifions que Toutes les valeurs sont maintenant entre 0 et 1

[illegible]

MODELISATION : Après la normalisation, passons maintenant à la modélisation.

Sélection des variables prédictrices pour notre algorithme Forêt aléatoire

```
# Sélection des variables prédictrices pour notre algorithme
# Forêt aléatoire
rf = RandomForestClassifier() # Création de la classe RandomForestClassifier car on est en face d'un problème de classification

rf.fit(train_features, train_labels) # entraîner avec les données d'entraînement

print(classification_report(y_val, rf.predict(x_val))) # Afficher le rapport de classification qui va permettre de visualiser les métriques
```

	precision	recall	f1-score	support
0	0.97	0.97	0.97	531
1	0.85	0.84	0.84	99
accuracy			0.95	630
macro avg	0.91	0.91	0.91	630
weighted avg	0.95	0.95	0.95	630

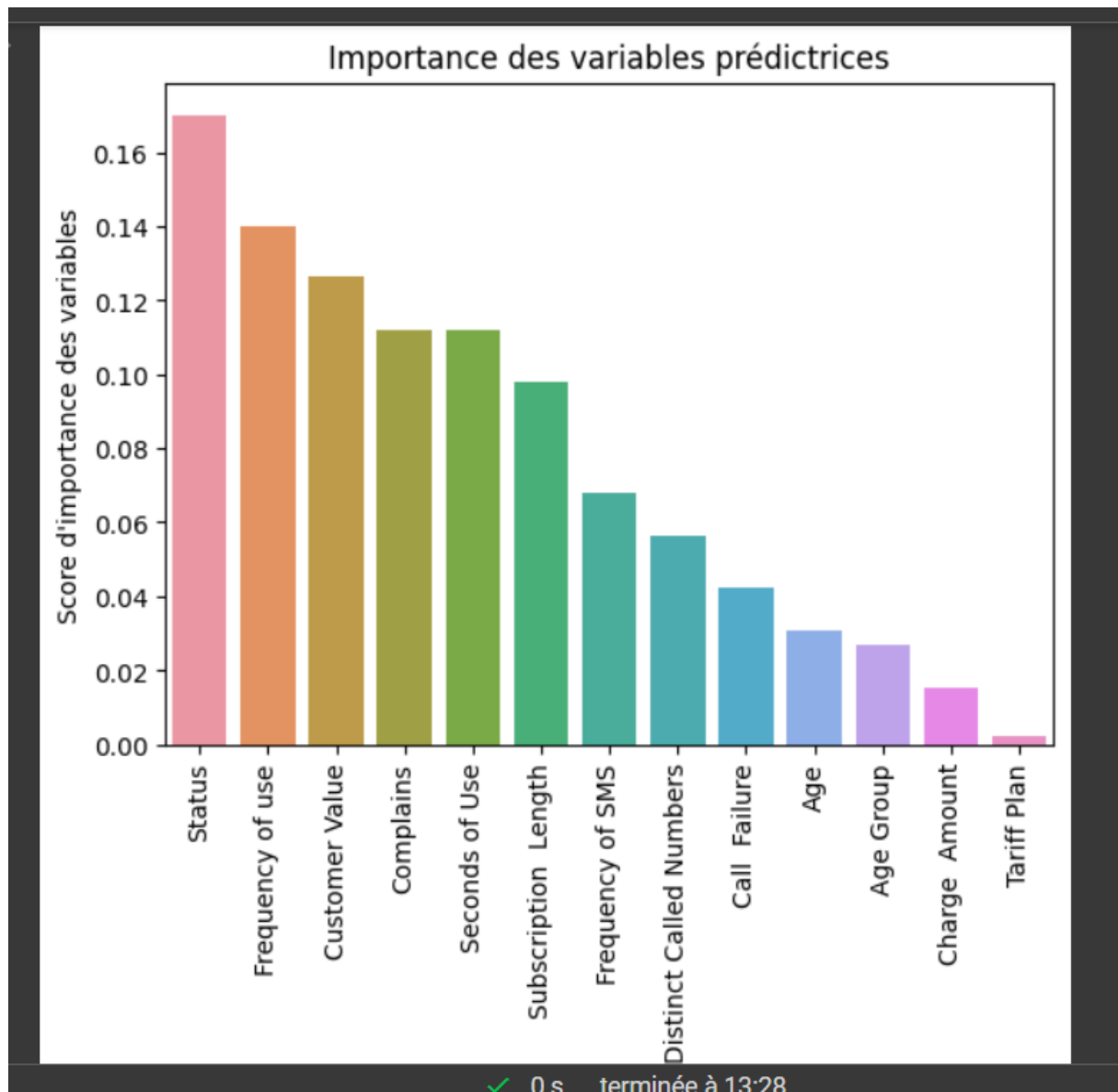
Remarque : Le modèle se comporte très bien face aux observations de la classe 0 malgré le problème de déséquilibre réglé.

Importance des variables indépendantes

```
# Importance des variables indépendantes

vars_imp = pd.Series(rf.feature_importances_, index = train_features.columns).sort_values(ascending = False)

sns.barplot(x=vars_imp.index, y=vars_imp)
plt.xticks(rotation = 90) # Afficher le nom des variables en vertical
plt.xlabel("Variables")
plt.ylabel("Score d'importance des variables")
plt.title("Importance des variables prédictrices")
# Afficher le graphique agrandi
plt.show()
```

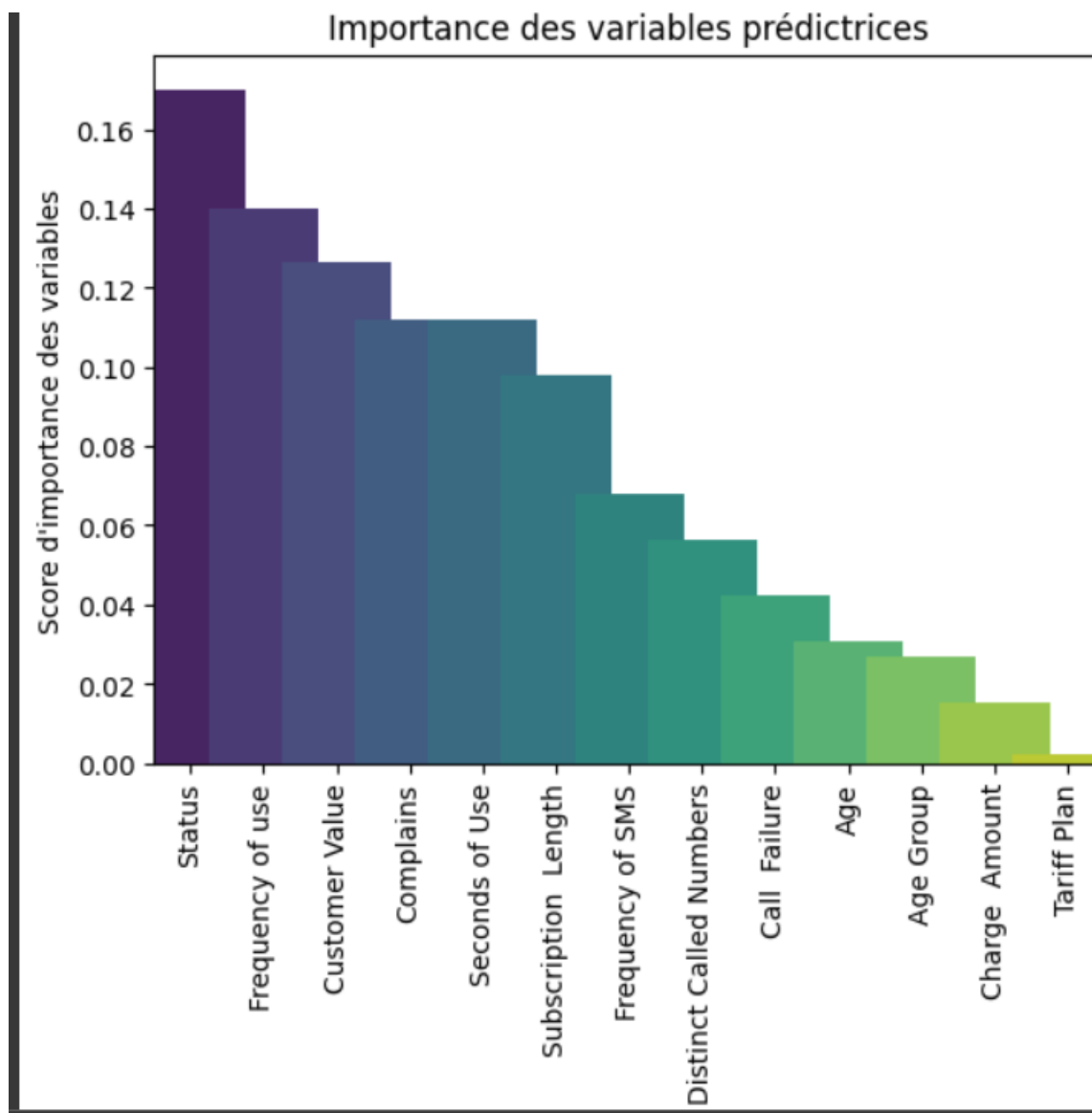


Autre façon de représenter l'importance des variables indépendante

```
# Autre representation Importance des variables independante
vars_imp = pd.Series(rf.feature_importances_, index = train_features.columns).sort_values(ascending = False)

sns.barplot(x=vars_imp.index, y=vars_imp, palette="viridis", ci=None, width=1.5) # Modifiez la valeur de width selon vos préférences

plt.xticks(rotation = 90)
plt.xlabel("Variables")
plt.ylabel("Score d'importance des variables")
plt.title("Importance des variables prédictives")
# Afficher le graphique agrandi
plt.show()
```



#Affichage des variables importantes (vars_imp) selon leur coefficients d'importance.


```
#Affichage des variables importantes (vars_imp) selon leur coefficients d'importance.
vars_imp

Status          0.170156
Frequency of use 0.139849
Customer Value   0.126402
Complains        0.111941
Seconds of Use   0.111772
Subscription Length 0.097961
Frequency of SMS 0.068081
Distinct Called Numbers 0.056260
Call Failure     0.042551
Age              0.030749
Age Group        0.026655
Charge Amount    0.015395
Tariff Plan      0.002229
dtype: float64
```

#Rétention de variables dont le seuil d'importance est inférieur à 0.03 pour nos algorithmes

```
#Rétention de variables dont le seuil d'importance est inférieur à 0.03 pour nos algorithmes

seuil = 0.03

vars_selected = vars_imp[vars_imp>0.03].index.to_list()

train_features = train_features[vars_selected]

x_val = x_val[vars_selected]
x_test = x_test[vars_selected]
```

#Nombre de variables prédictrices

```
#Nombre de variables prédictrices

len(vars_selected) # ou len(train_features.columns)

10
```

Finalement, 9 variables iront en modélisation

MODELISATION : Nous aurons à utiliser différents types d'algorithmes

*La régression logistique * la Modèle de foret aléatoire *

1-REGRESSION LOGISTIQUE

```
#Dictionnaires des hyperparametres

param_grid = {'C':[0.001,0.01,1,10,50,100,500]}    # définition de dictionnaire des hyperparametres

#Objet GridSearchCV

grid_logreg_class = GridSearchCV(estimator=LogisticRegression(random_state=seed, max_iter=500),
                                param_grid=param_grid,
                                scoring='f1',
                                cv=5)

#Entrainement de l'algorithme crée :

logreg_model = grid_logreg_class.fit(train_features,train_labels)

#Meilleur score , meilleur hyperparamètre

print(round(logreg_model.best_score_,3))

print(logreg_model.best_estimator_)
```

0.868
LogisticRegression(C=1, max_iter=500, random_state=111)

On obtient le meilleur score qui est de 0.874 et le meilleur C=1

Ce code crée un objet `grid_logreg_class` en utilisant la classe `GridSearchCV` de la bibliothèque `scikit-learn`.

`GridSearchCV` est une technique qui permet de rechercher les meilleurs hyperparamètres pour un modèle donné.

Voici une explication détaillée de chaque argument passé à `GridSearchCV` :

`estimator=LogisticRegression(random_state=seed, max_iter=500)` : C'est l'estimateur de base pour lequel nous recherchons les meilleurs hyperparamètres.

Les hyperparamètres : `random_state` (pour la reproductibilité) et `max_iter` (le nombre maximal d'itérations pour la convergence de l'algorithme de la régression logistique).

`param_grid=param_grid` : C'est le dictionnaire qui spécifie les combinaisons d'hyperparamètres à explorer.

`scoring='f1'` : C'est la métrique d'évaluation utilisée pour comparer les performances des différentes combinaisons d'hyperparamètres.

`cv=5` : C'est la stratégie de validation croisée utilisée pour évaluer les performances des modèles. Ici, la validation croisée à 5 plis est utilisée, ce qui signifie que les données seront divisées en 5 ensembles de formation et de test pour évaluer les performances de chaque modèle.

En résumé, ce code crée un objet GridSearchCV pour rechercher les meilleurs hyper paramètres pour un modèle de régression logistique en utilisant le score F1 comme métrique d'évaluation. La grille des hyper paramètres à explorer est spécifiée dans param_grid, et la validation croisée à 5 plis est utilisée pour évaluer les performances. Une fois que grid_logreg_class est créé, vous pouvez l'utiliser pour effectuer la recherche des hyper paramètres et obtenir le meilleur modèle avec les paramètres optimaux.

GENERALISATION : Le modèle a un bon score d'entraînement, maintenant évaluons le sur nos données de validations afin de voir sa capacité de généralisation.

```
[61] # Fonction d'évaluation de la performance du modèle

from sklearn.metrics import classification_report

def model_evaluation(model, features, labels):
    pred = model.predict(features)
    print(classification_report(labels, pred))
```

```
# Evaluation du modèle de regression logistique
model_evaluation(logreg_model,x_val,y_val)
```

	precision	recall	f1-score	support
0	0.97	0.84	0.90	531
1	0.50	0.88	0.64	99
accuracy			0.84	630
macro avg	0.74	0.86	0.77	630
weighted avg	0.90	0.84	0.86	630

On obtient un tableau de classification avec précision globale =0.85 , f1 score sur la classe positive qui est de 0.65 et le modèle qui se comporte bien avec la classe 0.

RFE : Appliquons l'algorithme rfe (recursive feature eliminator) afin de voir s'il garde les mêmes performances lorsqu'on réduit le nombre de prédicteurs.

```
[63] #Création de fonction de construction de modèle avec l'utilisation de l'algorithme RFE
```

```
def model_with_rfe(model):  
    rfe_model = RFE(estimator = model, verbose = 0)  
    rfe_model.fit(train_features, train_labels)  
    mask = rfe_model.support_  
    reduced_x = train_features.loc[:, mask]  
    print(reduced_x.columns)  
    return rfe_model
```

```
#Logistic Regression RFE
```

```
rfe_logreg_model = model_with_rfe(logreg_model.best_estimator_)  
  
rfe_logreg_model
```

```
Index(['Frequency of use', 'Complains', 'Frequency of SMS',  
      'Distinct Called Numbers', 'Call Failure'],  
      dtype='object')
```

```
RFE  
  estimator: LogisticRegression  
    LogisticRegression  
    LogisticRegression(C=1, max_iter=500, random_state=111)
```

Il a utilisé 4 variables prédictrices au lieu des 9 initiales.

#Evaluation du modèle de regression logistique avec RFE

```
#Evaluation du modèle de regression logistique avec RFE
```

```
model_evaluation(rfe_logreg_model, x_val, y_val)
```

	precision	recall	f1-score	support
0	0.97	0.81	0.88	531
1	0.46	0.85	0.59	99
accuracy			0.82	630
macro avg	0.71	0.83	0.74	630
weighted avg	0.89	0.82	0.84	630

REF a réduit le nombre de predicteurs de 9 à 4 mais n'a pas amélioré la performance du modèle. Ce qui montre la puissance du modèle RFE

MODELE DE FORET ALEATOIRE

C'est un ensemble d'arbres de décision : les paramètres les plus importants sont : le nombre d'arbres dans la forêt, la profondeur maximum d'un arbre de décision. Ce sont ces deux paramètres que nous allons régler.

```
#Dictionnaires des hyperparamètres

param_grid_rf = {'n_estimators':[10,50,100,500,1000], 'max_depth':[3,5,10,20, None],
                 'max_depth':[3,5,10,20,None]}

#Objet GridSearchCV

grid_rf_class = GridSearchCV(estimator = RandomForestClassifier(random_state = seed),
                             param_grid = param_grid_rf,
                             scoring = 'f1',
                             cv = 5)

#Entraînement de l'algorithme

rf_model = grid_rf_class.fit(train_features , train_labels)

# Meilleurs scores et meilleurs hyperparamètres
print(round(rf_model.best_score_, 3))
print(rf_model.best_estimator_)

0.984
RandomForestClassifier(max_depth=20, n_estimators=50, random_state=111)
```

Le meilleur score trouvé lors du processus d'ajustement est de 0.981 et le modèle avec 1000 arbres (n_estimators=1000) et une graine aléatoire de 111 (random_state=111) est celui qui a produit ce f1.

```
# Evaluation du modèle de forêt aléatoire

model_evaluation(rf_model.best_estimator_,x_val,y_val)
```

	precision	recall	f1-score	support
0	0.97	0.98	0.97	531
1	0.86	0.84	0.85	99
accuracy			0.95	630
macro avg	0.92	0.91	0.91	630
weighted avg	0.95	0.95	0.95	630

Comparé aux valeurs de la régression logistique, modèle de forêt aléatoire semble plus efficace car a des f1 scores plus élevés.

```
#Random Forest avec RFE

rfe_forest_model = model_with_rfe(rf_model.best_estimator_)
rfe_forest_model
```

Index(['Status', 'Frequency of use', 'Customer Value', 'Complains',
 'Seconds of Use'],
 dtype='object')

RFE

estimator: RandomForestClassifier

RandomForestClassifier

RandomForestClassifier(max_depth=20, n_estimators=50, random_state=111)

Le RFE de forêt aléatoire affiche uniquement les variables considérés : (4)'Status', 'Seconds of Use', 'Customer Value', 'Frequency of use'. Mais les hyper paramètres n'ont pas changés.

```
#Evaluation de modèle de forêt aléatoire avec RFE :
model_evaluation(rfe_forest_model,x_val,y_val)
```

	precision	recall	f1-score	support
0	0.95	0.96	0.95	531
1	0.75	0.71	0.73	99
accuracy			0.92	630
macro avg	0.85	0.83	0.84	630
weighted avg	0.92	0.92	0.92	630

max_depth=20: Cela signifie que les arbres de la forêt aléatoire auront une profondeur maximale de 20 niveaux. La profondeur d'un arbre correspond au nombre de niveaux dans l'arborescence de décision. Des arbres plus profonds peuvent capturer des relations plus complexes dans les données d'entraînement, mais cela peut également augmenter le risque de sur ajustement (overfitting).

CONCLUSION : Au vu de ces résultats nous retiendrons le modèle de forêt aléatoire sans RFE. Donc on peut maintenant appliquer ce modèle sur les données de test.



#Evaluation du model de foret aléatoire sur nos données de tests

```
model_evaluation(rf_model.best_estimator_,x_test,y_test)
```



	precision	recall	f1-score	support
0	0.97	0.97	0.97	531
1	0.85	0.82	0.84	99
accuracy			0.95	630
macro avg	0.91	0.90	0.90	630
weighted avg	0.95	0.95	0.95	630

On remarque un résultat sensiblement égal à celui des données de validations mais en moins bon. Mais l'accuracy reste le même.

CONCLUSION DES ANALYSES ET MODELISATION : NOUS AVONS POUR CE PROJET MIS EN CONCURRENCE DEUX CÉLÈBRES MODÈLES DE MACHINE LEARNING (la régression logistique et le classificateur de forêt aléatoire) ET CELUI QUE NOUS AVONS RETENU POUR L'ÉVALUATION DE NOS DONNÉES DE TESTS EST LE CLASSIFICATEUR DE FORÊTS ALÉATOIRE CAR IL PRÉSENTE UNE MEILLEURE PERFORMANCE.

CREATION DE L'APPLICATION DU MODÈLE GRÂCE À STREAMLIT AVEC PYTHON DANS VISUAL STUDIO

```
import streamlit as st
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error

#####

st.title("Application Iran_still_Call")
st.subheader("Auteur: KHALID")
```

```

st.write("Cette application donne la possibilité de voir
la distribution des abonnés"
        " à travers un histogramme "
        " en définissant leurs valeurs "
        " entre 5 et 500"
        )

data = pd.read_csv(r'C:/Users/Electro
Depot/Desktop/APPRENTISSAGE/Bases de données/Customer
Churn.csv')

##
data = pd.read_csv(r'C:/Users/Electro
Depot/Desktop/APPRENTISSAGE/Bases de données/Customer
Churn.csv')
data = pd.DataFrame(data, columns=["Frequency of use",
"Frequency of SMS", "Customer Value", "Seconds of Use"])
st.title("Pairplot des variables")
# Utiliser Seaborn pour créer un pairplot
pairplot = sns.pairplot(data=data)

# Afficher le pairplot dans Streamlit
st.pyplot(pairplot)

##

data_1 = pd.DataFrame(data, columns=["Customer Value"])
data_2 = pd.DataFrame(data, columns=["Frequency of use",
"Frequency of SMS", "Customer Value", "Seconds of Use"])
st.title("Matrice de corrélation des variables")

# Calculer la matrice de corrélation
corr_matrix = data_2.corr()
# Créer une figure Matplotlib
fig, ax = plt.subplots(figsize=(10, 8))
# Créer la heatmap et la placer dans l'axe
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm',
fmt=".2f", ax=ax)
# Afficher la figure dans Streamlit
st.pyplot(fig)

```



```

st.write("Bienvenue chez Iran_Still_Call")

st.write(data_1)

#st.dataframe(data.head())

fig, ax = plt.subplots()
n_bins = st.number_input(
    label="Définir la valeur du client",
    min_value = 5,
    value=250,
    max_value=500
)
ax.hist(data_1,bins=n_bins)
st.pyplot(fig)

# ## Représentons notre modèle de classification de forêt
# aléatoire :
# Charger les données
data = pd.read_csv(r'C:/Users/Electro
Depot/Desktop/APPRENTISSAGE/Bases de données/Customer
Churn.csv')

# Features
x = data[['Customer Value', 'Frequency of SMS']]# Target
y = data['Churn']

# Train/test split
seed = 111
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=seed)

# Fonction d'évaluation
def evaluate_model(model):
    train_pred = model.predict(x_train)
    test_pred = model.predict(x_test)
    # Erreur quadratique moyenne
    E_train = mean_squared_error(y_train, train_pred,
squared=False)
    E_test = mean_squared_error(y_test, test_pred,
squared=False)

```

```

        return E_train, E_test

# Construction du modèle
rf = RandomForestClassifier(random_state=seed)
rf.fit(x_train, y_train)

# Interface Streamlit
st.title("Application de prédiction de churn")

# Ajouter des widgets pour les hyperparamètres si
nécessaire
# ...

# Faire des prédictions sur de nouvelles données (peut
être personnalisé en fonction de vos besoins)
new_data = pd.DataFrame({'Customer Value': [250,500,1000],
'Frequency of SMS': [0.5,2,10]})
# Nouvelles données à prédire
prediction = rf.predict(new_data)

# Afficher les résultats
st.write("Prédiction de churn :", prediction)

```

Interface de mon application

Deploy

Rerun R

Settings

Print

Record a screencast

About

Developer options

Clear cache C

Application Iran_still_Call

Auteur: KHALID

Cette application donne la possibilité de voir la distribution des abonnés à travers un histogramme en définissant leurs valeurs entre 5 et 500

Pairplot des variables

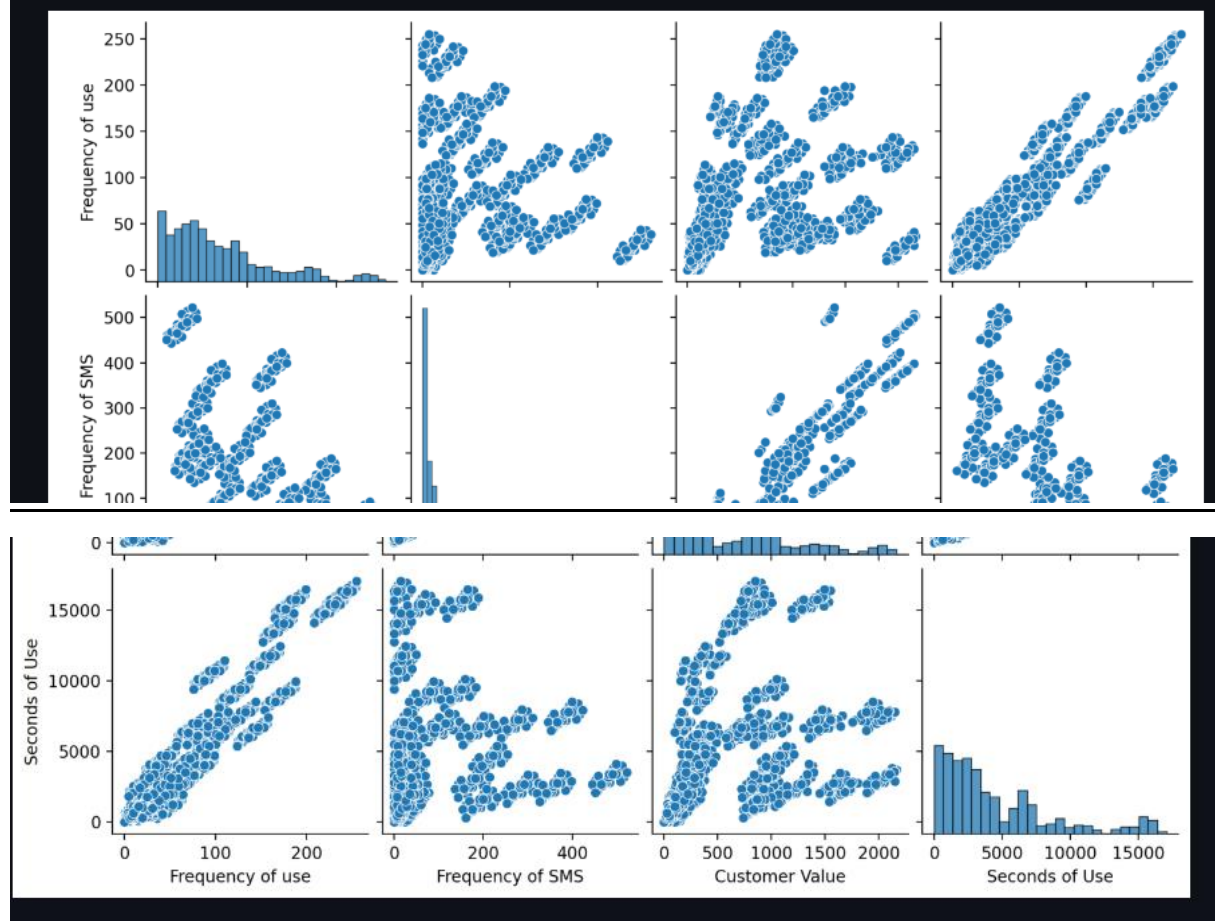
Représentation graphique des variables

Application Iran_still_Call

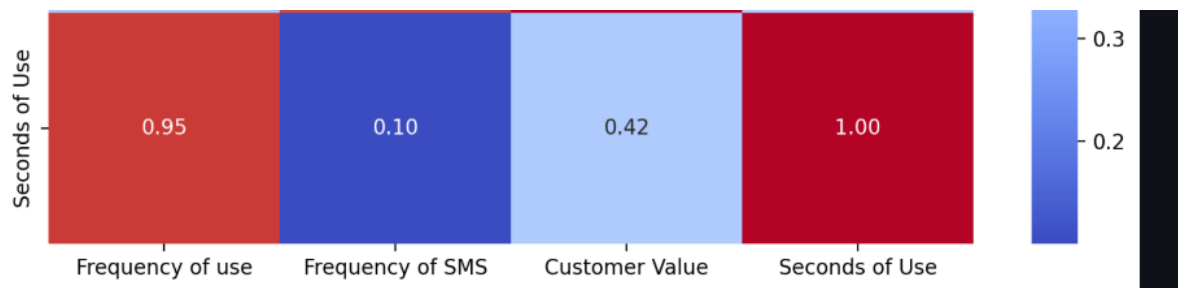
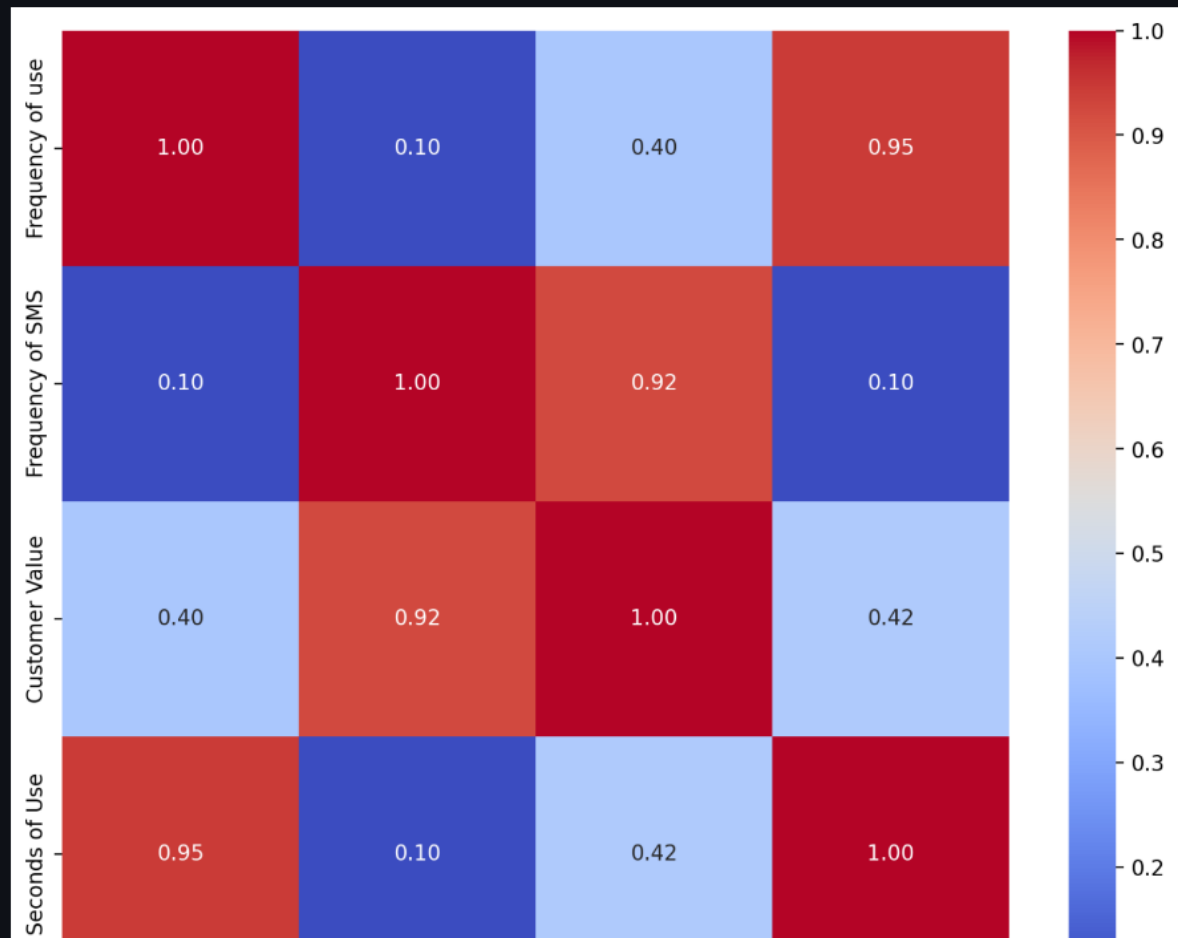
Auteur: KHALID

Cette application donne la possibilité de voir la distribution des abonnés à travers un histogramme en définissant leurs valeurs entre 5 et 500

Pairplot des variables



Matrice de corrélation des variables



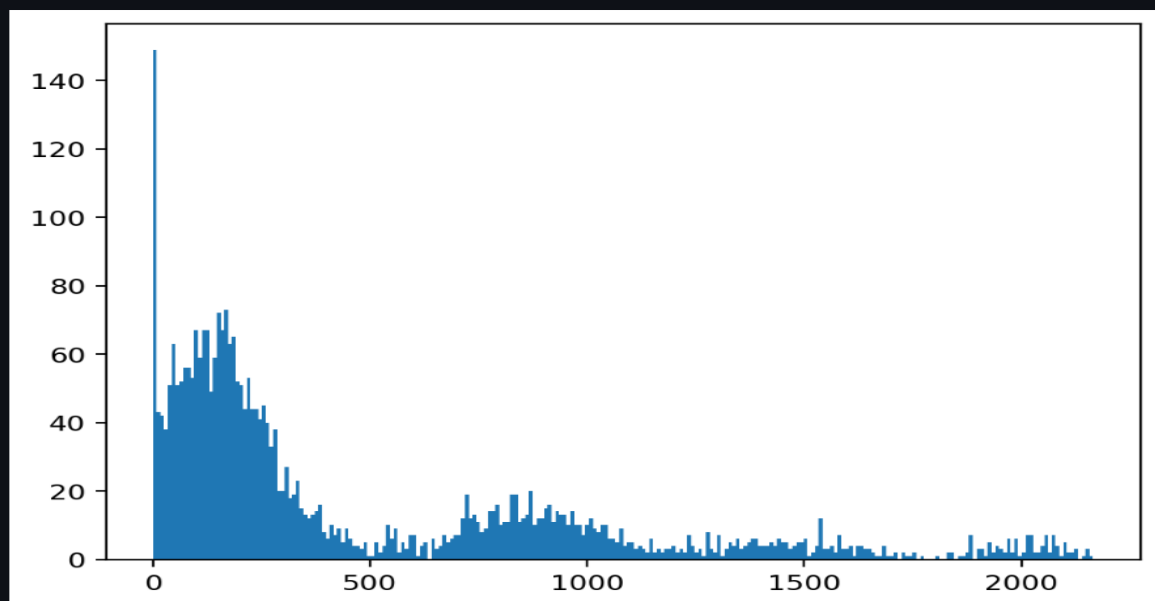
Bienvenue chez Iran_Still_Call

Bienvenue chez Iran_Still_Call

	Customer Value
0	197.64
1	46.035
2	1,536.52
3	240.02
4	145.805
5	282.28
6	1,235.96
7	945.44
8	557.68
9	191.92

Définir la valeur du client

250



Application du modèle : Le code ci-dessous nous a permis de prédire le désabonnement futur ou non de trois clients en définissant leur customer Value et leurs Frequency of SMS.

```
# Faire des prédictions sur de nouvelles données (peut être personnalisé en fonction de vos besoins)
```

```
new_data = pd.DataFrame({'Customer Value': [250,500,1000], 'Frequency of SMS': [0.5,2,10]})
```

```
# Nouvelles données à prédire
```

```
prediction = rf.predict(new_data)
```

```
# Afficher les résultats
```

```
st.write("Prédiction de churn :", prediction)
```

Application de prédiction de churn

Prédiction de churn :

value
0
0
0

Résultat : le modèle prédit un désabonnement des trois clients qui lui ont été soumis.

CONCLUSION

Ce projet a été entrepris dans le but de comprendre les facteurs qui influencent le désabonnement des clients dans une entreprise de télécommunications en Iran, ainsi que de construire des modèles pour prédire ces comportements de désabonnement. Il a permis fournir une compréhension approfondie du désabonnement des clients dans le secteur des télécommunications en Iran. L'analyse de la corrélation entre les variables offre des informations supplémentaires pour prendre des décisions éclairées. En fin de compte, ce projet contribue à l'objectif de l'entreprise de maintenir une base de clients satisfaits et rentables. Des axes d'amélioration futures seront l'introduction de nouveaux algorithmes et le déploiement de l'application en ligne.