# Importer les bibliothèques nécessaires

In [119…
```
# !pip install deap
```

In [ ]:
```
import random
from deap import base, creator, tools, algorithms
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd
import numpy as np
import missingno as msno
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier, ExtraTreesClassifier, RandomForest
from sklearn.model_selection import train_test_split

from sklearn import metrics
from sklearn.metrics import recall_score, confusion_matrix, precision_score, f1_score, accuracy_score, classifi
```

Chargement des données à partir du fichier 'data.csv' .

In [204…
```
df = pd.read_csv('data.csv')
df
```

Out[204]:

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | OnlineSecurity | ... | Devi |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7590-VHVEG | Female | 0 | Yes | No | 1 | No | No phone service | DSL | No | ... | |
| **1** | 5575-GNVDE | Male | 0 | No | No | 34 | Yes | No | DSL | Yes | ... | |
| **2** | 3668-QPYBK | Male | 0 | No | No | 2 | Yes | No | DSL | Yes | ... | |
| **3** | 7795-CFOCW | Male | 0 | No | No | 45 | No | No phone service | DSL | Yes | ... | |
| **4** | 9237-HQITU | Female | 0 | No | No | 2 | Yes | No | Fiber optic | No | ... | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **7038** | 6840-RESVB | Male | 0 | Yes | Yes | 24 | Yes | Yes | DSL | Yes | ... | |
| **7039** | 2234-XADUH | Female | 0 | Yes | Yes | 72 | Yes | Yes | Fiber optic | No | ... | |
| **7040** | 4801-JZAZL | Female | 0 | Yes | Yes | 11 | No | No phone service | DSL | Yes | ... | |
| **7041** | 8361-LTMKD | Male | 1 | Yes | No | 4 | Yes | Yes | Fiber optic | No | ... | |
| **7042** | 3186-AJIEK | Male | 0 | No | No | 66 | Yes | No | Fiber optic | Yes | ... | |

7043 rows × 21 columns

In [122…
```
#from google.colab import drive
#drive.mount('/content/drive')
```

Afficher la taille des données .

In [123…
```
df.shape
```

Out[123]:
```
(7043, 21)
```

Afficher les colonnes .

In [124…
```
df.columns.values
```

Out[124]:
```
array(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
       'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
       'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges',
       'TotalCharges', 'Churn'], dtype=object)
```

In [125…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

# Préparation des données .

In [205…
```
# la colonne 'customerID' n'est pas utile pour notre traitement .
df = df.drop(['customerID'], axis = 1)
df.head()
df['TotalCharges'] = pd.to_numeric(df.TotalCharges, errors = 'coerce') # erros = 'coerce' is used if there are .
```

In [206…
```
#Afficher le nombre des valeurs pour chaque colonne .
df.isnull().sum()
```

Out[206]:
```
gender               0
SeniorCitizen        0
Partner              0
Dependents           0
tenure               0
PhoneService         0
MultipleLines        0
InternetService      0
OnlineSecurity       0
OnlineBackup         0
DeviceProtection     0
TechSupport          0
StreamingTV          0
StreamingMovies      0
Contract             0
PaperlessBilling     0
PaymentMethod        0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64
```

In [128…
```
# Calculer le % des valeurs absantes pour chaque colonne .
def missing_values(n):
    df_m=pd.DataFrame()
    df_m["missing_values, %"]=df.isnull().sum()*100/len(df.isnull())
    df_m["missing_values, sum"]=df.isnull().sum()
    return df_m.sort_values(by="missing_values, %", ascending=False)
missing_values(df)
```

Out[128]:

| | missing_values, % | missing_values, sum |
|---|---|---|
| **TotalCharges** | 0.156183 | 11 |
| **gender** | 0.000000 | 0 |
| **SeniorCitizen** | 0.000000 | 0 |
| **MonthlyCharges** | 0.000000 | 0 |
| **PaymentMethod** | 0.000000 | 0 |
| **PaperlessBilling** | 0.000000 | 0 |
| **Contract** | 0.000000 | 0 |
| **StreamingMovies** | 0.000000 | 0 |
| **StreamingTV** | 0.000000 | 0 |
| **TechSupport** | 0.000000 | 0 |
| **DeviceProtection** | 0.000000 | 0 |
| **OnlineBackup** | 0.000000 | 0 |
| **OnlineSecurity** | 0.000000 | 0 |
| **InternetService** | 0.000000 | 0 |
| **MultipleLines** | 0.000000 | 0 |
| **PhoneService** | 0.000000 | 0 |
| **tenure** | 0.000000 | 0 |
| **Dependents** | 0.000000 | 0 |
| **Partner** | 0.000000 | 0 |
| **Churn** | 0.000000 | 0 |

In [129…

```
# Trouver les lignes avec des valeurs absantes pour chaque colonne .

df[np.isnan(df['TotalCharges'])]
```

Out[129]:

| | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | OnlineSecurity | OnlineBackup | Devic |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **488** | Female | 0 | Yes | Yes | 0 | No | No phone service | DSL | Yes | No | |
| **753** | Male | 0 | No | Yes | 0 | Yes | No | No | No internet service | No internet service | |
| **936** | Female | 0 | Yes | Yes | 0 | Yes | No | DSL | Yes | Yes | |
| **1082** | Male | 0 | Yes | Yes | 0 | Yes | Yes | No | No internet service | No internet service | |
| **1340** | Female | 0 | Yes | Yes | 0 | No | No phone service | DSL | Yes | Yes | |
| **3331** | Male | 0 | Yes | Yes | 0 | Yes | No | No | No internet service | No internet service | |
| **3826** | Male | 0 | Yes | Yes | 0 | Yes | Yes | No | No internet service | No internet service | |
| **4380** | Female | 0 | Yes | Yes | 0 | Yes | No | No | No internet service | No internet service | |
| **5218** | Male | 0 | Yes | Yes | 0 | Yes | No | No | No internet service | No internet service | |
| **6670** | Female | 0 | Yes | Yes | 0 | Yes | Yes | DSL | No | Yes | |
| **6754** | Male | 0 | No | Yes | 0 | Yes | Yes | DSL | Yes | Yes | |

In [130…

```
df[df['tenure'] == 0].index
```

Out[130]:

```
Int64Index([488, 753, 936, 1082, 1340, 3331, 3826, 4380, 5218, 6670, 6754], dtype='int64')
```

In [131…

```
# remplacer les valeurs manquantes dans la colonne 'TotalCharges' par la moyenne de cette colonne,
df['TotalCharges'].fillna(df['TotalCharges'].mean(), inplace=True)
# puis identifier les entrées où la durée de service est égale à zéro dans le DataFrame.
df[df['tenure'] == 0]
```

Out[131]:

| | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | OnlineSecurity | OnlineBackup | Devic |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **488** | Female | 0 | Yes | Yes | 0 | No | No phone service | DSL | Yes | No | |
| **753** | Male | 0 | No | Yes | 0 | Yes | No | No | No internet service | No internet service | |
| **936** | Female | 0 | Yes | Yes | 0 | Yes | No | DSL | Yes | Yes | |
| **1082** | Male | 0 | Yes | Yes | 0 | Yes | Yes | No | No internet service | No internet service | |
| **1340** | Female | 0 | Yes | Yes | 0 | No | No phone service | DSL | Yes | Yes | |
| **3331** | Male | 0 | Yes | Yes | 0 | Yes | No | No | No internet service | No internet service | |
| **3826** | Male | 0 | Yes | Yes | 0 | Yes | Yes | No | No internet service | No internet service | |
| **4380** | Female | 0 | Yes | Yes | 0 | Yes | No | No | No internet service | No internet service | |
| **5218** | Male | 0 | Yes | Yes | 0 | Yes | No | No | No internet service | No internet service | |
| **6670** | Female | 0 | Yes | Yes | 0 | Yes | Yes | DSL | No | Yes | |
| **6754** | Male | 0 | No | Yes | 0 | Yes | Yes | DSL | Yes | Yes | |

In [132…

```
df.isnull().sum()
```

Out[132]:

```
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

maintenant après le traitement les valeurs nulles n'éxistent plus.

# Détection des valeurs aberrantes .

In [133…

```
numerical_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
df[numerical_cols].describe().T
```

Out[133]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **tenure** | 7043.0 | 32.371149 | 24.559481 | 0.00 | 9.000 | 29.00 | 55.00 | 72.00 |
| **MonthlyCharges** | 7043.0 | 64.761692 | 30.090047 | 18.25 | 35.500 | 70.35 | 89.85 | 118.75 |
| **TotalCharges** | 7043.0 | 2283.300441 | 2265.000258 | 18.80 | 402.225 | 1400.55 | 3786.60 | 8684.80 |

In [134…

```
def detect_outliers_iqr(data):
    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)

    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = [x for x in data if x < lower_bound or x > upper_bound]

    return outliers

outliers_by_column = {}
```

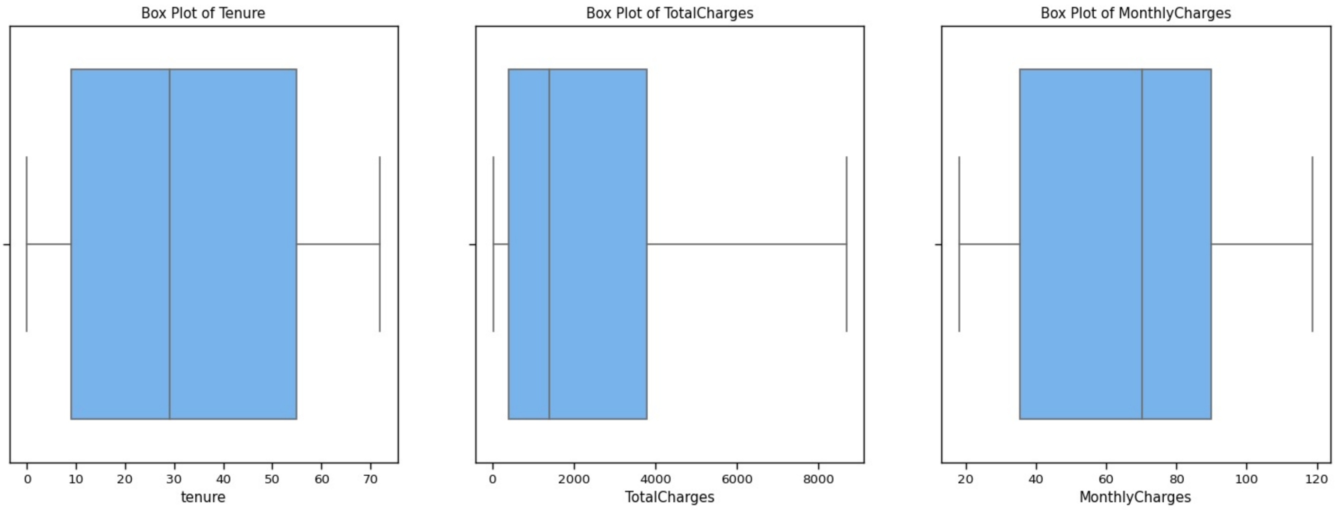
```
for column in numerical_cols:
    data_column = df[column]
    outliers = detect_outliers_iqr(data_column)
    outliers_by_column[column] = outliers

for column, outliers in outliers_by_column.items():
    print(f"Outliers in {column}: {outliers}")
```

```
Outliers in tenure: []
Outliers in MonthlyCharges: []
Outliers in TotalCharges: []
```

# Visualisation des données .

In [135…

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(18, 6))

plt.subplot(131)
sns.boxplot(x=df['tenure'], color='#66b3ff')
plt.title("Box Plot of Tenure")

plt.subplot(132)
sns.boxplot(x=df['TotalCharges'], color='#66b3ff')
plt.title("Box Plot of TotalCharges")

plt.subplot(133)
sns.boxplot(x=df['MonthlyCharges'], color='#66b3ff')
plt.title("Box Plot of MonthlyCharges")

plt.show()
```




In [136…

```
#  Vérifier les valeurs uniques pour prendre une décision d'encodage éclairéeunique_counts = df.nunique()
print("Unique Value Counts:")
print(unique_counts)
```

```
Unique Value Counts:
gender               2
SeniorCitizen        2
Partner              2
Dependents           2
tenure              73
PhoneService         2
MultipleLines        3
InternetService      3
OnlineSecurity       3
OnlineBackup         3
DeviceProtection     3
TechSupport          3
StreamingTV          3
StreamingMovies      3
Contract             3
PaperlessBilling     2
PaymentMethod        4
MonthlyCharges    1585
TotalCharges      6531
Churn                2
dtype: int64
```

# Traiter les varables catégorielles

In [137…

```
cols = ['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService', 'PaperlessBilling', 'Churn']
```

```
df[cols] = df[cols].astype('category')

for column in cols:
    df[column] = df[column].cat.codes

print(df.dtypes)
```

```
gender                 int8
SeniorCitizen          int8
Partner                int8
Dependents             int8
tenure                int64
PhoneService           int8
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling       int8
PaymentMethod        object
MonthlyCharges      float64
TotalCharges        float64
Churn                  int8
dtype: object
```

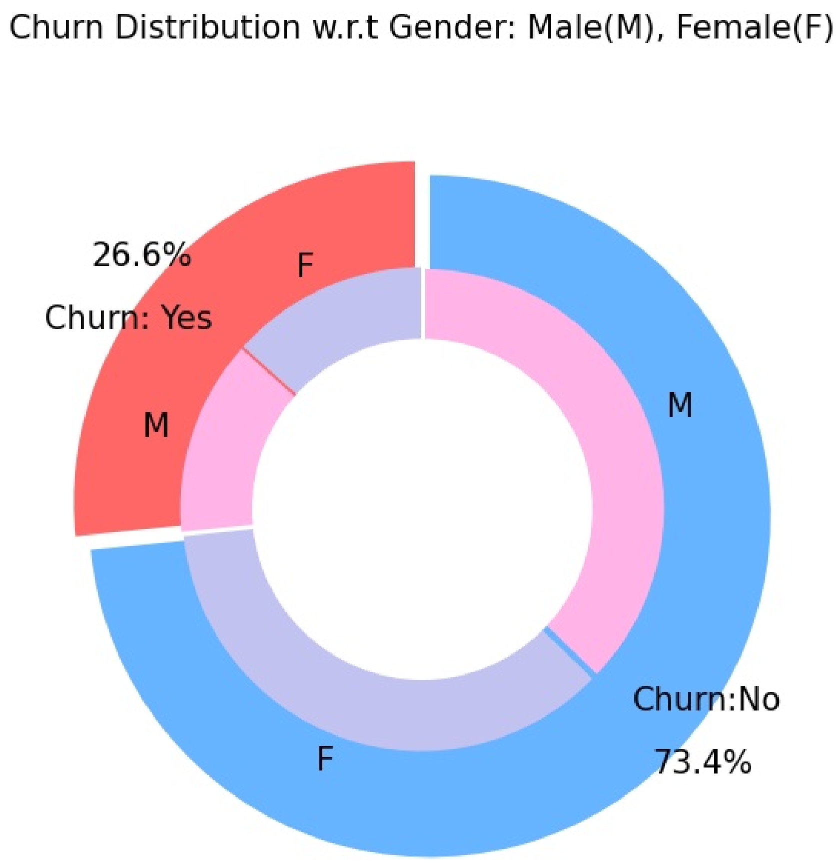
In [138…

```
g_labels = ['Male', 'Female']
c_labels = ['No', 'Yes']

fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'domain'}]])
fig.add_trace(go.Pie(labels=g_labels, values=df['gender'].value_counts(), name="Gender"),
              1, 1)
fig.add_trace(go.Pie(labels=c_labels, values=df['Churn'].value_counts(), name="Churn"),
              1, 2)

fig.update_traces(hole=.4, hoverinfo="label+percent+name", textfont_size=16)

fig.update_layout(
    title_text="Gender and Churn Distributions",
    annotations=[dict(text='Gender', x=0.16, y=0.5, font_size=20, showarrow=False),
                 dict(text='Churn', x=0.84, y=0.5, font_size=20, showarrow=False)])
fig.show()
```

In [139…

```
plt.figure(figsize=(6, 6))
labels =["Churn: Yes","Churn:No"]
values = [1869,5163]
labels_gender = ["F","M","F","M"]
sizes_gender = [939,930 , 2544,2619]
colors = ['#ff6666', '#66b3ff']
colors_gender = ['#c2c2f0','#ffb3e6', '#c2c2f0','#ffb3e6']
explode = (0.3,0.3)
explode_gender = (0.1,0.1,0.1,0.1)
```

```
textprops = {"fontsize":15}

plt.pie(values, labels=labels,autopct='%1.1f%%',pctdistance=1.08, labeldistance=0.8,colors=colors, startangle=9
plt.pie(sizes_gender,labels=labels_gender,colors=colors_gender,startangle=90, explode=explode_gender,radius=7,
centre_circle = plt.Circle((0,0),5,color='black', fc='white',linewidth=0)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.title('Churn Distribution w.r.t Gender: Male(M), Female(F)', fontsize=15, y=1.1)


plt.axis('equal')
plt.tight_layout()
plt.show()
```




In [140…
```
fig = px.histogram(df, x="Churn", color="Contract", barmode="group", title="<b>Customer contract distribution<b
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```

In [141...

```
labels = df['PaymentMethod'].unique()
values = df['PaymentMethod'].value_counts()

fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.3)])
fig.update_layout(title_text="<b>Payment Method Distribution</b>")
fig.show()
```

In [142...

```
fig = px.histogram(df, x="Churn", color="PaymentMethod", title="<b>Customer Payment Method distribution w.r.t.
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```

In [143...

```
fig = go.Figure()

fig.add_trace(go.Bar(
  x = [['Churn:No', 'Churn:No', 'Churn:Yes', 'Churn:Yes'],
       ["Female", "Male", "Female", "Male"]],
  y = [965, 992, 219, 240],
  name = 'DSL',
))

fig.add_trace(go.Bar(
  x = [['Churn:No', 'Churn:No', 'Churn:Yes', 'Churn:Yes'],
```

```
        ["Female", "Male", "Female", "Male"]],
  y = [889, 910, 664, 633],
  name = 'Fiber optic',
))

fig.add_trace(go.Bar(
  x = [['Churn:No', 'Churn:No', 'Churn:Yes', 'Churn:Yes'],
       ["Female", "Male", "Female", "Male"]],
  y = [690, 717, 56, 57],
  name = 'No Internet',
))

fig.update_layout(title_text="<b>Churn Distribution w.r.t. Internet Service and Gender</b>")

fig.show()
```

In [144…

```
color_map = {"Yes": "#FF97FF", "No": "#AB63FA"}
fig = px.histogram(df, x="Churn", color="Dependents", barmode="group", title="<b>Dependents distribution</b>",
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```

In [145…

```
color_map = {"Yes": '#FFA15A', "No": '#00CC96'}
```

```
fig = px.histogram(df, x="Churn", color="Partner", barmode="group", title="<b>Chrun distribution w.r.t. Partner
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```

In [146…

```
color_map = {"Yes": '#00CC96', "No": '#B6E880'}
fig = px.histogram(df, x="Churn", color="SeniorCitizen", title="<b>Chrun distribution w.r.t. Senior Citizen</b>
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```

In [147…

```
color_map = {"Yes": "#FF97FF", "No": "#AB63FA"}
fig = px.histogram(df, x="Churn", color="OnlineSecurity", barmode="group", title="<b>Churn w.r.t Online Securit
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```

In [148…

```
color_map = {"Yes": '#FFA15A', "No": '#00CC96'}
fig = px.histogram(df, x="Churn", color="PaperlessBilling",  title="<b>Chrun distribution w.r.t. Paperless Bill
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```

In [149…

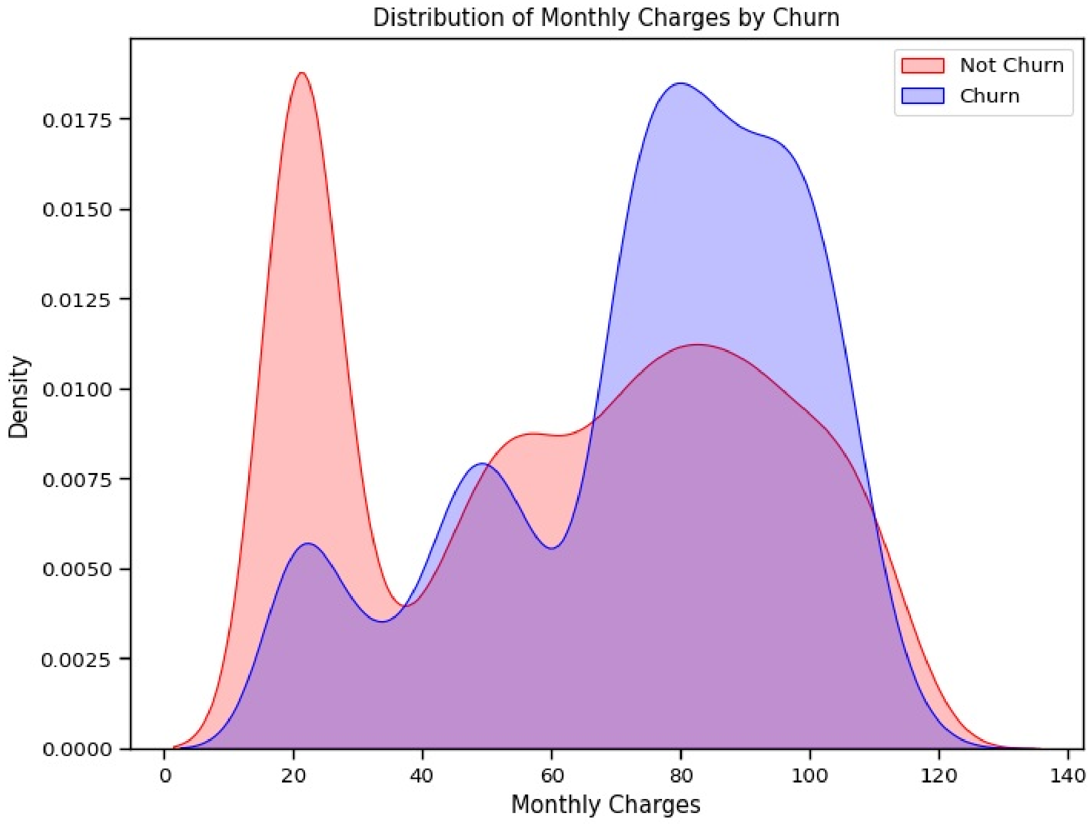
```
fig = px.histogram(df, x="Churn", color="TechSupport",barmode="group",  title="<b>Chrun distribution w.r.t. Tec
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```

In [150…

```
color_map = {"Yes": '#00CC96', "No": '#B6E880'}
fig = px.histogram(df, x="Churn", color="PhoneService", title="<b>Chrun distribution w.r.t. Phone Service</b>",
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```

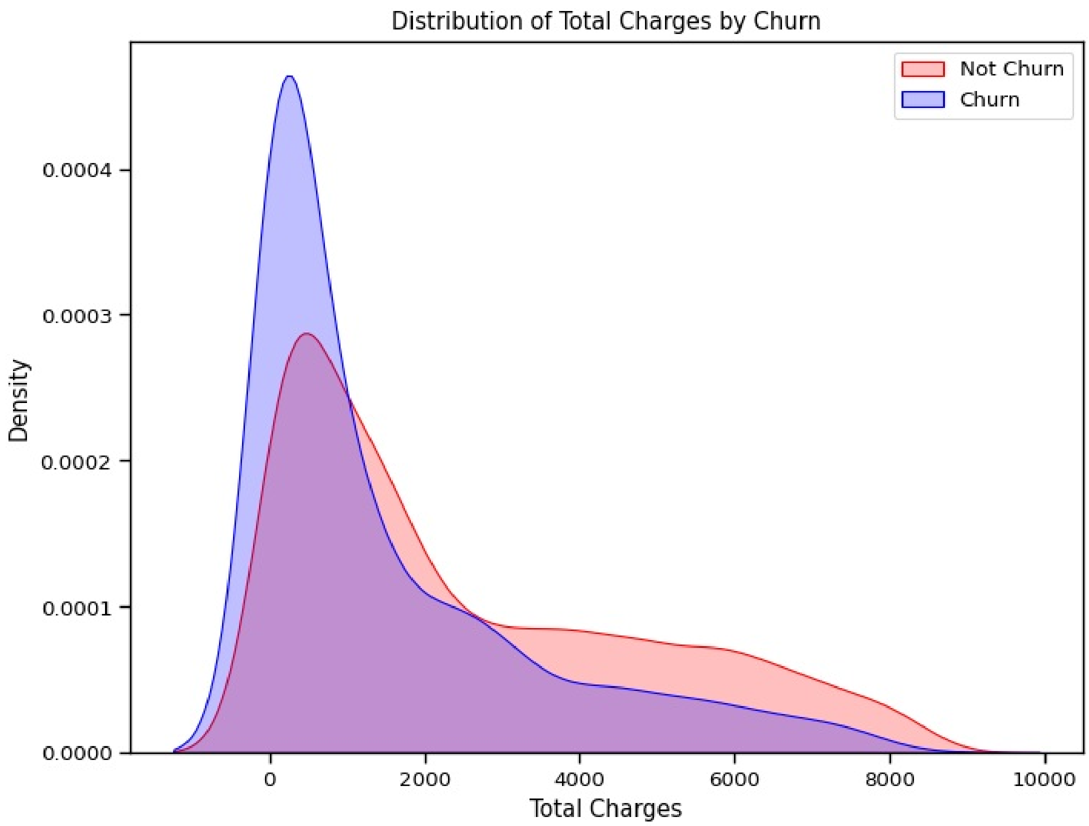
In [151…

```
sns.set_context("paper", font_scale=1.1)
plt.figure(figsize=(8, 6))

sns.kdeplot(df.MonthlyCharges[df['Churn'] == 0], color='red', label='Not Churn', shade=True)

sns.kdeplot(df.MonthlyCharges[df['Churn'] == 1], color='blue', label='Churn', shade=True)

plt.xlabel('Monthly Charges')
plt.ylabel('Density')
plt.title('Distribution of Monthly Charges by Churn')
plt.legend()

plt.show()
```




In [152…

```
sns.set_context("paper", font_scale=1.1)
plt.figure(figsize=(8, 6))

sns.kdeplot(df.TotalCharges[df['Churn'] == 0], color='red', label='Not Churn', shade=True)

sns.kdeplot(df.TotalCharges[df['Churn'] == 1], color='blue', label='Churn', shade=True)

plt.xlabel('Total Charges')
plt.ylabel('Density')
plt.title('Distribution of Total Charges by Churn')
plt.legend()

plt.show()
```




In [153…

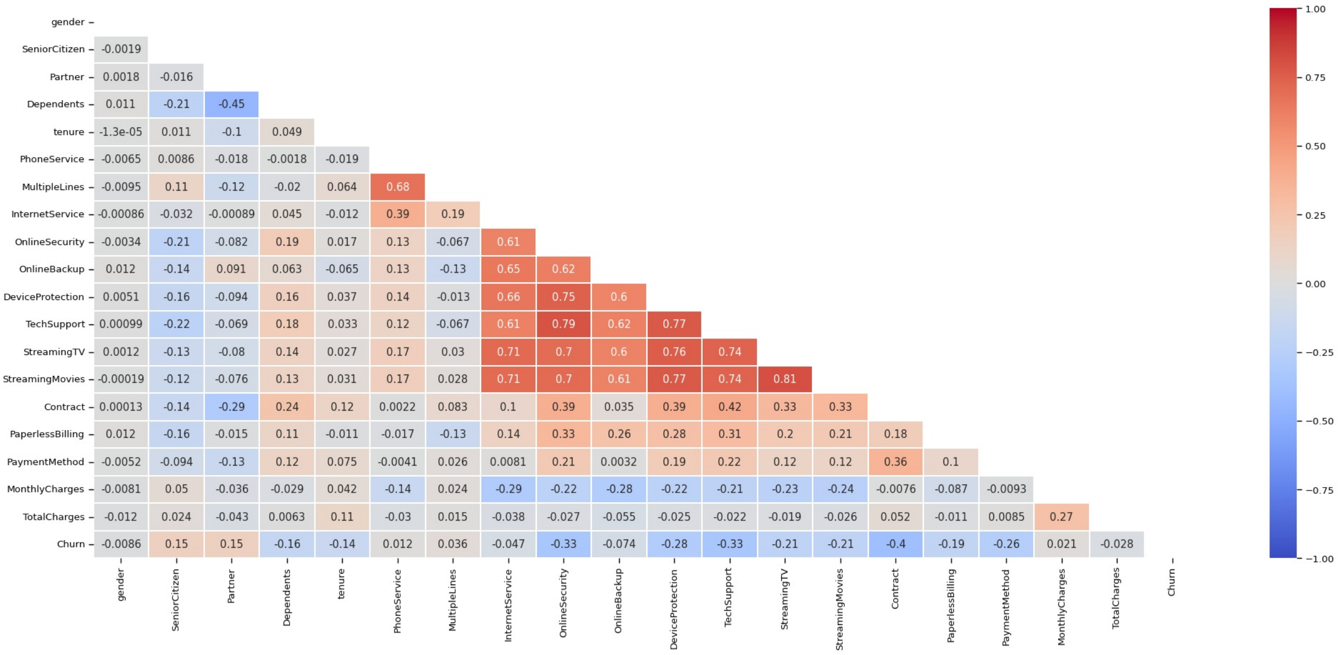
```
fig = px.box(df, x='Churn', y = 'tenure')

fig.update_yaxes(title_text='Tenure (Months)', row=1, col=1)
fig.update_xaxes(title_text='Churn', row=1, col=1)
```

```
fig.update_layout(autosize=True, width=750, height=600,
    title_font=dict(size=25, family='Courier'),
    title='<b>Tenure vs Churn</b>',
)

fig.show()
```

## Afficher la matrice de corrélation entre les variables .

In [154…

```
plt.figure(figsize=(25, 10))

corr = df.apply(lambda x: pd.factorize(x)[0]).corr()

mask = np.triu(np.ones_like(corr, dtype=bool))

ax = sns.heatmap(corr, mask=mask, xticklabels=corr.columns, yticklabels=corr.columns, annot=True, linewidths=.2
```

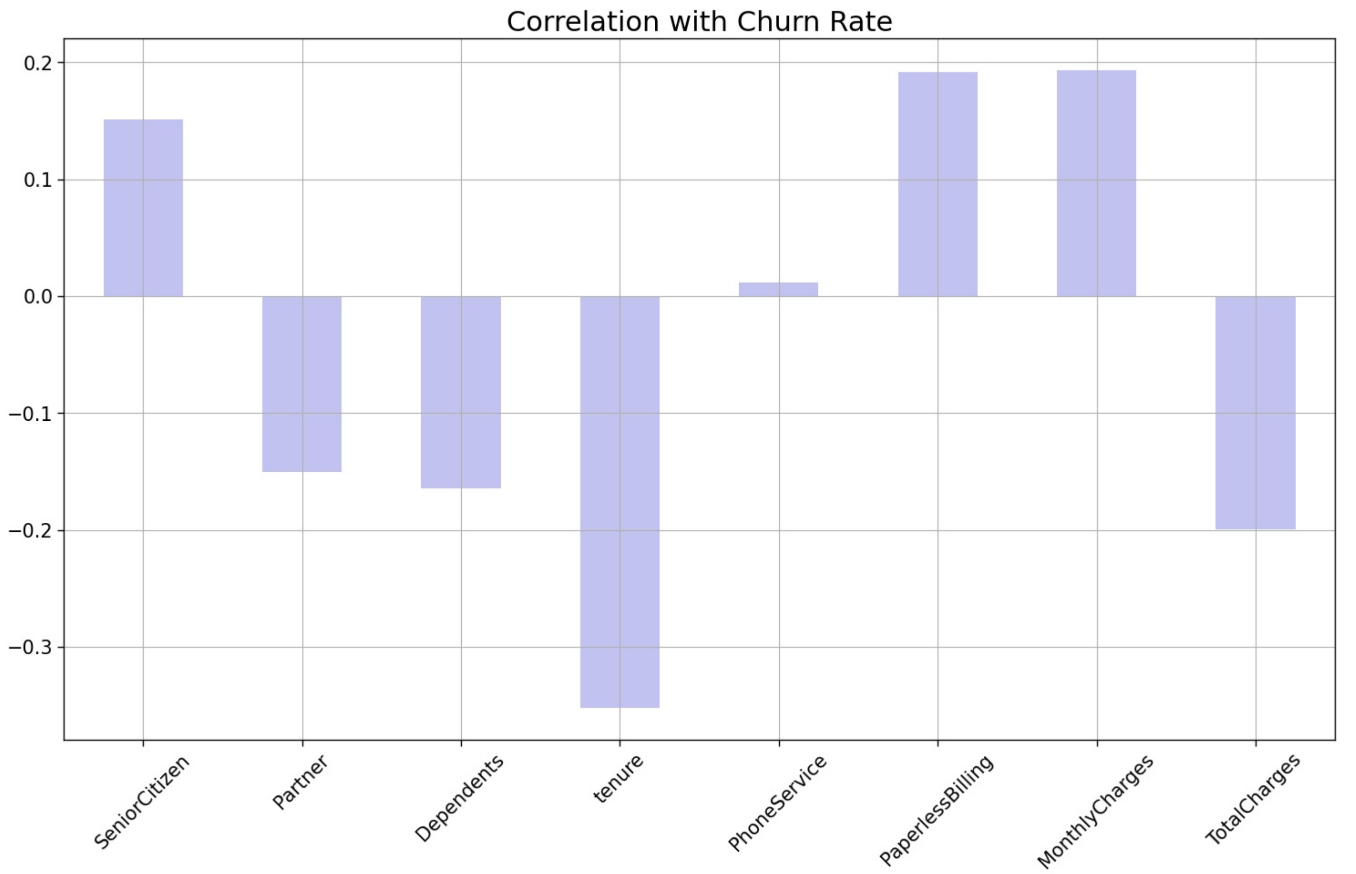



In [155…

```
plt.figure
ds_corr = df[['SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'PaperlessBilling',
```

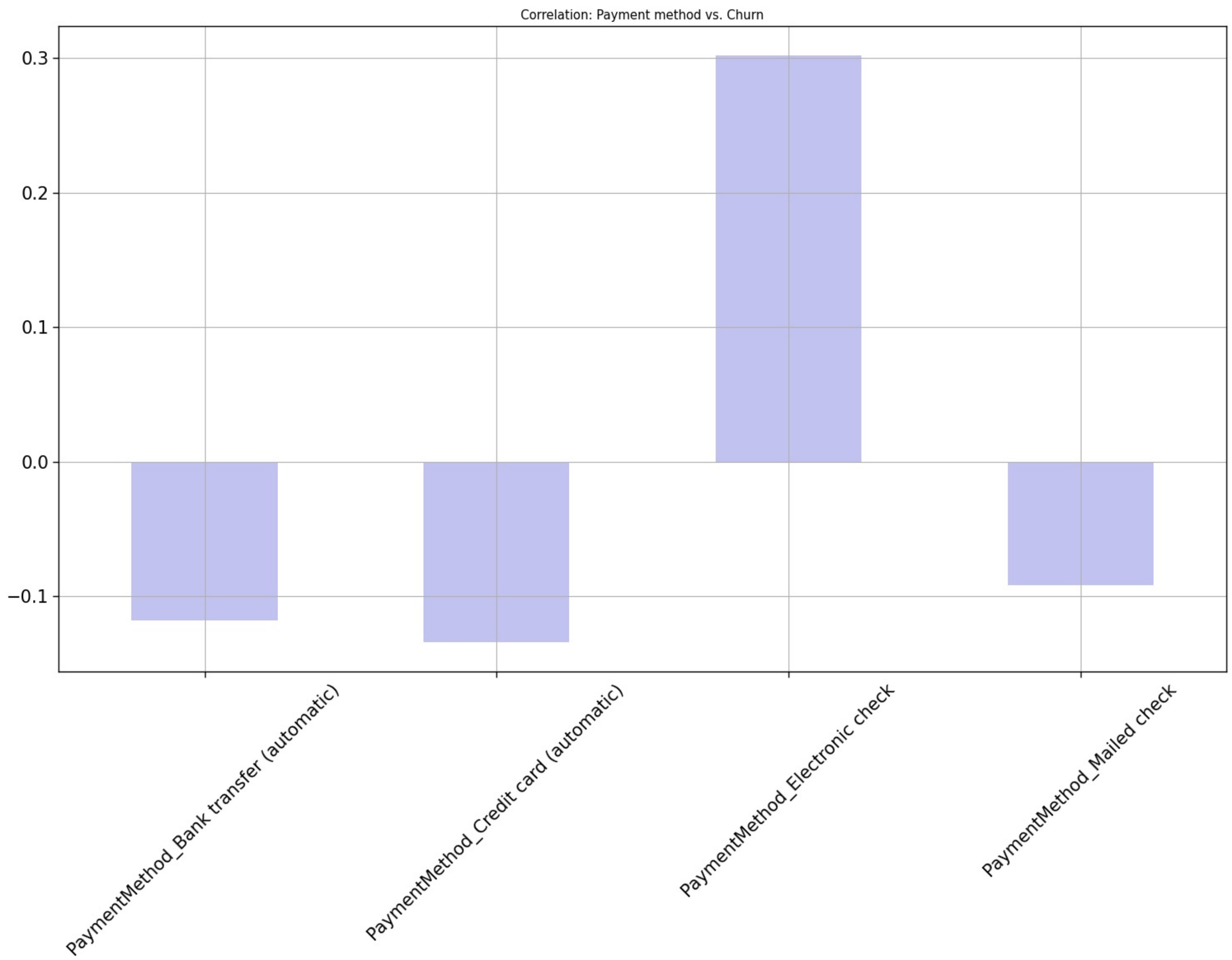
```
        'MonthlyCharges', 'TotalCharges']]

correlations = ds_corr.corrwith(df.Churn)
correlations = correlations[correlations!=1]
correlations.plot.bar(
        figsize = (18, 10),
        fontsize = 15,
        color = '#c2c2f0',
        rot = 45, grid = True)

plt.title('Correlation with Churn Rate', horizontalalignment="center", fontstyle = "normal", fontsize = "22", f
```

Out[155]:
```
Text(0.5, 1.0, 'Correlation with Churn Rate')
```




In [158…
```
plt.figure

ds_payment_method_corr = dataset[['PaymentMethod_Bank transfer (automatic)',
          'PaymentMethod_Credit card (automatic)',
          'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check']]

correlations = ds_payment_method_corr.corrwith(dataset.Churn)
correlations = correlations[correlations!=1]

correlations.plot.bar(
        figsize = (18, 10),
        fontsize = 15,
        color = '#c2c2f0',
        rot = 45, grid = True)

plt.title('Correlation: Payment method vs. Churn')
```

Out[158]:
```
Text(0.5, 1.0, 'Correlation: Payment method vs. Churn')
```




# Préparer les données pour l'entrainement .

In [156…

```
dataset = df.copy()

dataset = pd.get_dummies(dataset)
dataset.head()
```

Out[156]:

| | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | PaperlessBilling | MonthlyCharges | TotalCharges | Churn | ... | Streaming |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 29.85 | 29.85 | 0 | ... | |
| **1** | 1 | 0 | 0 | 0 | 34 | 1 | 0 | 56.95 | 1889.50 | 0 | ... | |
| **2** | 1 | 0 | 0 | 0 | 2 | 1 | 1 | 53.85 | 108.15 | 1 | ... | |
| **3** | 1 | 0 | 0 | 0 | 45 | 0 | 0 | 42.30 | 1840.75 | 0 | ... | |
| **4** | 0 | 0 | 0 | 0 | 2 | 1 | 1 | 70.70 | 151.65 | 1 | ... | |

5 rows × 41 columns

## Entrainer des modèles ML

Dans cette partie on va entrainer six modèles de ML ,pour chaque modèle on a appliqué un GridSearch pour trouver la combinaison d'hyperparamètres optimale afin d'augmenter la précision.

In [164…

```
#la première étape c'est le split des données .

X = dataset.drop('Churn', axis=1)
y = dataset['Churn']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [199…

```
from sklearn.model_selection import GridSearchCV
# Définir les hyperparamètres pour chaque modèle.
param_grid = {
    'SVM': {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf']
    },
    'K-Nearest Neighbors': {
        'n_neighbors': [3, 5, 7],
        'weights': ['uniform', 'distance']
    },
    'Random Forest': {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20]
    },
    'Decision Tree': {
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5, 10]
    },
    'Gradient Boosting': {
        'n_estimators': [100, 200, 300],
        'learning_rate': [0.01, 0.1, 0.2]
    },
    'Logistic Regression': {
        'C': [0.1, 1, 10],
        'penalty': ['l1', 'l2']
    }
}


best_models = {}

for model_name, model in models:
    if model_name in param_grid:
        grid_search = GridSearchCV(model, param_grid[model_name], cv=5, scoring='accuracy')
        grid_search.fit(X_train, y_train)
        best_model = grid_search.best_estimator_
        best_models[model_name] = best_model
        print(f"Best {model_name} Parameters: {grid_search.best_params_}")
```

```
Best SVM Parameters: {'C': 10, 'kernel': 'linear'}
Best K-Nearest Neighbors Parameters: {'n_neighbors': 7, 'weights': 'uniform'}
Best Random Forest Parameters: {'max_depth': 10, 'n_estimators': 200}
Best Decision Tree Parameters: {'max_depth': 10, 'min_samples_split': 10}
Best Naive Bayes Parameters: {}
Best Gradient Boosting Parameters: {'learning_rate': 0.1, 'n_estimators': 100}
Best Logistic Regression Parameters: {'C': 0.1, 'penalty': 'l2'}
```

In [200…

```
# maintenant on va afficher les précisions obtenus avec les meilleurs hyperparamètres .
for model_name, best_model in best_models.items():
    predictions = best_model.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    print(f"Best {model_name} Accuracy: {accuracy}")
```

```
Best SVM Accuracy: 0.807382867960246
Best K-Nearest Neighbors Accuracy: 0.7737813535257927
Best Random Forest Accuracy: 0.8026502602934217
Best Decision Tree Accuracy: 0.7704685281590156
Best Naive Bayes Accuracy: 0.6961665877898722
Best Gradient Boosting Accuracy: 0.808329389493611
Best Logistic Regression Accuracy: 0.8130619971604354
```

Les résultats affichent que le modèle "logistic regreesion " est le meilleur modèle avec une précision de 0.81 .

## Application de SVM-TLBO

Il s'agit d'un processus d'optimisation de ML qui utilise un algorithme d'optimisation basée sur l'apprentissage (TLBO) pour trouver les meilleurs hyperparamètres pour un classificateur (SVM).

La fonction "optimize" a pour objectif d'optimiser les hyperparamètres « C » et « gamma » du SVM pour la classification, et on va utiliser un algorithme TLBO de la bibliothèque DEAP pour réaliser cette optimisation.

In [194…
```python
C = 10 ** random.uniform(-3, 3)

def optimize(self):
    best_individual = None
    best_fitness = float('-inf')

    for i in range(self.n_iter):
        self.population = self.teach(self.population)

        self.population = self.learn(self.population)

        for individual in self.population:
            if individual[1] not in ['linear', 'rbf', 'sigmoid', 'poly', 'precomputed']:
                raise ValueError('Invalid kernel parameter')

            svm = SVC(C=C, kernel=individual[1])

            svm.fit(self.X, self.y)

            y_pred = svm.predict(self.X)

            accuracy = np.sum(y_pred == self.y) / len(self.y)

            if accuracy > best_fitness:
                best_individual = individual
                best_fitness = accuracy

    return best_individual
```

In [195…
```python
def evaluate(individual):
    C, gamma = individual
    gamma = max(gamma, 0.0)
    clf = SVC(C=C, gamma=gamma)
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    return accuracy,
```

In [196…
```python
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)
toolbox = base.Toolbox()
toolbox.register("attr_float", random.uniform, 0.1, 100)  # Parameters range for C and gamma
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_float, n=2)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("evaluate", evaluate)
toolbox.register("mate", tools.cxBlend, alpha=0.5)
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)
```

In [197…
```python
n_gen = 10

pop_size = 20

pop = toolbox.population(n=pop_size)

algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=n_gen, verbose=False)
```

Out[197]:
```
([[98.20001460588114, 95.27645556764747],
  [65.29584400362171, 97.59302237503792],
  [83.42372442934261, 96.26913138150483],
  [75.76143322311184, 94.65202631471266],
  [75.56462552615592, 95.96350052889788],
  [63.646850985588124, 94.5553310251619],
  [75.91220835203295, 93.2892632356101],
  [75.40088438807365, 92.2870936462713],
  [68.75368473799541, 98.16502619847509],
  [94.74217387150743, 94.7044517442103],
  [68.87440494401412, 93.23294750485563],
  [69.03889181352679, 93.40672987251298],
  [77.43186730259957, 97.87404122972235],
  [80.19370268672553, 91.74404944593472],
  [69.29300537576879, 96.03889935761596],
  [76.7492561948676, 94.55313028483519],
  [69.03889181352679, 93.40672987251298],
  [63.655654125669244, 96.98454590940652],
  [71.14384125591643, 96.04650962646379],
  [75.5558223860748, 93.53428564465327]],
 [{'gen': 0, 'nevals': 20},
  {'gen': 1, 'nevals': 9},
  {'gen': 2, 'nevals': 11},
  {'gen': 3, 'nevals': 8},
  {'gen': 4, 'nevals': 12},
  {'gen': 5, 'nevals': 11},
  {'gen': 6, 'nevals': 11},
  {'gen': 7, 'nevals': 14},
  {'gen': 8, 'nevals': 15},
  {'gen': 9, 'nevals': 14},
  {'gen': 10, 'nevals': 13}])
```

Initialiser et exécuter l'algorithme SVM-TLBO

In [198…
```
best_ind = tools.selBest(pop, 1)[0]
best_C, best_gamma = best_ind
best_clf = SVC(C=best_C, gamma=best_gamma)
best_clf.fit(X_train, y_train)
predictions = best_clf.predict(X_test)
accuracy = accuracy_score(y_test, predictions)

print("Best C:", best_C)
print("Best gamma:", best_gamma)
print("Accuracy:", accuracy)
```

```
Best C: 98.20001460588114
Best gamma: 95.27645556764747
Accuracy: 0.7316611452910554
```

## Application de SVM-ANT

In [177…
```
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import random

num_ants = 10
max_iterations = 100
alpha = 1.0
beta = 1.0
rho = 0.1
q0 = 0.7
num_features = 10
pheromone = np.ones(num_ants)
best_solution = None
best_score = float('-inf')

def evaluate_svm(C):
    svm = SVC(C=C)
    svm.fit(X_train, y_train)
    y_pred = svm.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    return score

for iteration in range(max_iterations):
    for ant in range(num_ants):
        C_values = [1.0, 10.0, 100.0]
        if random.random() < q0:
            heuristic_info = [pheromone[i] ** alpha * evaluate_svm(C_values[i]) ** beta for i in range(len(C_va
            selected_C = C_values[np.argmax(heuristic_info)]
        else:
            selected_C = random.choice(C_values)

        score = evaluate_svm(selected_C)

        pheromone[ant] = (1 - rho) * pheromone[ant] + score
```

```
        if score > best_score:
            best_solution = selected_C
            best_score = score

best_svm = SVC(C=best_solution)
best_svm.fit(X_train, y_train)

y_pred = best_svm.predict(X_test)
final_accuracy = accuracy_score(y_test, y_pred)

print(f"Best C: {best_solution}")
print(f"Final Accuracy: {final_accuracy}")
```

```
Best C: 100.0
Final Accuracy: 0.7908187411263606
```

# Comparaison des résultats :

On remarque que meme si On utilisé les méthodes d'optimisation le modèle traditionnel de SVM est le meilleur ,avec une précision "0.807382867960246" . tant que SVM-TLBO a donné une précision de "0.73166114529105" , et SVM-ANT a donné "0.7908187411263606" .

On peux éxpliquer les résultats par l'utilisation de GridSearch qui adopte la méthode de CrossValidation .