

# PROJET ATP TENNIS


## MEMBRES DE L'EQUIPE


Thibault DEMANUEL

Yawo-messah AMOUDJI

## INTRODUCTION

Bienvenue dans le projet ATP Tennis, une immersion complète dans le monde fascinant de l'exploration de données, de la visualisation et de la modélisation avec Python. Ce projet, conçu comme un guide exhaustif, vise à vous fournir une expérience pratique et approfondie des étapes essentielles pour analyser les données du circuit professionnel de tennis ATP.


 **Objectif du Projet:** Le projet ATP Tennis est une opportunité passionnante d'appliquer vos compétences en programmation Python à des données du monde réel. Notre objectif est de comprendre les différentes facettes de l'analyse de données, depuis l'exploration initiale jusqu'à la création de modèles prédictifs, en utilisant un ensemble de données riche en informations sur les performances des joueurs de tennis.

 **Ce que vous Apprendrez:**

**Exploration de Données:** Plongez dans les données ATP pour découvrir des tendances, des modèles et des insights clés.

**Visualisation Avancée:** Utilisez des bibliothèques Python telles que Matplotlib et Seaborn pour créer des visualisations percutantes.

**Modélisation Prédictive:** Appliquez des concepts de machine learning pour prédire des résultats et comprendre les facteurs qui influent sur les performances des joueurs.

 **Pour Qui:** Que vous soyez débutant ou déjà familier avec Python, ce projet est conçu pour vous offrir une expérience pratique et progressive. Améliorez vos compétences en programmation, plongez dans le monde du tennis professionnel, et découvrez comment utiliser Python pour donner vie aux données.

Préparez-vous à explorer, visualiser, modéliser et, surtout, à découvrir les mystères cachés derrière les chiffres du tennis professionnel avec le projet ATP Tennis!

## EXPLORATION DE DONNEES

### INFORMATIONS SUR DATASET ATP

```
from google.colab import autoviz
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
%matplotlib inline
import pandas as pd

t1 = pd.read_csv("/content/atp_data.csv", sep=",")
t2 = pd.read_csv("/content/confidence_data.csv", sep=",")

print(t1.head())
print("La taille du jeu de donnée T1 est de", t1.shape)
print("_____")
print(t2.head())
print("La taille du jeu de donnée T1 est de", t2.shape)
```

	ATP	Location		Tournament	Date	\
0	1	Adelaide	Australian Hardcourt	Championships	2000-01-03	
1	3	Doha		Qatar Open	2000-01-03	
2	3	Doha		Qatar Open	2000-01-03	
3	3	Doha		Qatar Open	2000-01-03	
4	3	Doha		Qatar Open	2000-01-03	

	Series	Court	Surface	Round	Best of	Winner	...	\
0	International	Outdoor	Hard	1st Round	3	Dosedel S.	...	
1	International	Outdoor	Hard	1st Round	3	Kiefer N.	...	
2	International	Outdoor	Hard	1st Round	3	Gaudio G.	...	

3	International	Outdoor	Hard	1st Round	3	El Aynaoui Y.	...
4	International	Outdoor	Hard	1st Round	3	Cherkasov A.	...

	Wsets	Lsets	Comment	PSW	PSL	B365W	B365L	elo_winner	elo_loser	\
0	2.0	0.0	Completed	NaN	NaN	NaN	NaN	1500.0	1500.0	
1	2.0	0.0	Completed	NaN	NaN	NaN	NaN	1500.0	1500.0	
2	2.0	1.0	Completed	NaN	NaN	NaN	NaN	1500.0	1500.0	
3	2.0	1.0	Completed	NaN	NaN	NaN	NaN	1500.0	1500.0	
4	2.0	0.0	Completed	NaN	NaN	NaN	NaN	1500.0	1500.0	

	proba_elo
0	0.5
1	0.5
2	0.5
3	0.5
4	0.5

[5 rows x 23 columns]  
 La taille du jeu de donnée T1 est de (44708, 23)

	match	PSW	win0	confidence0	date
0	20445	2.85	1	2.037852	2016-07-22
1	19518	1.41	0	1.973181	2016-04-16
2	15396	3.15	1	1.950877	2013-08-13
3	18868	1.55	0	1.914305	2016-01-25
4	22118	2.70	1	1.888781	2017-04-18

La taille du jeu de donnée T1 est de (11054, 5)

```
print(t1["Winner"])

0      Dosedel S.
1      Kiefer N.
2      Gaudio G.
3      El Aynaoui Y.
4      Cherkasov A.
...
44703  Bautista Agut R.
44704  Anderson K.
44705  Jarry N.
44706  Del Potro J.M.
44707  Fognini F.
Name: Winner, Length: 44708, dtype: object
```

```
print(len(t1["Winner"].unique()))

print("_____")

print(t1["Tournament"].value_counts(normalize=True)*100)
```

899	
Australian Open	4.829113
US Open	4.545048
French Open	4.545048
Wimbledon	4.545048
Sony Ericsson Open	2.124899
...	
LA Tennis Open	0.060392
Gazprom Hungarian Open	0.060392
Power Horse Cup	0.060392
Portugal Open	0.060392
New York Open	0.060392

Name: Tournament, Length: 207, dtype: float64

```
t1.info()
print("_____")
somme_t1=t1.isna().sum().sum()
moyenne_t1=round((t1.isna().mean().mean()*100), 2)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44708 entries, 0 to 44707
Data columns (total 23 columns):
#   Column      Non-Null Count  Dtype
---  -
0   ATP          44708 non-null  int64
1   Location     44708 non-null  object
2   Tournament   44708 non-null  object
3   Date         44708 non-null  object
4   Series       44708 non-null  object
5   Court        44708 non-null  object
6   Surface      44708 non-null  object
7   Round        44708 non-null  object
8   Best of      44708 non-null  int64
```

```

9 Winner 44708 non-null object
10 Loser 44708 non-null object
11 WRank 44708 non-null int64
12 LRank 44708 non-null int64
13 Wsets 44521 non-null float64
14 Lsets 44521 non-null float64
15 Comment 44708 non-null object
16 PSW 32743 non-null float64
17 PSL 32743 non-null float64
18 B365W 39037 non-null float64
19 B365L 39057 non-null float64
20 elo_winner 44708 non-null float64
21 elo_loser 44708 non-null float64
22 proba_elo 44708 non-null float64
dtypes: float64(9), int64(4), object(10)
memory usage: 7.8+ MB

```

---

Sur l'ensemble du dataset t1, il manque 35626 données. Ce qui représente 3.46 % du jeu de données

## INFORMATIONS SUR DATASET CONFIDENCE DATA

```

t2.info()
print("_____")

somme_t2=t2.isna().sum().sum()

moyenne_t2=round((t2.isna().mean().mean()*100), 2)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11054 entries, 0 to 11053
Data columns (total 5 columns):
#   Column      Non-Null Count  Dtype
---  -
0   match       11054 non-null  int64
1   PSW         11008 non-null  float64
2   win0        11054 non-null  int64
3   confidence0 11054 non-null  float64
4   date        11054 non-null  object
dtypes: float64(2), int64(2), object(1)
memory usage: 431.9+ KB

```

---

Sur l'ensemble du dataset t2, il manque 46 données. Ce qui représente 0.08 % du jeu de données

## ✓ DISPONIBILITE DES JEUX DE DONNEES ET VOLUMETRIE

Les jeux de données sont disponibles sur le site kaggle et sont open source.

Pour y accéder: <https://www.kaggle.com/edouardthomas/atp-matches-dataset>

Les explications de ces derniers sont accessibles sur :

- <https://towardsdatascience.com/making-big-bucks-with-a-data-driven-sports-betting-strategy-6c21a6869171>
- [https://www.researchgate.net/publication/331218530\\_Exploiting\\_sports-betting\\_market\\_using\\_machine\\_learning](https://www.researchgate.net/publication/331218530_Exploiting_sports-betting_market_using_machine_learning)

```

print("Les dimensions du dataset ATP sont :", t1.shape)
print("_____")
print("Les dimensions du dataset CONFIDENCE ATP sont :", t2.shape)

```

```

Les dimensions du dataset ATP sont : (44708, 23)

```

```

Les dimensions du dataset CONFIDENCE ATP sont : (11054, 5)

```

X

## ✓ TRANSFORMATION DES COLONNES DE DATE

```

t1["Date"]=pd.to_datetime(t1["Date"])
t2["date"]=pd.to_datetime(t2["date"])

print(t1.info())
print("_____")
print(t2.info())

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44708 entries, 0 to 44707
Data columns (total 23 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   ATP              44708 non-null  int64
1   Location         44708 non-null  object
2   Tournament       44708 non-null  object
3   Date             44708 non-null  datetime64[ns]
4   Series           44708 non-null  object
5   Court            44708 non-null  object
6   Surface          44708 non-null  object
7   Round            44708 non-null  object
8   Best of         44708 non-null  int64
9   Winner           44708 non-null  object
10  Loser            44708 non-null  object
11  WRank            44708 non-null  int64
12  LRank            44708 non-null  int64
13  Wsets            44521 non-null  float64
14  Lsets            44521 non-null  float64
15  Comment          44708 non-null  object
16  PSW              32743 non-null  float64
17  PSL              32743 non-null  float64
18  B365W            39037 non-null  float64
19  B365L            39057 non-null  float64
20  elo_winner       44708 non-null  float64
21  elo_loser        44708 non-null  float64
22  proba_elo        44708 non-null  float64
dtypes: datetime64[ns](1), float64(9), int64(4), object(9)
memory usage: 7.8+ MB
None

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11054 entries, 0 to 11053
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   match           11054 non-null  int64
1   PSW             11008 non-null  float64
2   win0            11054 non-null  int64
3   confidence0     11054 non-null  float64
4   date            11054 non-null  datetime64[ns]
dtypes: datetime64[ns](1), float64(2), int64(2)
memory usage: 431.9 KB
None

```

Après cette première partie de prise de connaissance sur nos jeux de données, nous constatons que pour la suite du travail nous allons travailler avec le jeu de données ATP qui, recense les variables nécessaires à la mise en place des algorithmes.

```

t1 = t1.dropna(axis = 0, how = 'any')
print(t1.isna().sum())
print(t1.shape)
#12517 lignes supprimés
#Nous décidons de supprimer les lignes de codes avec des NaNs car nous n'avons pas la possibilité de
#les remplacer par la moyenne, la médian ou le mode. Cela fausserai l'algorithme.

```

```

ATP              0
Location         0
Tournament       0
Date             0
Series           0
Court            0
Surface          0
Round            0
Best of         0
Winner           0
Loser            0
WRank            0
LRank            0
Wsets            0
Lsets            0
Comment          0
PSW              0
PSL              0
B365W            0
B365L            0
elo_winner       0
elo_loser        0
proba_elo        0
dtype: int64
(32191, 23)

```

```

t1.set_index("ATP")

```

	Location	Tournament	Date	Series	Court	Surface	Round	Best of	W
ATP									
3	Doha	Qatar Exxon Mobil Open	2004-01-05	International	Outdoor	Hard	1st Round	3	Ul
3	Doha	Qatar Exxon Mobil Open	2004-01-05	International	Outdoor	Hard	1st Round	3	Yc
3	Doha	Qatar Exxon Mobil Open	2004-01-05	International	Outdoor	Hard	1st Round	3	Sa
2	Chennai	TATA Open	2004-01-05	International	Outdoor	Hard	1st Round	3	L
2	Chennai	TATA Open	2004-01-05	International	Outdoor	Hard	1st Round	3	L
...	...	...	...	...	...	...	...	...	
17	Dubai	Dubai Tennis Championships	2018-03-03	ATP500	Outdoor	Hard	The Final	3	B: A
16	Acapulco	Abierto Mexicano	2018-03-03	ATP500	Outdoor	Hard	Semifinals	3	Anc
18	Sao Paulo	Brasil Open	2018-03-03	ATP250	Indoor	Clay	Semifinals	3	J:
16	Acapulco	Abierto Mexicano	2018-03-04	ATP500	Outdoor	Hard	The Final	3	De
18	Sao Paulo	Brasil Open	2018-03-04	ATP250	Indoor	Clay	The Final	3	F

32191 rows × 22 columns

t1.dtypes

```

ATP                int64
Location           object
Tournament         object
Date              datetime64[ns]
Series            object
Court             object
Surface           object
Round            object
Best of           int64
Winner           object
Loser            object
WRank            int64
LRank            int64
Wsets            float64
Lsets            float64
Comment          object
PSW              float64
PSL              float64
B365W            float64
B365L            float64
elo_winner       float64
elo_loser        float64
proba_elo        float64
dtype: object

```

```
print(t1["Location"].unique())
```

```

['Doha' 'Chennai' 'Adelaide' 'Auckland' 'Sydney' 'Melbourne'
 'Vina del Mar' 'San Jose' 'Milan' 'Memphis' 'Rotterdam' 'Buenos Aires'
 'Marseille' 'Costa Do Sauipe' 'Dubai' 'Scottsdale' 'Acapulco'
 'Indian Wells' 'Miami' 'Valencia' 'Houston' 'Estoril' 'Monte Carlo'
 'Barcelona' 'Munich' 'Rome' 'Hamburg' 'St. Polten' 'Casablanca' 'Paris'
 'Queens Club' 'Halle' 'Nottingham' "'s-Hertogenbosch" 'London' 'Bastad'
 'Gstaad' 'Newport' 'Los Angeles' 'Stuttgart' 'Amersfoort' 'Kitzbuhel'
 'Umag' 'Indianapolis' 'Toronto' 'Cincinnati' 'Sopot' 'Washington'
 'Long Island' 'New York' 'Bucharest' 'Delray Beach' 'Beijing' 'Palermo'
 'Shanghai' 'Bangkok' 'Tokyo' 'Lyon' 'Moscow' 'Vienna' 'Metz' 'Madrid'
 'St. Petersburg' 'Stockholm' 'Basel' 'Montreal' 'New Haven'
 'Ho Chi Min City' 'Zagreb' 'Las Vegas' 'Portschach' 'Mumbai' 'Warsaw'
 'Brisbane' 'Santiago' 'Johannesburg' 'Belgrade' 'Nice' 'Eastbourne'
 'Atlanta' 'Kuala Lumpur' 'Montpellier' 'Winston-Salem' 'Sao Paulo'
 'Oeiras' 'Dusseldorf' 'Bogota' 'Quito' 'Rio de Janeiro' 'Istanbul'
 'Estoril' 'Geneva' 'Shenzhen' 'Sofia' 'Marrakech' 'Los Cabos' 'Chengdu'
 'Antwerp' 'Budapest' 'Antalya' 'Pune']

```

La colonne Location du dataset représente les lieux où se sont déroulés les tournois

```
print(t1["Tournament"].unique())

['Qatar Exxon Mobil Open' 'TATA Open' 'AAPT Championships' 'Heineken Open'
 'adidas International' 'Australian Open' 'Bellsouth Open' 'Siebel Open'
 'Indesit ATP Milano Indoor ' 'Kroger St. Jude'
 'ABN AMRO World Tennis Tournament' 'ATP Buenos Aires 2004' 'Open 13'
 'Brasil Open' 'Dubai Championships' 'Franklin Templeton Tennis Classic'
 'Abierto Mexicano' 'Pacific Life Open' 'NASDAQ-100 Open'
 'CAM Open Comunidad Valenciana' 'U.S. Men's Clay Court Championships'
 'Estoril Open' 'Monte Carlo Masters' 'Open Seat Godo' 'BMW Open'
 'Telecom Italia Masters Roma' 'Hamburg TMS'
 'Internationaler Raiffeisen Grand Prix' 'Grand Prix Hassan II'
 'French Open' 'Stella Artois' 'Gerry Weber Open' 'Nottingham Open'
 'Ordina Open' 'Wimbledon' 'Swedish Open' 'Allianz Suisse Open'
 'Hall of Fame Championships' 'Mercedes-Benz Cup' 'Mercedes Cup'
 'Priority Telecom Dutch Open' 'Generali Open' 'Croatia Open'
 'RCA Championships' 'Toronto TMS'
 'Western & Southern Financial Group Masters' 'Idea Prokom Open'
 'Legg Mason Classic' 'TD Waterhouse Cup' 'US Open' 'Open Romania'
 'International Championships' 'China Open'
 'Campionati Internazionali Di Sicilia' 'Thailand Open' 'Japan Open'
 'Grand Prix de Lyon' 'Kremlin Cup' 'CA Tennis Trophy' 'Open de Moselle'
 'Madrid Masters' 'St. Petersburg Open' 'Stockholm Open' 'Swiss Indoors'
 'BNP Paribas' 'Masters Cup' 'Next Generation Hardcourts'
 'Medibank International' 'Internazionali di Lombardia' 'SAP Open'
 'ATP Buenos Aires 2005' 'Regions Morgan Keegan Championships'
 'Channel Open' 'Dubai Duty Free Men's Open'
 'Open de Tennis Comunidad Valenciana' 'Rogers Cup' 'Pilot Pen Tennis'
 'Vietnam Open' 'BA-CA Tennis Trophy' 'Davidoff Swiss Indoors'
 'Chennai Open' 'Next Generation Adelaide International'
 'Sydney International' 'Movistar Open' 'PBZ Zagreb Indoors' 'Copa Telmex'
 'Dubai Tennis Championships' 'Campionati Internazionali d'Italia'
 'Hypo Group Tennis International' 'Red Letter Days Open'
 'Symsam Swedish Open' 'Dutch Open' 'Countrywide Classic' 'Rogers Masters'
 'Kingfisher Airlines Tennis Open' 'Sony Ericsson Open'
 'Internazionali BNL d'Italia' 'The Nottingham Open'
 'Catella Swedish Open' 'Austrian Open'
 'Indianapolis Tennis Championships' 'AIG Japan Open Tennis Championships'
 'Open Sabadell Atlántico 2008' 'Orange Prokom Open' 'Slazenger Open'
 'Studena Croatia Open' 'Brisbane International' 'SA Tennis Open'
 'BNP Paribas Open' 'Open Banco Sabadell' 'Serbia Open'
 'Mutua Madrileña Madrid Open' 'Open de Nice Côte d'Azur'
 'AEGON Championships' 'Unicef Open' 'AEGON International'
 'SkiStar Swedish Open' 'German Open Tennis Championships'
 'Atlanta Tennis Championships' 'Farmers Classic' 'Proton Malaysian Open'
 'Rakuten Japan Open Tennis Championships' 'Shanghai Masters'
 'Open Sud de France' 'Valencia Open 500' 'BNP Paribas Masters'
 'Copa Claro' 'Bet-At-Home Cup'
 'Winston-Salem Open at Wake Forest University' 'Malaysian Open'
 'Erste Bank Open' 'Apia International' 'VTR Open'
 'BRD Nastase Tiriac Trophy' 'Mutua Madrid Open' 'ATP Vegeta Croatia Open'
 'Crédit Agricole Suisse Open Gstaad' 'BB&T Atlanta Open' 'Citi Open'
 'U.S. National Indoor Tennis Championships' 'Portugal Open'
 'Power Horse Cup' 'Topshelf Open' 'Claro Open Colombia' 'Ecuador Open'
 'Memphis Open' 'Rio Open' 'Delray Beach Open' 'Argentina Open'
 'Istanbul Open' 'Millenium Estoril Open' 'Geneva Open' 'AEGON Open'
 'Konzum Croatia Open' 'Suisse Open Gstaad' 'bet-at-home Open'
 'Shenzhen Open' 'ASB Classic' 'Garanti Koza Sofia Open'
 'German Tennis Championships' 'Abierto Mexicano Mifel' 'Chengdu Open'
 'European Open' 'Gazprom Hungarian Open' 'Open Banco Sabadell ']
```

La colonne Tournament du dataset représente le nom des tournois

```
print(t1["Series"].value_counts())

ATP250          8176
Grand Slam      6148
International    6067
Masters 1000    3900
ATP500          3025
Masters         2827
International Gold 1872
Masters Cup      176
Name: Series, dtype: int64
```

La colonne Series du dataset représente la catégories de tournoi.

```
print(t1["Court"].value_counts())

Outdoor    26306
Indoor      5885
Name: Court, dtype: int64
```

La colonne Court du dataset permet de savoir si le match a eu lieu à l'intérieur ou à l'extérieur.

```
print(t1["Surface"].value_counts())
```

Hard	17354
Clay	10265
Grass	3610
Carpet	962

Name: Surface, dtype: int64

La colonne Surface du dataset permet de savoir sur quel type de surface a eu lieu le match.

```
print(t1["Round"].value_counts())
```

1st Round	14547
2nd Round	9045
Quarterfinals	3105
3rd Round	2288
Semifinals	1581
The Final	797
4th Round	578
Round Robin	250

Name: Round, dtype: int64

La colonne Round du dataset permet de savoir combien de match a eu lieu en fonction des phases finales

```
print(t1["Winner"].value_counts())
```

Federer R.	735
Nadal R.	705
Djokovic N.	603
Ferrer D.	548
Murray A.	488
...	
Gomez E.	1
Harrison C.	1
Peliwo F.	1
Lee M.	1
Clezar G.	1

Name: Winner, Length: 665, dtype: int64

La colonne Winner du dataset permet de savoir combien de match chaque joueurs a remporté.

```
print(t1["Loser"].value_counts())
```

Verdasco F.	298
Lopez F.	294
Seppi A.	272
Ferrer D.	267
Garcia-Lopez G.	265
...	
Statham R.	1
Bowles C.	1
Rodriguez B.	1
Topic J.	1
Seyboth Wild T.	1

Name: Loser, Length: 1019, dtype: int64

La colonne Loser du dataset permet de savoir combien de match chaque joueurs a perdu.

```
print(t1["Comment"].value_counts())
```

Completed	31112
Retired	1072
Walkover	6
Disqualified	1

Name: Comment, dtype: int64

La colonne Comment du dataset permet de savoir comment s'est fini le match.

```
t1["Winner"].mode()[0]
```

'Federer R.'

```
gagnant =t1["Winner"].value_counts()
```

```
gagnant = list(gagnant)
```

```

gagnant = list(gagnant)
print(gagnant)

print("-----")

gagnant_top_10 = []

for i in gagnant :
    if i > 548 :
        gagnant_top_10.append(i)

print(gagnant_top_10)

[735, 705, 603, 548, 488, 484, 419, 391, 376, 358, 346, 339, 334, 330, 326, 325, 319, 317, 317, 316, 302, 279, 279, 265, 265, 260,
-----
[735, 705, 603]

```

```

t1["Winner"].value_counts(normalize=True)*100

```

```

Federer R.      2.283247
Nadal R.        2.190053
Djokovic N.     1.873194
Ferrer D.       1.702339
Murray A.       1.515952
...
Gomez E.        0.003106
Harrison C.     0.003106
Peliwo F.       0.003106
Lee M.          0.003106
Clezar G.       0.003106
Name: Winner, Length: 665, dtype: float64

```

```

t1.info()

```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 32191 entries, 8678 to 44707
Data columns (total 23 columns):
#   Column          Non-Null Count  Dtype
---  -
0   ATP              32191 non-null  int64
1   Location         32191 non-null  object
2   Tournament       32191 non-null  object
3   Date             32191 non-null  datetime64[ns]
4   Series           32191 non-null  object
5   Court            32191 non-null  object
6   Surface          32191 non-null  object
7   Round            32191 non-null  object
8   Best of          32191 non-null  int64
9   Winner           32191 non-null  object
10  Loser            32191 non-null  object
11  WRank            32191 non-null  int64
12  LRank            32191 non-null  int64
13  Wsets            32191 non-null  float64
14  Lsets            32191 non-null  float64
15  Comment          32191 non-null  object
16  PSW              32191 non-null  float64
17  PSL              32191 non-null  float64
18  B365W           32191 non-null  float64
19  B365L           32191 non-null  float64
20  elo_winner       32191 non-null  float64
21  elo_loser        32191 non-null  float64
22  proba_elo        32191 non-null  float64
dtypes: datetime64[ns](1), float64(9), int64(4), object(9)
memory usage: 5.9+ MB

```

```

t1.corr()

```

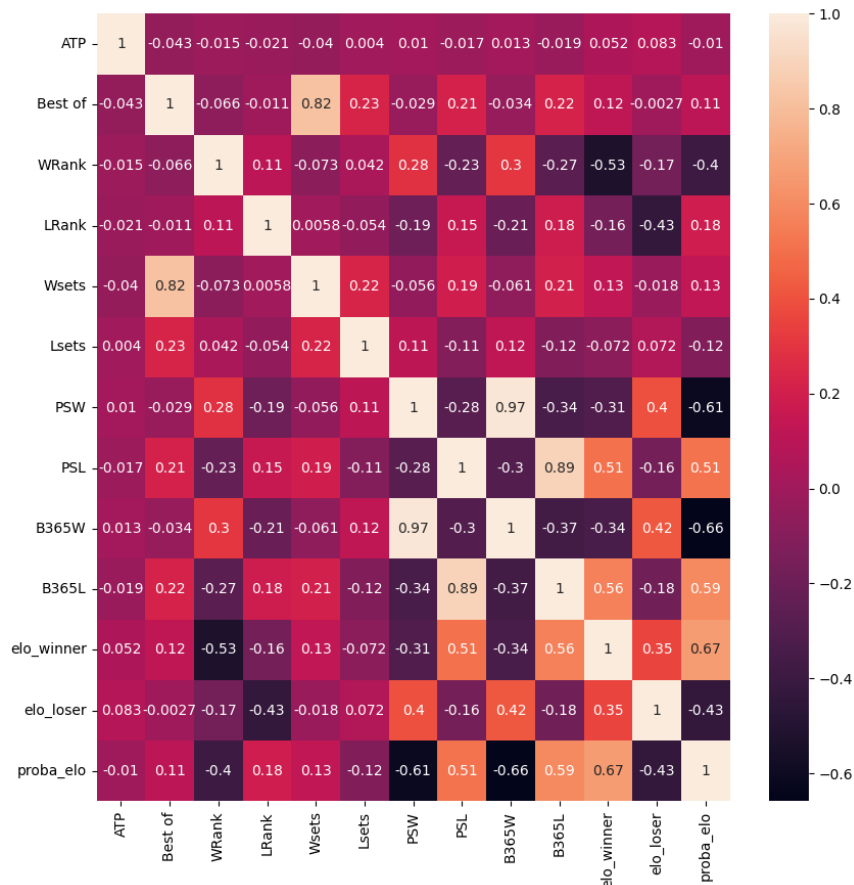


<ipython-input-24-9d0db3df0f86>:1: FutureWarning: The default value of numeric\_only  
t1.corr()

	ATP	Best of	WRank	LRank	Wsets	Lsets	PSW	
ATP	1.000000	-0.043449	-0.014871	-0.020668	-0.039516	0.003985	0.010133	-0
Best of	-0.043449	1.000000	-0.066044	-0.010952	0.823580	0.228785	-0.029271	0
WRank	-0.014871	-0.066044	1.000000	0.109297	-0.072564	0.041758	0.280820	-0
LRank	-0.020668	-0.010952	0.109297	1.000000	0.005798	-0.054226	-0.191980	0
Wsets	-0.039516	0.823580	-0.072564	0.005798	1.000000	0.224999	-0.055548	0
Lsets	0.003985	0.228785	0.041758	-0.054226	0.224999	1.000000	0.113292	-0
PSW	0.010133	-0.029271	0.280820	-0.191980	-0.055548	0.113292	1.000000	-0
PSL	-0.016996	0.210216	-0.228761	0.153665	0.192513	-0.106853	-0.276959	1
B365W	0.013167	-0.033627	0.303669	-0.206286	-0.061108	0.122653	0.974868	-0
B365L	-0.019487	0.221752	-0.266152	0.182537	0.206415	-0.117784	-0.336384	0
elo_winner	0.052287	0.121540	-0.531486	-0.163024	0.126768	-0.071757	-0.310632	0
elo_loser	0.083434	-0.002691	-0.171928	-0.434266	-0.018369	0.072293	0.396772	-0
proba_elo	-0.010171	0.108944	-0.397454	0.184876	0.129171	-0.120162	-0.612747	0

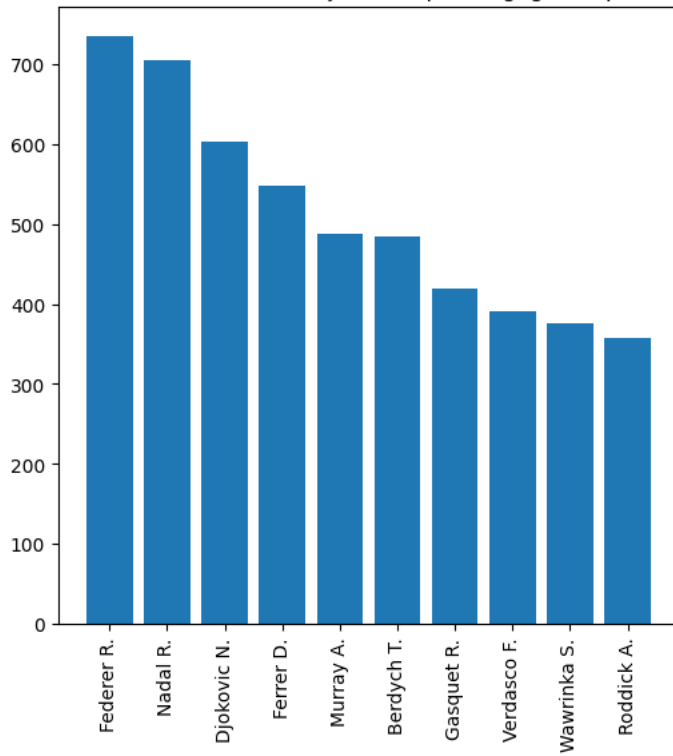
```
plt.figure(figsize=(10,10))
sns.heatmap(t1.corr(), annot=True);
# Ajouter une barre de couleur avec l'argument 'extend'
#heatmap.figure.colorbar(heatmap.collections[0], extend='both')
```

```
<ipython-input-25-b64cae3643f1>:2: FutureWarning: The default value of numeric_only
sns.heatmap(t1.corr(), annot=True);
```



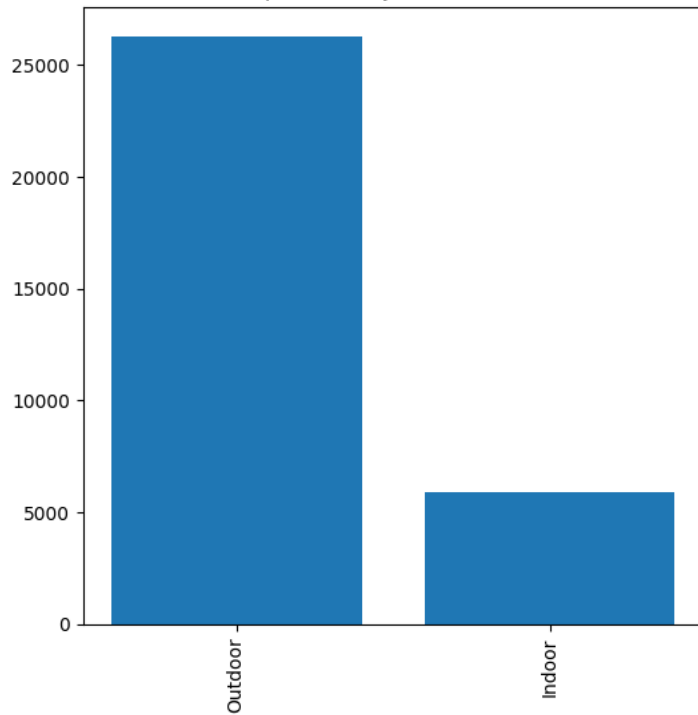
```
plt.figure(figsize=(6,6))
valeurs_x = t1["Winner"].value_counts().head(10).index
valeurs_y = t1["Winner"].value_counts().head(10).values
plt.xticks(rotation='vertical')
plt.bar(valeurs_x, valeurs_y)
plt.title("Représentation des 10 meilleurs joueurs qui ont gagné le plus de matchs");
```

Représentation des 10 meilleurs joueurs qui ont gagné le plus de matchs

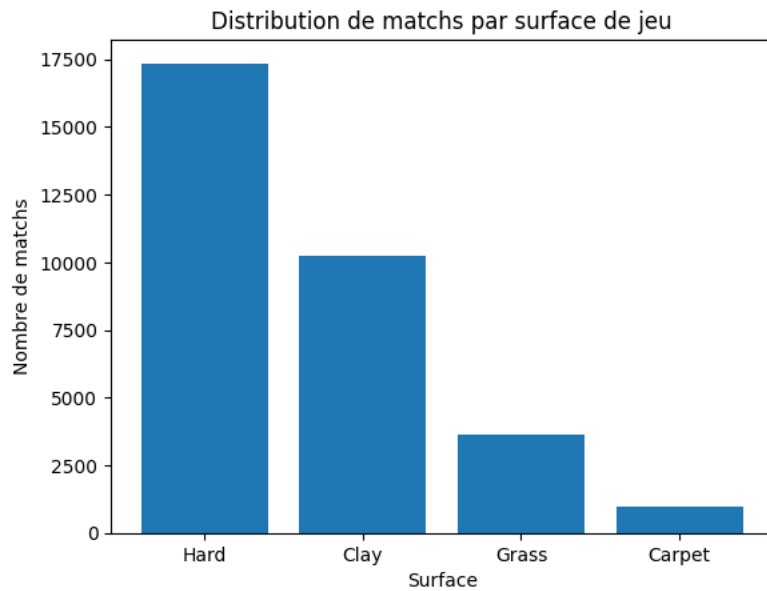


```
plt.figure(figsize=(6,6))
valeurs_x = t1["Court"].value_counts().index
valeurs_y = t1["Court"].value_counts().values
plt.xticks(rotation='vertical')
plt.bar(valeurs_x, valeurs_y)
plt.title("Nombre de matchs qui ont été joués à l'intérieur et à l'extérieur");
```

Nombre de matchs qui ont été joués à l'intérieur et à l'extérieur

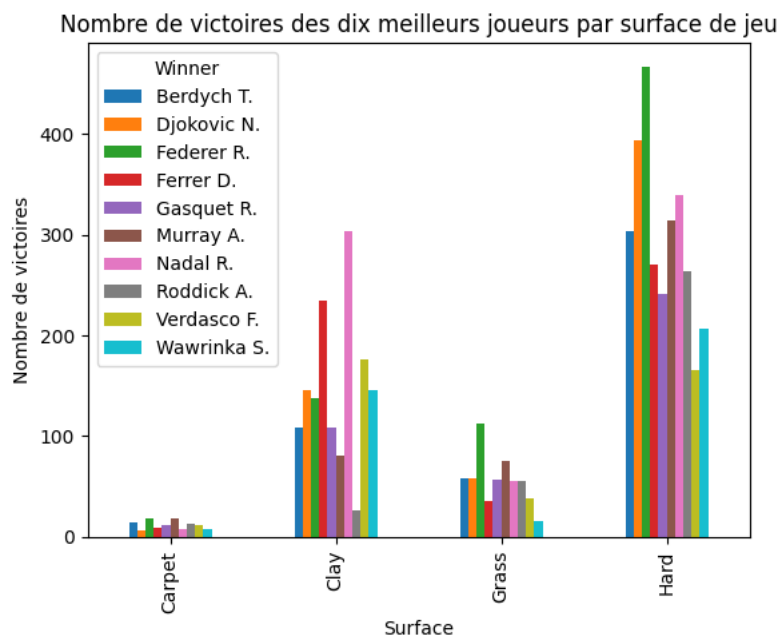


```
matches_par_surface = t1['Surface'].value_counts()
plt.bar(matches_par_surface.index, matches_par_surface.values)
plt.title('Distribution de matchs par surface de jeu')
plt.xlabel('Surface')
plt.ylabel('Nombre de matchs');
```



```
top_players = t1['Winner'].value_counts().head(10).index
df_top_players = t1[t1['Winner'].isin(top_players)]
victoires_par_surface_joueur = df_top_players.groupby(['Surface', 'Winner']).size().unstack()
victoires_par_surface_joueur.plot(kind='bar')

plt.title('Nombre de victoires des dix meilleurs joueurs par surface de jeu')
plt.xlabel('Surface')
plt.ylabel('Nombre de victoires');
```



```
import statsmodels.api

result=statsmodels.formula.api.ols("elo_winner~Winner", data=t1).fit()
table=statsmodels.api.stats.anova_lm(result)
display(table)
```

	df	sum_sq	mean_sq	F	PR(>F)
<b>Winner</b>	664.0	8.469565e+08	1.275537e+06	167.632092	0.0
<b>Residual</b>	31526.0	2.398859e+08	7.609145e+03	NaN	NaN

```
import statsmodels.api
```

```
result=statsmodels.formula.api.ols("elo_loser~Loser", data=t1).fit()
table=statsmodels.api.stats.anova_lm(result)
display(table)
```

	df	sum_sq	mean_sq	F	PR(>F)
<b>Loser</b>	1018.0	4.604219e+08	452280.862274	74.433471	0.0
<b>Residual</b>	31172.0	1.894107e+08	6076.310230	NaN	NaN

```
table=pd.crosstab(t1["Surface"], t1["Winner"])
from scipy.stats import chi2_contingency
resultat = chi2_contingency(table)
stat=resultat[0]
p_valeur=resultat[1]
print(stat)
print(p_valeur)
print("Le test renvoie un tuple de plusieurs valeurs. La première valeur ([0]) est la statistique du test et la deuxième valeur ([1]) e
```

```
10597.777929104808
```

```
0.0
```

```
Le test renvoie un tuple de plusieurs valeurs. La première valeur ([0]) est la statistique du test et la deuxième valeur ([1]) est
```

```
table=pd.crosstab(t1["Round"], t1["Winner"])
from scipy.stats import chi2_contingency
resultat = chi2_contingency(table)
stat=resultat[0]
p_valeur=resultat[1]
print(stat)
print(p_valeur)
print("Le test renvoie un tuple de plusieurs valeurs. La première valeur ([0]) est la statistique du test et la deuxième valeur ([1]) e
```

```
7671.213244316005
```

```
2.0846630272592256e-153
```

```
Le test renvoie un tuple de plusieurs valeurs. La première valeur ([0]) est la statistique du test et la deuxième valeur ([1]) est
```

## SCINDONS NOS VARIABLES EXPLICATIVES EN DEUX CATEGORIES (NUMERIQUES ET CATEGORIELLES)

```
var_num = t1.select_dtypes(["int", "float"])
```

```
var_num.head()
```

	ATP	Best of	WRank	LRank	Wsets	Lsets	PSW	PSL	B365W	B365L	elo_winner
<b>8678</b>	3	3	246	58	2.0	0.0	2.210	1.719	2.200	1.615	1486.195239
<b>8679</b>	3	3	43	6	2.0	0.0	3.250	1.385	2.875	1.364	1615.057648
<b>8680</b>	3	3	39	67	2.0	1.0	1.746	2.180	1.571	2.250	1654.468551
<b>8682</b>	2	3	97	440	2.0	0.0	1.562	2.580	1.571	2.250	1533.308961

```
var_cat = t1.select_dtypes(["object"])
```

```
var_cat.head()
```

	Location	Tournament	Series	Court	Surface	Round	Winner	Loser
<b>8678</b>	Doha	Qatar Exxon Mobil Open	International	Outdoor	Hard	1st Round	Ulihrach B.	Kiefer N.
<b>8679</b>	Doha	Qatar Exxon Mobil Open	International	Outdoor	Hard	1st Round	Youzhny M.	Schuettler R.
		Qatar						

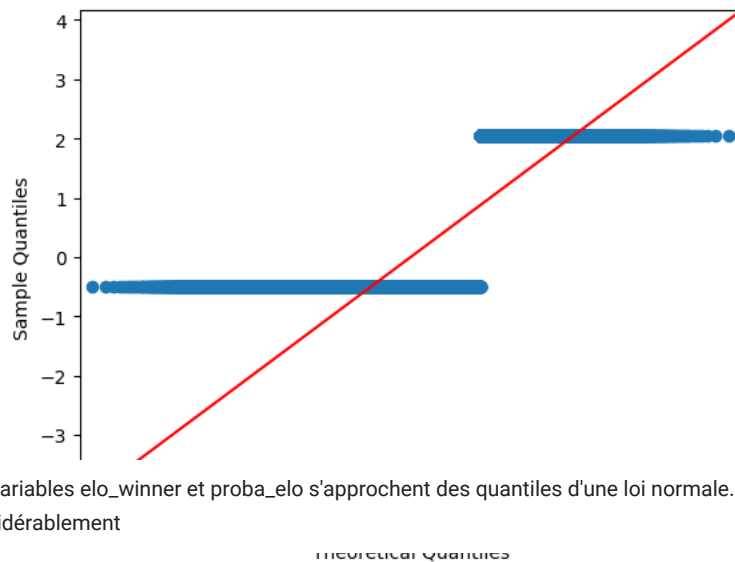
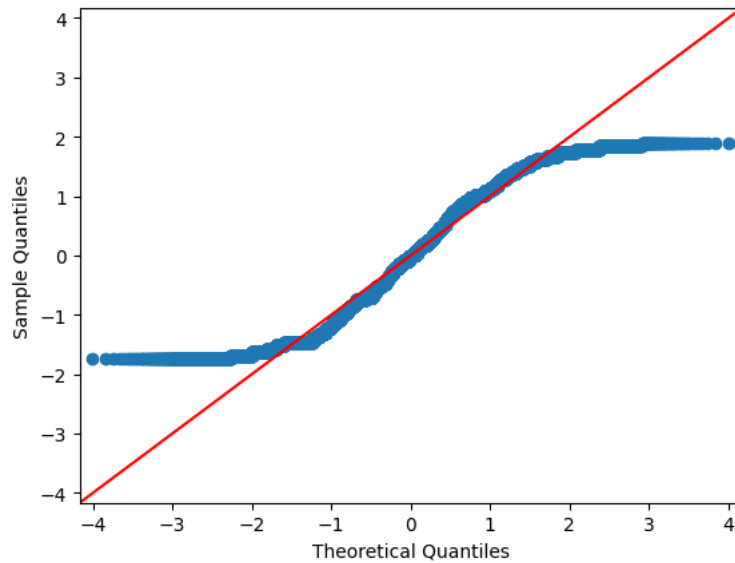
```
print("Le tournoiement le plus présent ou joué est", var_cat["Tournament"].mode()[0])
```

```
Le tournoiement le plus présent ou joué est Australian Open
```

```
import statsmodels.api as sm
```

```
for column in var_num.columns:
    print(column)
    sm.qqplot(var_num[column], line='45', fit = True)
```

```
ATP
Best of
WRank
LRank
Wsets
Lsets
PSW
PSL
B365W
B365L
elo_winner
elo_loser
proba_elo
```



Les variables elo\_winner et proba\_elo s'approchent des quantiles d'une loi normale. pour le reste ce n'est pas le cas et la distribution diffère considérablement

Au regard de cette première partie les variables suivantes sont pertinentes : Series, Court, Surface, Round, WRank, LRank, Wsets, Lsets, elo\_winner, elo\_loser et Winner.

Aussi la variable cible est : Winner.

## ✓ Pre-processing et feature engineering

```

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree
from sklearn.metrics import classification_report
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_score

```

4 |

t1.head()

	ATP	Location	Tournament	Date	Series	Court	Surface	Round	Best of	W
8678	3	Doha	Qatar Exxon Mobil Open	2004-01-05	International	Outdoor	Hard	1st Round	3	UI
8679	3	Doha	Qatar Exxon Mobil Open	2004-01-05	International	Outdoor	Hard	1st Round	3	Yo
8680	3	Doha	Qatar Exxon Mobil Open	2004-01-05	International	Outdoor	Hard	1st Round	3	Sa
8682	2	Chennai	TATA Open	2004-01-05	International	Outdoor	Hard	1st Round	3	L
8683	2	Chennai	TATA Open	2004-01-05	International	Outdoor	Hard	1st Round	3	L

Theoretical Quantiles

```

def createtarget(row):
    alea = np.random.randint(0, 2)
    if alea == 0:
        inter = row["Winner"]
        row["Winner"] = row["Loser"]
        row["Loser"] = inter
        inter = row["elo_winner"]
        row["elo_winner"] = row["elo_loser"]
        row["elo_loser"] = inter
        inter = row["WRank"]
        row["WRank"] = row["LRank"]
        row["LRank"] = inter
        inter = row["PSW"]
        row["PSW"] = row["PSL"]
        row["PSL"] = inter
        inter = row["B365W"]
        row["B365W"] = row["B365L"]
        row["B365L"] = inter
        row["target"] = 0
    else:
        row["target"] = 1
    return row

```

t1 = t1.apply(createtarget, axis=1)

|

t1.head()

	ATP	Location	Tournament	Date	Series	Court	Surface	Round	Best of	W
8678	3	Doha	Qatar Exxon Mobil Open	2004-01-05	International	Outdoor	Hard	1st Round	3	Ul
8679	3	Doha	Qatar Exxon Mobil Open	2004-01-05	International	Outdoor	Hard	1st Round	3	Yo
8680	3	Doha	Qatar Exxon Mobil Open	2004-01-05	International	Outdoor	Hard	1st Round	3	Sa
8682	2	Chennai	TATA Open	2004-01-05	International	Outdoor	Hard	1st Round	3	L

```

feat = t1[["Series", "Court", "Surface", "WRank", "LRank", "elo_winner", "elo_loser", "PSW", "PSL", "B365W", "B365L"]]
target = t1["target"]

```

```

### Utiliser la fonction createtarget pour les variables PSW, B365W, WSets, WRank. Puis appliquer la fonction via apply pour toutes les \
### Mettre des étapes d'affichage du dataframe dans un souci de clareté.

```

```

# Division des données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(feat, target, test_size=0.20, random_state=48)

```

```

# Encodage des caractéristiques
X_train_encoded = pd.get_dummies(X_train)
X_test_encoded = pd.get_dummies(X_test)

```

```

# Initialisation de l'encodeur de labels
label_encoder = LabelEncoder()

```

```

# Ajustement de l'encodeur avec toutes les valeurs uniques
unique_values = np.unique(np.concatenate([y_train, y_test]))
label_encoder.fit(unique_values)

```

```

# Appliquer l'encodage numérique à la variable cible
y_train_encoded = label_encoder.transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

```

```

# Vérifier et ajuster les colonnes
common_columns = set(X_train_encoded.columns).intersection(X_test_encoded.columns)
X_train_encoded = X_train_encoded[list(common_columns)]
X_test_encoded = X_test_encoded[list(common_columns)]

```

```

# Normalisation des données
scaler = MinMaxScaler()
scaler.fit(X_train_encoded)

```

```

X_train_scaled = scaler.transform(X_train_encoded)
X_test_scaled = scaler.transform(X_test_encoded)

```

```

# Initialisation et entraînement du classificateur RandomForest
clf1 = RandomForestClassifier(max_depth=2, random_state=0)
clf1.fit(X_train_scaled, y_train_encoded)

```

```

# Prédiction sur les données de test
y_pred = clf1.predict(X_test_scaled)

```

```

# Calcul de l'exactitude (accuracy) et du F1-score
accuracy = accuracy_score(y_test_encoded, y_pred)
f1 = f1_score(y_test_encoded, y_pred, average='weighted')

```

```

# Afficher les métriques
print("Accuracy :", accuracy)
print("F1-Score :", f1)

```

```

# Matrice de confusion
confusion = confusion_matrix(y_test_encoded, y_pred)
print("Matrice de confusion :\n", confusion)

```

```

# Tableau croisé
pd.crosstab(y_test_encoded, y_pred, rownames=["Classe réelle"], colnames=["Classe prédite"])

```



Accuracy : 0.6921882279857121  
 F1-Score : 0.6922345595776868  
 Matrice de confusion :  
 [[2196 943]  
 [1039 2261]]

Classe prédite      0      1

Classe réelle

0      2196   943

1      1039   2261

5

clf2 = DecisionTreeClassifier(random\_state=0,criterion='entropy', max\_depth=4 )

# Entraînement du modèle

clf2.fit(X\_train\_scaled, y\_train\_encoded)

# Prédiction sur les données de test

y\_pred = clf2.predict(X\_test\_scaled)

# Calcul de l'exactitude (accuracy) et du F1-score

accuracy = accuracy\_score(y\_test\_encoded, y\_pred)

f2 = f1\_score(y\_test\_encoded, y\_pred, average='weighted')

# Afficher les métriques

print("Accuracy :", accuracy)

print("F1-Score :", f2)

# Matrice de confusion

confusion = confusion\_matrix(y\_test\_encoded, y\_pred)

print("Matrice de confusion :\n", confusion)

pd.crosstab(y\_test\_encoded, y\_pred, rownames= ["Classe réelle"], colnames=["Classe prédite"])

Accuracy : 0.6915670135114148  
 F1-Score : 0.6916111163149795  
 Matrice de confusion :  
 [[2198 941]  
 [1045 2255]]

Classe prédite      0      1

Classe réelle

0      2198   941

1      1045   2255

5

from sklearn import svm

from sklearn.svm import SVC

# Créez un modèle SVM

clf3 = SVC(kernel='linear', random\_state=0)

# Entraînement du modèle

clf3.fit(X\_train\_scaled, y\_train\_encoded)

# Prédiction sur les données de test

y\_pred = clf3.predict(X\_test\_scaled)

# Calcul de l'exactitude (accuracy) et du F1-score

accuracy = accuracy\_score(y\_test\_encoded, y\_pred)

f3 = f1\_score(y\_test\_encoded, y\_pred, average='weighted')

# Afficher les métriques

print("Accuracy :", accuracy)

print("F1-Score :", f3)

# Matrice de confusion

confusion = confusion\_matrix(y\_test\_encoded, y\_pred)

print("Matrice de confusion :\n", confusion)

pd.crosstab(y\_test\_encoded, y\_pred, rownames= ["Classe réelle"], colnames=["Classe prédite"])

```

Accuracy : 0.682248796396956
F1-Score : 0.6822450257527815
Matrice de confusion :
[[2198  941]
 [1105 2195]]

```

Classe prédite	0	1
Classe réelle		
0	2198	941
1	1105	2195

X

```

from sklearn.naive_bayes import GaussianNB

clf4 = GaussianNB()

# Entraînement du modèle
clf4.fit(X_train_scaled, y_train_encoded)

# Prédiction sur les données de test
y_pred = clf4.predict(X_test_scaled)

# Calcul de l'exactitude (accuracy) et du F1-score
accuracy = accuracy_score(y_test_encoded, y_pred)
f4 = f1_score(y_test_encoded, y_pred, average='weighted')

# Afficher les métriques
print("Accuracy :", accuracy)
print("F1-Score :", f4)

# Matrice de confusion
confusion = confusion_matrix(y_test_encoded, y_pred)
print("Matrice de confusion :\n", confusion)

pd.crosstab(y_test_encoded, y_pred, rownames= ["Classe réelle"], colnames=["Classe prédite"])

```

```

Accuracy : 0.6817828855412331
F1-Score : 0.6800846072689032
Matrice de confusion :
[[1917 1222]
 [ 827 2473]]

```

Classe prédite	0	1
Classe réelle		
0	1917	1222
1	827	2473

```

from sklearn.neural_network import MLPClassifier

clf5 = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500, random_state=0)

# Entraînement du modèle
clf5.fit(X_train_scaled, y_train_encoded)

# Prédiction sur les données de test
y_pred = clf5.predict(X_test_scaled)

# Calcul de l'exactitude (accuracy) et du F1-score
accuracy = accuracy_score(y_test_encoded, y_pred)
f5 = f1_score(y_test_encoded, y_pred, average='weighted')

# Afficher les métriques
print("Accuracy :", accuracy)
print("F1-Score :", f5)

# Matrice de confusion
confusion = confusion_matrix(y_test_encoded, y_pred)
print("Matrice de confusion :\n", confusion)

pd.crosstab(y_test_encoded, y_pred, rownames= ["Classe réelle"], colnames=["Classe prédite"])

```

Accuracy : 0.6920329243671377  
F1-Score : 0.6918918950047861  
Matrice de confusion :  
[[2108 1031]  
[ 952 2348]]

Classe prédite	0	1
Classe réelle		
0	2108	1031
1	952	2348

```
from sklearn.neighbors import KNeighborsClassifier  
  
clf6 = KNeighborsClassifier(n_neighbors=7,p=2, metric='minkowski')  
  
clf6.fit(X_train_scaled, y_train_encoded)
```