PROJET 3PDQ

Choses à faire :

220 observations

Sans le rééquilibrage des données

Sans imputation

Faire un point des metrics de chaque modèle et les comparés (Je dois maitriser la significations de chaque métrics)

Pareille pour la courbe roc

Voir le calcul mathématique de chaque métrics

Installation des packages

In [1]:
```
# Packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as st
from sklearn.ensemble import IsolationForest
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
dataset = pd.read_csv("PDQ_3 -220observations.csv")
dataset
```

Out[2]:

	N°de_Patient	Centre	Initiales	Sexe	Age	Date_Naissance	Date_visite1	Date_visite2	EVA_mean	EVA_max	...	HADa.5	HADd.5	HADa.6	HADd
0	1	1	BM	M	71.0	juil.-46	2019/05/22	2019/05/22	50.0	60	...	0.0	0.0	0.0	0
1	2	1	BM	M	49.0	sept.-69	2019/06/28	2019/06/28	35.0	80	...	0.0	1.0	1.0	1
2	3	1	NC	F	61.0	août-57	2019/07/05	2019/07/05	50.0	75	...	2.0	2.0	3.0	1
3	4	1	GE	F	65.0	juil.-53	2019/07/10	2019/07/10	54.0	80	...	1.0	0.0	2.0	1
4	5	1	ML	F	59.0	oct.-58	2019/07/17	2019/07/17	51.0	69	...	1.0	2.0	3.0	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
215	221	14	GM	F	65.0	mai-55	2022/03/10	2022/03/11	70.0	80	...	2.0	2.0	2.0	1
216	222	15	MM	M	76.0	févr.-45	2022/06/02	2022/06/02	80.0	100	...	0.0	0.0	2.0	0
217	223	15	PA	F	64.0	nov.-56	2022/06/21	2022/06/21	45.0	70	...	0.0	2.0	3.0	0
218	224	15	TL	M	60.0	févr.-61	2022/06/30	2022/06/30	40.0	70	...	2.0	0.0	0.0	2
219	225	15	MC	F	45.0	nov.-76	2022/07/19	2022/07/19	88.0	90	...	2.0	2.0	1.0	1

220 rows × 298 columns

In [3]:
```
dataset.shape
```

Out[3]:(220, 298)

In [4]:
```
print(dataset.columns.tolist())
```

['N°de_Patient', 'Centre ', 'Initiales', 'Sexe', 'Age', 'Date_Naissance', 'Date_visite1', 'Date_visite2', 'EVA_mean', 'EVA_max', 'Inclusion1', 'Inclusion2', 'Inclusion 3', 'Inclusion4', 'Inclusion5', 'Inclusion6', 'Inclusion7', 'Non_Inclusion1', 'Non_Inclusion2', 'Non_Inclusion3', 'Non_Inclusion4', 'MOCA1', 'MOCA2', 'MOCA3', 'M OCA4', 'MOCA5', 'MOCA6', 'MOCA7', 'MOCA8', 'MOCA9', 'MOCA10', 'MOCA_Total', 'superieure_gauche', 'superieure_droite', 'inferieure_gauche', 'inferieure _droite', 'Tronc', 'tete', 'Localisation', 'item1', 'item2', 'item3', 'item4', 'item5', 'item6', 'item7', 'item8', 'item9', 'item10', 'item11', 'item12', 'item13', 'item14', 'item15' 'item16', 'item17', 'item18', 'item19', 'item20', 'item21', 'item22', 'item23', 'item24', 'item25', 'item26', 'item27', 'item28', 'item29', 'item30', 'item31', 'item32', 'item3 3', 'Douleur_Centrale', 'DN4.1', 'DN4.2', 'DN4.3', 'DN4.4', 'DN4.5', 'DN4.6', 'DN4.7', 'DN4.8', 'DN4.9', 'DN4.10', 'DN4.Total', 'KPPS1_s', 'KPPS1_f', 'Domain1_T otal', 'KPPS2_s', 'KPPS2_f', 'KPPS3_s', 'KPPS3_f', 'Domain2_Total', 'KPPS4_s', 'KPPS4_f', 'KPPS5_s', 'KPPS5_f', 'KPPS6_s', 'KPPS6_f', 'Domain3_Total', ' KPPS7_s', 'KPPS7_f', 'KPPS8_s', 'KPPS8_f', 'Domain4_Total', 'KPPS9_s', 'KPPS9_f', 'KPPS10_s', 'KPPS10_f', 'KPPS11_s', 'KPPS11_f', 'Domain5_Total', 'K PPS12_s', 'KPPS12_f', 'KPPS13_s', 'KPPS13_f', 'Domain6_Total', 'KPPS14_s', 'KPPS14_f', 'Domain7_Total', 'KPPS_Total_Score', 'MDS_UPDRS1.A', 'MDS _UPDRS1.1', 'MDS_UPDRS1.2', 'MDS_UPDRS1.3', 'MDS_UPDRS1.4', 'MDS_UPDRS1.5', 'MDS_UPDRS1.6', 'MDS_UPDRS1.6a', 'MDS_UPDRS1.7', 'MDS _UPDRS1.8', 'MDS_UPDRS1.9', 'MDS_UPDRS1.10', 'MDS_UPDRS1.11', 'MDS_UPDRS1.12', 'MDS_UPDRS1.13', 'P1_MDS_UPDRS', 'MDS_UPDRS2.1', ' MDS_UPDRS2.2', 'MDS_UPDRS2.3', 'MDS_UPDRS2.4', 'MDS_UPDRS2.5', 'MDS_UPDRS2.6', 'MDS_UPDRS2.7', 'MDS_UPDRS2.8', 'MDS_UPDRS2.9', 'M DS_UPDRS2.10', 'MDS_UPDRS2.11', 'MDS_UPDRS2.12', 'MDS_UPDRS2.13', 'P2_MDS_UPDRS', 'MDS_UPDRS3a', 'MDS_UPDRS3b', 'MDS_UPDRS3c', ' MDS_UPDRS3.C1', 'MDS_UPDRS3.1', 'MDS_UPDRS3.2', 'MDS_UPDRS3.3a', 'MDS_UPDRS3.3b', 'MDS_UPDRS3.3c', 'MDS_UPDRS3.3d', 'MDS_UPDRS 3.3e', 'MDS_UPDRS3.4a', 'MDS_UPDRS3.4b', 'MDS_UPDRS3.5a', 'MDS_UPDRS3.5b', 'MDS_UPDRS3.6a', 'MDS_UPDRS3.6b', 'MDS_UPDRS3.7a', 'MDS_ UPDRS3.7b', 'MDS_UPDRS3.8a', 'MDS_UPDRS3.8b', 'MDS_UPDRS3.9', 'MDS_UPDRS3.10', 'MDS_UPDRS3.11', 'MDS_UPDRS3.12', 'MDS_UPDRS3.13', 'MDS_UPDRS3.14', 'MDS_UPDRS3.15a', 'MDS_UPDRS3.15b', 'MDS_UPDRS3.16a', 'MDS_UPDRS3.16b', 'MDS_UPDRS3.17a', 'MDS_UPDRS3.17b', 'MD S_UPDRS3.17c', 'MDS_UPDRS3.17d', 'MDS_UPDRS3.17e', 'MDS_UPDRS3.18', 'P3_MDS_UPDRS', 'MDS_UPDRSdiskiné', 'MDS_UPDRSinterféré', 'MDS_ UPDRS_Hoehn&Yahr', 'MDS_UPDRS4.1', 'MDS_UPDRS4.2', 'MDS_UPDRS4.3', 'MDS_UPDRS4.4', 'MDS_UPDRS4.5', 'MDS_UPDRS4.6', 'P4_MDS_UPDR S', 'MDS_UPDS_Total', 'McGilla1', 'McGilla2', 'McGilla3', 'McGilla4', 'McGilla5', 'McGilla6', 'McGilla7', 'McGilla8', 'McGilla9', 'McGilla10', 'McGilla11', 'McGilla1 2', 'McGilla13', 'McGilla14', 'McGilla15', 'McGillb', 'McGill_Total', 'BPIa', 'BPIb', 'BPIc', 'BPId', 'BPIe', 'BPIf', 'BPIg', 'BPI_Total', 'PCS1', 'PCS2', 'PCS3', 'PCS4', ' PCS5', 'PCS6', 'PCS7', 'PCS8', 'PCS9', 'PCS10', 'PCS11', 'PCS12', 'PCS13', 'PCS_Total', 'PDQ1', 'PDQ2', 'PDQ3', 'PDQ4', 'PDQ5', 'PDQ6', 'PDQ7', 'PDQ8', ' PDQ9', 'PDQ10', 'PDQ11', 'PDQ12', 'PDQ13', 'PDQ14', 'PDQ15', 'PDQ16', 'PDQ17', 'PDQ18', 'PDQ19', 'PDQ20', 'PDQ21', 'PDQ22', 'PDQ23', 'PDQ24', 'PDQ2 5', 'PDQ26', 'PDQ27', 'PDQ28', 'PDQ29', 'PDQ30', 'PDQ31', 'PDQ32', 'PDQ33', 'PDQ34', 'PDQ35', 'PDQ36', 'PDQ37', 'PDQ38', 'PDQ39', 'PDQ_Total', 'PDQ_ Score_Complète', ' PDQ_Résultat_Pourcentage', 'HADa.1', 'HADd.1', 'HADa.2', 'HADd.2', 'HADa.3', 'HADd.3', 'HADa.4', 'HADd.4', 'HADa.5', 'HADd.5', 'HADa. 6', 'HADd.6', 'HADa.7', 'HADd.7', 'Anxiété_Total', 'Depression_Total', 'Total_Score', 'Conformité_au_protocol']

In [5]:
```
df= dataset[['item1', 'item2', 'item3', 'item4', 'item5', 'item6', 'item7', 'item8', 'item9', 'item10', 'item11', 'item12', 'item13', 'item14', 'item15', 'item16', 'item17
 'item19', 'item20', 'item21', 'item22', 'item23', 'item24', 'item25', 'item26', 'item27', 'item28','Douleur_Centrale']]
df
```

Out[5]:

	item1	item2	item3	item4	item5	item6	item7	item8	item9	item10	...	item20	item21	item22	item23	item24	item25	item26	item27	item28
0	0.0	1.0	1	0.0	1	0	1.0	0	0.0	0	...	0	0	1	1	0	1	0	0	0
1	1.0	0.0	0	1.0	0	0	1.0	1	0.0	0	...	0	0	1	1	1	1	1	0	0
2	0.0	0.0	0	0.0	1	0	1.0	0	0.0	0	...	0	0	0	0	0	0	1	0	0
3	0.0	1.0	1	0.0	0	0	1.0	0	1.0	0	...	0	0	0	1	1	0	0	0	0
4	1.0	0.0	1	1.0	1	0	0.0	1	0.0	0	...	0	1	0	1	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
215	0.0	1.0	1	0.0	0	0	1.0	1	0.0	0	...	0	0	1	1	1	1	1	0	0
216	1.0	1.0	1	0.0	1	0	1.0	0	0.0	0	...	0	0	0	1	1	1	0	1	0
217	0.0	0.0	0	0.0	0	0	0.0	0	1.0	1	...	0	0	0	1	1	0	0	0	0
218	0.0	0.0	0	0.0	1	0	0.0	1	0.0	1	...	0	0	1	1	1	1	1	1	0
219	1.0	0.0	0	0.0	1	1	1.0	0	0.0	0	...	1	1	1	1	1	1	1	1	1

220 rows × 29 columns

In [6]:
```
df.shape
```

Out[6]:(220, 29)

In [7]:
```
from IPython.display import Image

# Afficher l'image
Image(filename="C:/Users/aliah/Downloads/Capture d’écran 2023-04-21 140544.png")
```

Out[7]:

item1	Numeric	12	0	Présente-t-elle un Brûlure ?
item2	Numeric	12	0	un Etau
item3	Numeric	1	0	une compression
item4	Numeric	12	0	des décharges électriques
item5	Numeric	1	0	des Elancements
item6	Numeric	1	0	Froid douleureux
item7	Numeric	12	0	Crampe
item8	Numeric	1	0	Douleur sourde
item9	Numeric	12	0	coups de couteau
item10	Numeric	1	0	Piqûre
item11	Numeric	12	0	Broiement
item12	Numeric	1	0	Profonde
item13	Numeric	1	0	Lancinante
item14	Numeric	12	0	est-elle associée aux Fourmillements ?
item15	Numeric	1	0	est-elle associée aux picotements
item16	Numeric	1	0	est-elle associée aux Démangeaisons
item17	Numeric	12	0	est-elle associée aux Engourdissement
item18	Numeric	12	0	La douleur est augmenté par le Forttement sur la zone douleureuse ?
item19	Numeric	1	0	La douleur est augmenté par le pression sur la zone douleureuse
item20	Numeric	1	0	La douleur est augmenté par le contact acen un objet froid sur la zone douleureuse
item21	Numeric	1	0	La douleur est augmenté par le contact acen un objet chaud sur la zone douleureuse
item22	Numeric	1	0	Elle était le premier symptôme de la maladie ?
item23	Numeric	1	0	Elle se situe du coté le plus aateint de la maladie
item24	Numeric	1	0	Elle augmente quand l'état moteur s'aggrave
item25	Numeric	1	0	Elle s'ameliore par la prise des médicaments antiparkinsoniens
item26	Numeric	1	0	Elle est présente la nuit
item27	Numeric	1	0	Elle est présente de façon diffuse sur vôtre corps
item28	Numeric	1	0	Elle se déplace d'un endroit à l'autre de vôtre corps
DE1	Numeric	12	0	Une étiologie traumatique, orthopédique ou rhumatologique ?
DE2	Numeric	1	0	Est-elle musculo-squelettique ?
DE3	Numeric	1	0	Est-elle de type radiculaire ?
DE4	Numeric	1	0	Est elle dûe à la syndrome des jampes sans repos ?
DE5	Numeric	1	0	Est-elle dtstonique ?
Diagnostic_...	Numeric	12	0	La douleur Parkinsonienne Centrale

Vérifier le type de chaque variable

In [8]:
```
df.select_dtypes(object).columns
```

Out[8]:
```
Index(['item3', 'item5', 'item6', 'item8', 'item10', 'item12', 'item13',
       'item15', 'item16', 'item19', 'item20', 'item21', 'item22', 'item23',
       'item24', 'item25', 'item26', 'item27', 'item28'],
      dtype='object')
```

Conversion de toutes les variables en float

In [9]:
```
# Parcours de chaque colonne du DataFrame
for column in df.columns:
    # Conversion de la colonne en type float en ignorant les erreurs de conversion
    df[column] = pd.to_numeric(df[column], errors='coerce')

# Afficher les types de données des colonnes
print(df.dtypes)
```

```
item1              float64
item2              float64
item3              float64
item4              float64
item5              float64
item6              float64
item7              float64
item8              float64
item9              float64
item10             float64
item11             float64
item12             float64
item13             float64
item14             float64
item15             float64
item16             float64
item17             float64
item18             float64
item19             float64
item20             float64
item21             float64
item22             float64
item23             float64
item24             float64
item25             float64
item26             float64
item27             float64
item28             float64
Douleur_Centrale     int64
dtype: object
```

In [10]:
```
# Parcours de chaque colonne du DataFrame
for column in df.columns:
    # Conversion de la colonne en type float
    df[column] = df[column].astype(float)

# Afficher les types de données des colonnes
print(df.dtypes)
```

```
item1              float64
item2              float64
item3              float64
item4              float64
item5              float64
item6              float64
item7              float64
item8              float64
item9              float64
item10             float64
item11             float64
item12             float64
item13             float64
item14             float64
item15             float64
item16             float64
item17             float64
item18             float64
item19             float64
item20             float64
item21             float64
item22             float64
item23             float64
item24             float64
item25             float64
item26             float64
item27             float64
item28             float64
Douleur_Centrale   float64
dtype: object
```

Gestion des données manquantes

In [11]:
```
#Compter le nombre de valeurs manquantes par variable
missing_values_count = df.isnull().sum()

# Afficher le nombre de valeurs manquantes par variable
print("Nombre de valeurs manquantes par variable :\n", missing_values_count)

# Afficher les variables qui contiennent des valeurs manquantes
missing_variables = df.columns[df.isnull().any()].tolist()
print("\nVariables avec des valeurs manquantes :", missing_variables)
```

```
Nombre de valeurs manquantes par variable :
 item1            3
item2             3
item3             3
item4             3
item5             5
item6             5
item7             2
item8             3
item9             3
item10            4
item11            3
item12            5
item13            4
item14            1
item15            2
item16            2
item17            1
item18            2
item19            3
item20            2
item21            3
item22            3
item23            4
item24            6
item25            3
item26            3
item27            4
item28            2
Douleur_Centrale    0
dtype: int64

Variables avec des valeurs manquantes : ['item1', 'item2', 'item3', 'item4', 'item5', 'item6', 'item7', 'item8', 'item9', 'item10', 'item11', 'item12', 'item13', 'item14', 'it
em15', 'item16', 'item17', 'item18', 'item19', 'item20', 'item21', 'item22', 'item23', 'item24', 'item25', 'item26', 'item27', 'item28']
In [12]: df.isnull().sum().sum()

Out[12]:87
In [13]: # Calculer le nombre total de cellules dans le DataFrame
         total_cells = df.shape[0] * df.shape[1]
         total_cells

Out[13]:6380
In [14]: # Calculer le nombre total de valeurs manquantes dans le DataFrame
         total_missing = df.isnull().sum().sum()

         # Calculer la proportion totale de valeurs manquantes dans le DataFrame
         proportion_missing = total_missing / total_cells * 100

         # Afficher la proportion totale de valeurs manquantes en pourcentage
         print('Proportion totale de valeurs manquantes : {:.2f}%'.format(proportion_missing))

Proportion totale de valeurs manquantes : 1.36%
In [15]: proportion_missing /100

Out[15]:0.013636363636363636
In [16]: # Suppression des données manquantes
         df = df.dropna()
         #Vérifier à nouveau
         df.isnull().sum().sum()

Out[16]:0
In [17]: df.shape

Out[17]:(200, 29)
In [18]: df
```

Out[18]:

	item1	item2	item3	item4	item5	item6	item7	item8	item9	item10	...	item20	item21	item22	item23	item24	item25	item26	item27	item2
0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0
1	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	...	0.0	0.0	1.0	1.0	1.0	1.0	1.0	0.0	0
2	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0
3	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	...	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0
4	1.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	...	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
215	0.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	...	0.0	0.0	1.0	1.0	1.0	1.0	1.0	0.0	0
216	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	0
217	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	...	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0
218	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	...	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0
219	1.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	...	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1

200 rows × 29 columns

MODELISATION

In [19]:
```
# 1) Créer une matrice des variables indépendantes et le vecteur de la variable dépendante.
# X est la matrice et Y est le vecteur
# La matrice des variables indépendantes est aussi appeelée matrice de featuresμ

X = df.drop('Douleur_Centrale', axis=1)  # Supprimer la colonne "target" de la matrice X
Y = df['Douleur_Centrale']               # Sélectionner uniquement la colonne "target" pour Y
```

In [20]:
```
X
```

Out[20]:

	item1	item2	item3	item4	item5	item6	item7	item8	item9	item10	...	item19	item20	item21	item22	item23	item24	item25	item26	item2
0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0
1	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	0
2	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0
3	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	...	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0
4	1.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	...	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
215	0.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	...	1.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	0
216	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	1
217	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	...	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0
218	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	...	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1
219	1.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	...	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1

200 rows × 28 columns

In [21]:
```
Y
```

Out[21]:
```
0      1.0
1      1.0
2      0.0
3      0.0
4      1.0
      ...
215    0.0
216    1.0
217    1.0
218    1.0
219    1.0
Name: Douleur_Centrale, Length: 200, dtype: float64
```

Séparation du dataset en training_set et en test_set

In [22]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/3, random_state= 0)

# Calculer la proportion de chaque ensemble
train_prop = len(X_train) / len(X)
test_prop = len(X_test) / len(X)

print("Proportion de données dans l'ensemble d'entraînement: {:.2f}".format(train_prop))
```

```
print("Proportion de données dans l'ensemble de test: {:.2f}".format(test_prop))
```

```
Proportion de données dans l'ensemble d'entraînement: 0.67
Proportion de données dans l'ensemble de test: 0.34
```

1- Regression Logistique

In [23]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE

# Créer une instance du modèle de régression logistique
RL = LogisticRegression()

# Créer une instance de RFE avec le modèle de régression logistique
rfe = RFE(estimator=RL, n_features_to_select=6)

# Adapter RFE sur les données d'entraînement
rfe.fit(X_train, Y_train)

# Obtenir les indices des fonctionnalités sélectionnées
selected_features_indices = rfe.get_support(indices=True)

# Obtenir les noms des fonctionnalités sélectionnées
selected_features_names = X_train.columns[selected_features_indices]

# Entraîner le modèle de régression logistique sur les fonctionnalités sélectionnées
RL.fit(X_train[selected_features_names], Y_train)

# Afficher les variables du feature selection
print("Variables du feature selection : ")
print(selected_features_names)
```

```
Variables du feature selection :
Index(['item4', 'item5', 'item19', 'item24', 'item25', 'item27'], dtype='object')
```

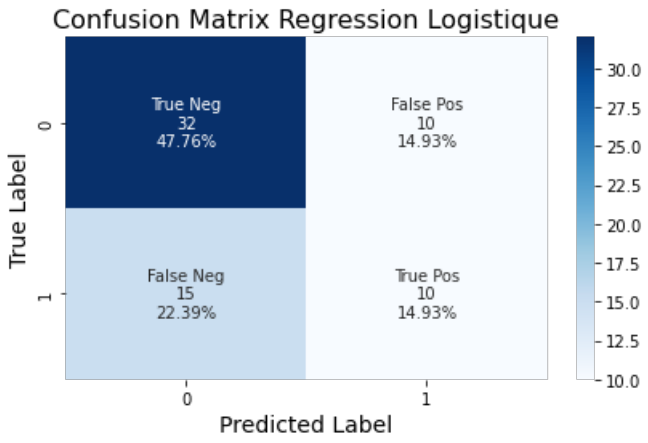
In [24]:
```
# Prédire les classes pour les données de test
Y_pred1 = rfe.predict(X_test)
Y_pred1
```

Out[24]:
```
array([1., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0.,
       1., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 1.,
       0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
       1., 0., 1., 0., 0., 1., 1., 0., 0., 0., 0., 0., 1., 1., 0., 1.])
```

In [25]:
```
from sklearn.metrics import confusion_matrix
CM1 = confusion_matrix(Y_test, Y_pred1)
CM1
```

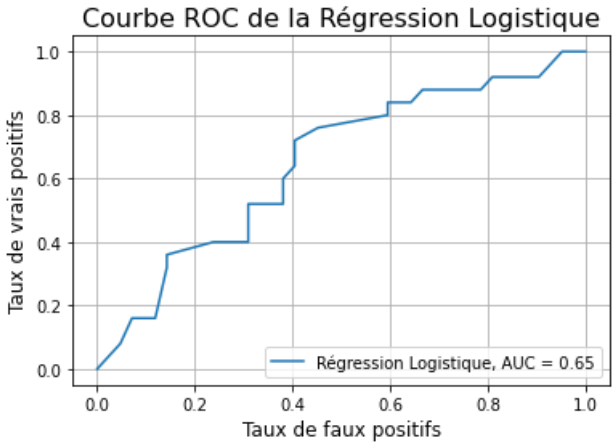
Out[25]:
```
array([[32, 10],
       [15, 10]], dtype=int64)
```

In [26]:
```
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ['{0:0.0f}'.format(value) for value in
                CM1.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
                     CM1.flatten()/np.sum(CM1)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(CM1, annot=labels, fmt='', cmap='Blues')
plt.xlabel('Predicted Label', fontsize=14)
plt.ylabel('True Label', fontsize=14)
plt.title('Confusion Matrix Regression Logistique', fontsize=16)
plt.tight_layout()
plt.show()
```



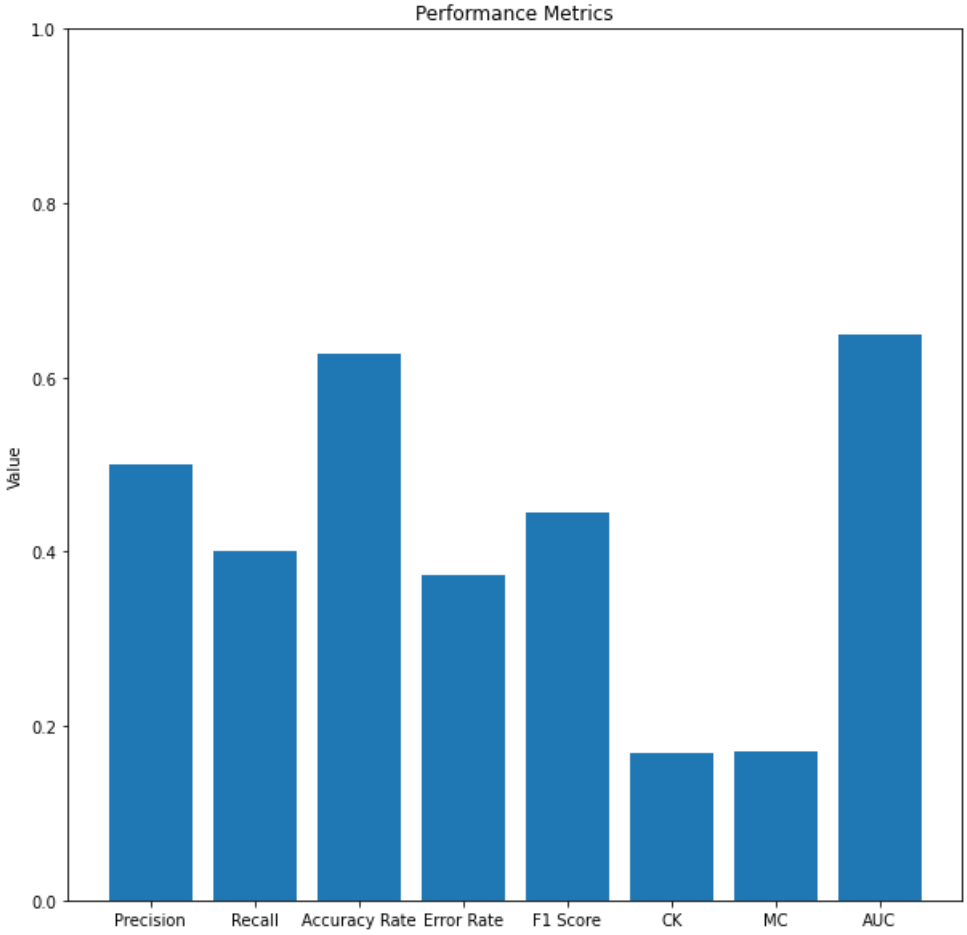
In [52]:
```
from sklearn import metrics
```

```
y_pred_proba1 = rfe.predict_proba(X_test)[:, 1]
fpr, tpr, _ = metrics.roc_curve(Y_test, y_pred_proba1)
auc = metrics.roc_auc_score(Y_test, y_pred_proba1)

plt.plot(fpr, tpr, label="Régression Logistique, AUC = {:.2f}".format(auc))
plt.legend(loc="lower right")
plt.title('Courbe ROC de la Régression Logistique', fontsize=16)
plt.xlabel('Taux de faux positifs', fontsize=12)
plt.ylabel('Taux de vrais positifs', fontsize=12)
plt.grid(True)
plt.show()
```



In [46]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, cohen_kappa_score, matthews_corrcoef

Accuracy_Rate1 = accuracy_score(Y_test, Y_pred1)
Error_rate1 = 1 - Accuracy_Rate1
F1_score1 = f1_score(Y_test, Y_pred1)
Precision1 = precision_score(Y_test, Y_pred1)
Recall1 = recall_score(Y_test, Y_pred1)
CK1 = cohen_kappa_score (Y_test,Y_pred1)
MC1 = matthews_corrcoef(Y_test,Y_pred1)
auc1 = metrics.roc_auc_score(Y_test, y_pred_proba1)

print("Precision : {:.2f}".format(Precision1))
print("Recall : {:.2f}".format(Recall1))
print("Accuracy rate: ", Accuracy_Rate1)
print("Error rate: ", Error_rate1)
print("F1_score: ", F1_score1)
print("CK:", CK1)
print("MC:", MC1)
print("AUC:", auc1)


# create a list of metric names and values
metric_names = ["Precision", "Recall", "Accuracy Rate", "Error Rate", "F1 Score", "CK", "MC","AUC"]
metric_values = [Precision1, Recall1, Accuracy_Rate1, Error_rate1, F1_score1, CK1, MC1,auc1]

# create a bar chart
fig, ax = plt.subplots(figsize=(10,10))
ax.bar(metric_names, metric_values)
ax.set_ylabel('Value')
ax.set_ylim([0,1])
ax.set_title('Performance Metrics')
plt.show()
```

Precision : 0.50
Recall : 0.40
Accuracy rate: 0.6268656716417911
Error rate: 0.3731343283582089
F1_score: 0.4444444444444445
CK: 0.1687344913151364
MC: 0.171115891808986
AUC: 0.6490476190476191



2- Random Forest

In [29]:
```
from sklearn.ensemble import RandomForestClassifier
from boruta import BorutaPy

# Convertir X_train en tableau NumPy
X_train_np = X_train.values

# Créer une instance du modèle Random Forest
RF = RandomForestClassifier(n_estimators=100000, n_jobs=1, max_depth=2,random_state = 0)

# Créer une instance de Boruta
feat_selector = BorutaPy(RF, n_estimators='auto', max_iter=100, verbose=2, random_state = 0)
# Adapter Boruta sur les données d'entraînement
feat_selector.fit(X_train_np, Y_train)

# Obtenir les indices des fonctionnalités sélectionnées
selected_features_indices = feat_selector.support_

# Sélectionner les fonctionnalités avec Boruta
selected_features = X_train.columns[feat_selector.support_]

# Adapter le modèle Random Forest sur les fonctionnalités sélectionnées
RF.fit(X_train[selected_features], Y_train)

# Afficher les variables du feature selection
print("Variables du feature selection : ")
print(selected_features_names)
```

Iteration: 1 / 100
Confirmed: 0
Tentative: 28
Rejected: 0
Iteration: 2 / 100
Confirmed: 0
Tentative: 28
Rejected: 0
Iteration: 3 / 100

Iteration: 3 / 100
Confirmed: 0
Tentative: 28
Rejected: 0
Iteration: 4 / 100
Confirmed: 0
Tentative: 28
Rejected: 0
Iteration: 5 / 100
Confirmed: 0
Tentative: 28
Rejected: 0
Iteration: 6 / 100
Confirmed: 0
Tentative: 28
Rejected: 0
Iteration: 7 / 100
Confirmed: 0
Tentative: 28
Rejected: 0
Iteration: 8 / 100
Confirmed: 0
Tentative: 9
Rejected: 19
Iteration: 9 / 100
Confirmed: 2
Tentative: 7
Rejected: 19
Iteration: 10 / 100
Confirmed: 2
Tentative: 7
Rejected: 19
Iteration: 11 / 100
Confirmed: 2
Tentative: 7
Rejected: 19
Iteration: 12 / 100
Confirmed: 2
Tentative: 7
Rejected: 19
Iteration: 13 / 100
Confirmed: 2
Tentative: 7
Rejected: 19
Iteration: 14 / 100
Confirmed: 2
Tentative: 7
Rejected: 19
Iteration: 15 / 100
Confirmed: 2
Tentative: 7
Rejected: 19
Iteration: 16 / 100
Confirmed: 3
Tentative: 6
Rejected: 19
Iteration: 17 / 100
Confirmed: 3
Tentative: 6
Rejected: 19
Iteration: 18 / 100
Confirmed: 3
Tentative: 6
Rejected: 19
Iteration: 19 / 100
Confirmed: 3
Tentative: 6
Rejected: 19
Iteration: 20 / 100
Confirmed: 3
Tentative: 6
Rejected: 19
Iteration: 21 / 100
Confirmed: 3
Tentative: 6
Rejected: 19
Iteration: 22 / 100
Confirmed: 3
Tentative: 6
Rejected: 19
Iteration: 23 / 100
Confirmed: 3
Tentative: 6

Rejected: 19
Iteration: 24 / 100
Confirmed: 3
Tentative: 6
Rejected: 19
Iteration: 25 / 100
Confirmed: 3
Tentative: 6
Rejected: 19
Iteration: 26 / 100
Confirmed: 4
Tentative: 5
Rejected: 19
Iteration: 27 / 100
Confirmed: 4
Tentative: 5
Rejected: 19
Iteration: 28 / 100
Confirmed: 4
Tentative: 5
Rejected: 19
Iteration: 29 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 30 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 31 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 32 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 33 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 34 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 35 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 36 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 37 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 38 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 39 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 40 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 41 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 42 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 43 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 44 / 100
Confirmed: 4

Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 45 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 46 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 47 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 48 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 49 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 50 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 51 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 52 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 53 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 54 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 55 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 56 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 57 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 58 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 59 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 60 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 61 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 62 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 63 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 64 / 100
Confirmed: 4
Tentative: 3
Rejected: 21

Iteration: 65 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 66 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 67 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 68 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 69 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 70 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 71 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 72 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 73 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 74 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 75 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 76 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 77 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 78 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 79 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 80 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 81 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 82 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 83 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 84 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 85 / 100
Confirmed: 4
Tentative: 3

Rejected: 21
Iteration: 86 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 87 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 88 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 89 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 90 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 91 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 92 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 93 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 94 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 95 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 96 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 97 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 98 / 100
Confirmed: 4
Tentative: 3
Rejected: 21
Iteration: 99 / 100
Confirmed: 4
Tentative: 3
Rejected: 21

BorutaPy finished running.

Iteration: 100 / 100
Confirmed: 4
Tentative: 2
Rejected: 21
Variables du feature selection :
Index(['item4', 'item5', 'item19', 'item24', 'item25', 'item27'], dtype='object')

In [30]:
```
# Prédire les classes pour les données de test en utilisant les mêmes fonctionnalités sélectionnées
Y_pred2 = RF.predict(X_test[selected_features])

# Afficher les prédictions
print(Y_pred2)
```

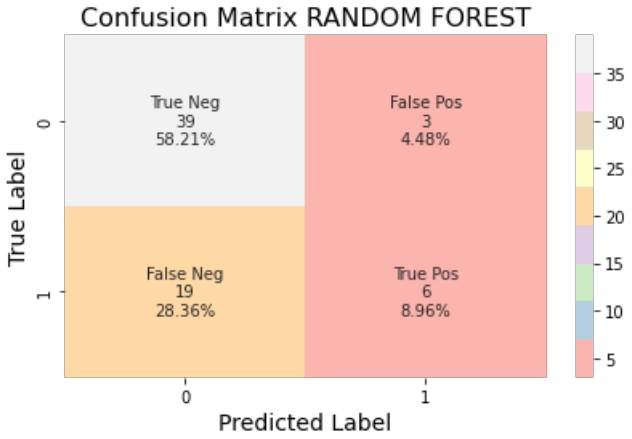
```
[1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
```

In [31]:
```
CM2 = confusion_matrix(Y_test, Y_pred2)
CM2
```

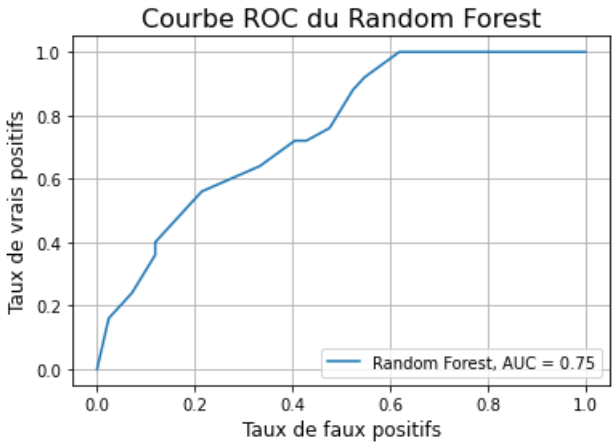
Out[31]:
```
array([[39,  3],
       [19,  6]], dtype=int64)
```

In [32]:
```
group_names = ['True Neg','False Pos','False Neg','True Pos']
```

```
group_counts = ['{0:0.0f}'.format(value) for value in
             CM2.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
                  CM2.flatten()/np.sum(CM2)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
         zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(CM2, annot=labels, fmt='', cmap='Pastel1')
plt.xlabel('Predicted Label', fontsize=14)
plt.ylabel('True Label', fontsize=14)
plt.title('Confusion Matrix RANDOM FOREST', fontsize=16)
plt.tight_layout()
plt.show()
```



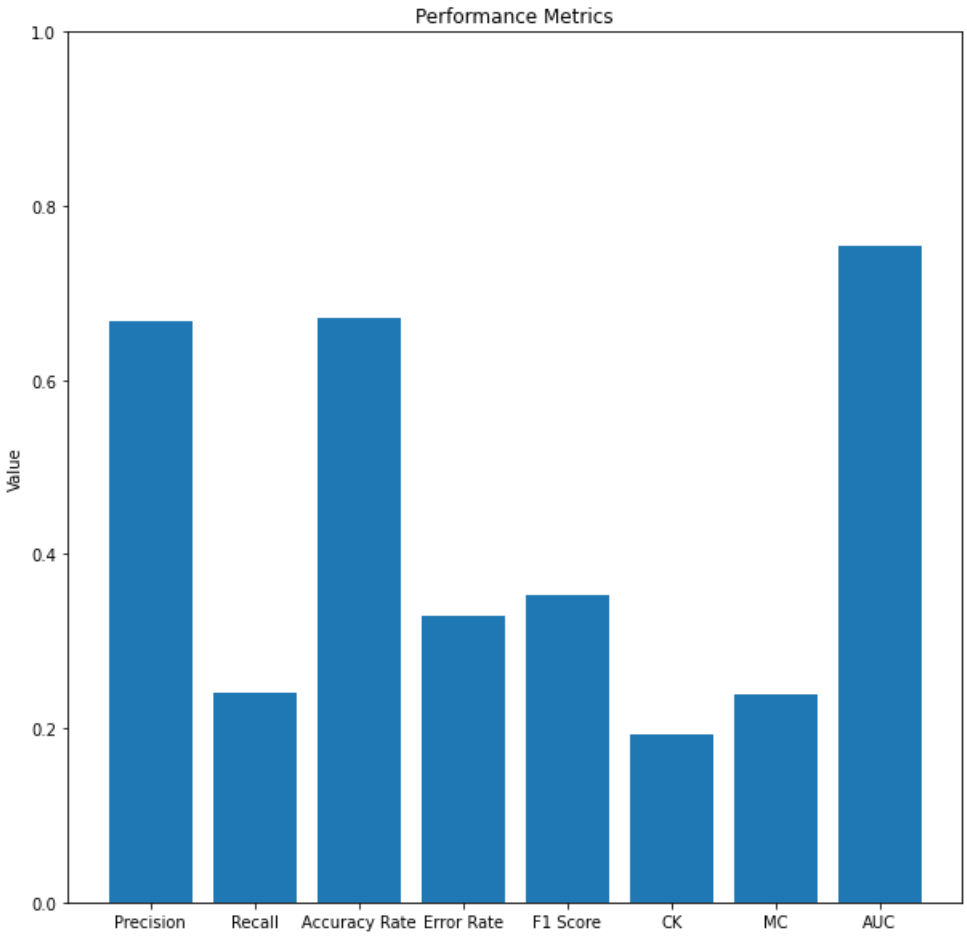
In [33]:
```
y_pred_proba2 = RF.predict_proba(X_test[selected_features])[:, 1]
fpr, tpr, _ = metrics.roc_curve(Y_test, y_pred_proba2)
auc2 = metrics.roc_auc_score(Y_test, y_pred_proba2)

plt.plot(fpr, tpr, label="Random Forest, AUC = {:.2f}".format(auc2))
plt.legend(loc="lower right")
plt.title('Courbe ROC du Random Forest', fontsize=16)
plt.xlabel('Taux de faux positifs', fontsize=12)
plt.ylabel('Taux de vrais positifs', fontsize=12)
plt.grid(True)
plt.show()
```



In [47]:
```
Accuracy_Rate2 = accuracy_score(Y_test, Y_pred2)
Error_rate2 = 1 - Accuracy_Rate2
F1_score2 = f1_score(Y_test, Y_pred2)
Precision2 = precision_score(Y_test, Y_pred2)
Recall2 = recall_score(Y_test, Y_pred2)
CK2 = cohen_kappa_score (Y_test,Y_pred2)
MC2 = matthews_corrcoef(Y_test,Y_pred2)
auc2 = metrics.roc_auc_score(Y_test, y_pred_proba2)

print("Precision : {:.2f}".format(Precision2))
print("Recall : {:.2f}".format(Recall2))
print("Accuracy rate: ", Accuracy_Rate2)
print("Error rate: ",Error_rate2)
print("F1_score: ", F1_score2)
print("CK:", CK2)
print("MC:", MC2)
print("AUC:", auc2)


# create a list of metric names and values
metric_names = ["Precision", "Recall", "Accuracy Rate", "Error Rate", "F1 Score", "CK", "MC","AUC"]
metric_values = [Precision2, Recall2, Accuracy_Rate2, Error_rate2, F1_score2, CK2, MC2,auc2]
```

```
# create a bar chart
fig, ax = plt.subplots(figsize=(10,10))
ax.bar(metric_names, metric_values)
ax.set_ylabel('Value')
ax.set_ylim([0,1])
ax.set_title('Performance Metrics')
plt.show()
```

```
Precision : 0.67
Recall : 0.24
Accuracy rate:  0.6716417910447762
Error rate:  0.32835820895522383
F1_score:  0.3529411764705882
CK: 0.19365426695842458
MC: 0.2390800650495668
AUC: 0.7538095238095238
```



3- Gradient Boosting

In [35]:
```
from sklearn.ensemble import GradientBoostingClassifier


# Créer une instance du modèle Gradient Boosting
GB = GradientBoostingClassifier(n_estimators=100000, max_depth=3, min_samples_leaf = 1 ,learning_rate = 0.1, random_state = 0)

# Créer une instance de Boruta
feat_selector = BorutaPy(GB, n_estimators='auto', verbose=2, random_state = 0)

# Adapter Boruta sur les données d'entraînement
feat_selector.fit(X_train.values, Y_train)

# Obtenir les indices des fonctionnalités sélectionnées
selected_features_indices = feat_selector.support_

# Sélectionner les fonctionnalités avec Boruta
selected_features = X_train.columns[selected_features_indices]

# Adapter le modèle Gradient Boosting sur les fonctionnalités sélectionnées
GB.fit(X_train[selected_features], Y_train)

# Afficher les variables du feature selection
print("Variables du feature selection : ")
print(selected_features)
```

```
Iteration:  1 / 100
Confirmed:  0
Tentative:  28
```

Rejected: 0
Iteration: 2 / 100
Confirmed: 0
Tentative: 28
Rejected: 0
Iteration: 3 / 100
Confirmed: 0
Tentative: 28
Rejected: 0
Iteration: 4 / 100
Confirmed: 0
Tentative: 28
Rejected: 0
Iteration: 5 / 100
Confirmed: 0
Tentative: 28
Rejected: 0
Iteration: 6 / 100
Confirmed: 0
Tentative: 28
Rejected: 0
Iteration: 7 / 100
Confirmed: 0
Tentative: 28
Rejected: 0
Iteration: 8 / 100
Confirmed: 0
Tentative: 5
Rejected: 23
Iteration: 9 / 100
Confirmed: 1
Tentative: 4
Rejected: 23
Iteration: 10 / 100
Confirmed: 1
Tentative: 4
Rejected: 23
Iteration: 11 / 100
Confirmed: 1
Tentative: 4
Rejected: 23
Iteration: 12 / 100
Confirmed: 1
Tentative: 4
Rejected: 23
Iteration: 13 / 100
Confirmed: 1
Tentative: 4
Rejected: 23
Iteration: 14 / 100
Confirmed: 1
Tentative: 4
Rejected: 23
Iteration: 15 / 100
Confirmed: 1
Tentative: 4
Rejected: 23
Iteration: 16 / 100
Confirmed: 1
Tentative: 4
Rejected: 23
Iteration: 17 / 100
Confirmed: 1
Tentative: 4
Rejected: 23
Iteration: 18 / 100
Confirmed: 1
Tentative: 4
Rejected: 23
Iteration: 19 / 100
Confirmed: 1
Tentative: 3
Rejected: 24
Iteration: 20 / 100
Confirmed: 1
Tentative: 3
Rejected: 24
Iteration: 21 / 100
Confirmed: 1
Tentative: 3
Rejected: 24
Iteration: 22 / 100
Confirmed: 1

Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 23 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 24 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 25 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 26 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 27 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 28 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 29 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 30 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 31 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 32 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 33 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 34 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 35 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 36 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 37 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 38 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 39 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 40 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 41 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 42 / 100
Confirmed: 1
Tentative: 2
Rejected: 25

Iteration: 43 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 44 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 45 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 46 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 47 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 48 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 49 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 50 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 51 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 52 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 53 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 54 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 55 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 56 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 57 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 58 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 59 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 60 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 61 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 62 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 63 / 100
Confirmed: 1
Tentative: 2

Tentative: 2
Rejected: 25
Iteration: 64 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 65 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 66 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 67 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 68 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 69 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 70 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 71 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 72 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 73 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 74 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 75 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 76 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 77 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 78 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 79 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 80 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 81 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 82 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 83 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 84 / 100

Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 85 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 86 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 87 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 88 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 89 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 90 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 91 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 92 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 93 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 94 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 95 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 96 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 97 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 98 / 100
Confirmed: 1
Tentative: 2
Rejected: 25
Iteration: 99 / 100
Confirmed: 1
Tentative: 2
Rejected: 25

BorutaPy finished running.

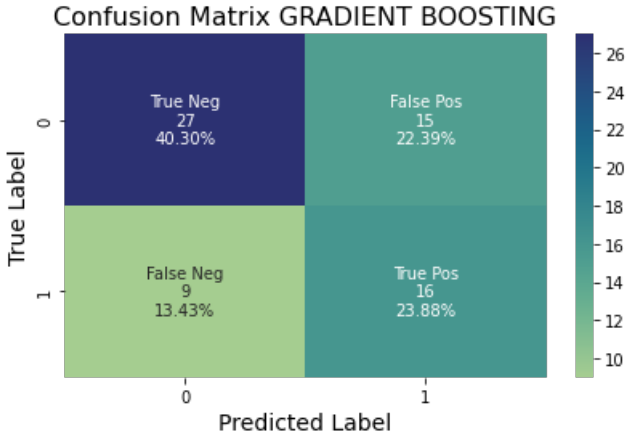
Iteration: 100 / 100
Confirmed: 1
Tentative: 1
Rejected: 25
Variables du feature selection :
Index(['item25'], dtype='object')
In [36]:
```
# Prédire les classes pour les données de test
Y_pred3 = GB.predict(X_test[selected_features])

# Afficher les prédictions
print(Y_pred3)
```

```
[1. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1.
 0. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0.
 0. 0. 1. 1. 0. 1. 0. 1. 1. 0. 1. 0. 1. 0. 1. 1. 1. 1. 1.]
```

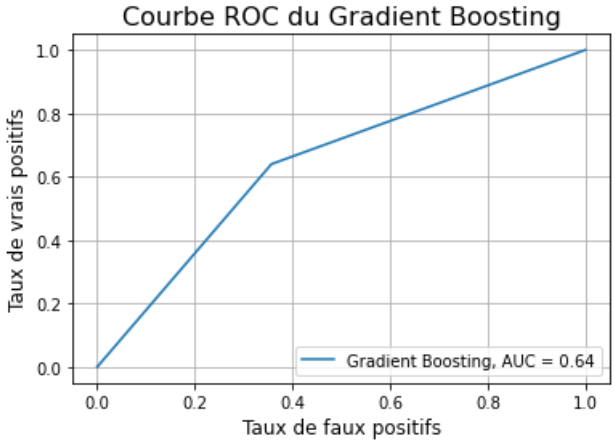
In [37]:
```
CM3 = confusion_matrix(Y_test, Y_pred3)
CM3
```

Out[37]:
```
array([[27, 15],
       [ 9, 16]], dtype=int64)
```

In [38]:
```
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ['{0:0.0f}'.format(value) for value in
                CM3.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
                     CM3.flatten()/np.sum(CM3)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(CM3, annot=labels, fmt='', cmap='crest')
plt.xlabel('Predicted Label', fontsize=14)
plt.ylabel('True Label', fontsize=14)
plt.title('Confusion Matrix GRADIENT BOOSTING', fontsize=16)
plt.tight_layout()
plt.show()
```



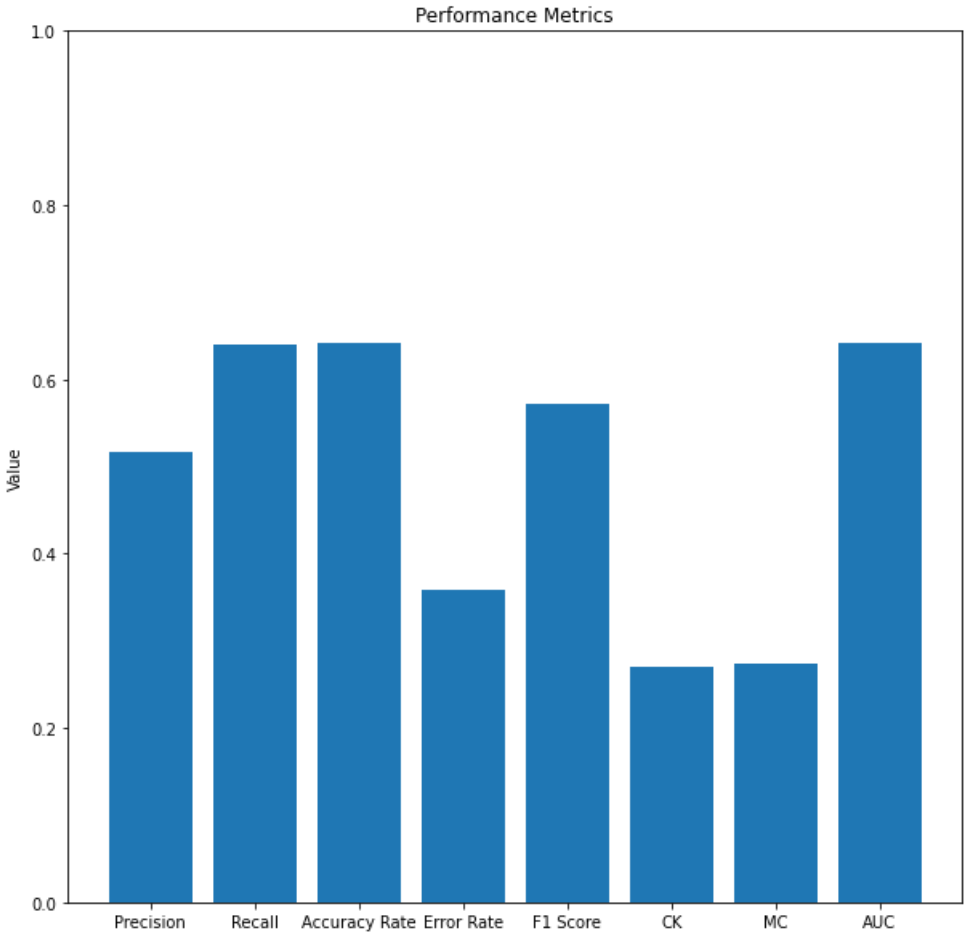
In [39]:
```
y_pred_proba3 = GB.predict_proba(X_test[selected_features])[:, 1]
fpr, tpr, _ = metrics.roc_curve(Y_test, y_pred_proba3)
auc3 = metrics.roc_auc_score(Y_test, y_pred_proba3)

plt.plot(fpr, tpr, label="Gradient Boosting, AUC = {:.2f}".format(auc3))
plt.legend(loc="lower right")
plt.title('Courbe ROC du Gradient Boosting', fontsize=16)
plt.xlabel('Taux de faux positifs', fontsize=12)
plt.ylabel('Taux de vrais positifs', fontsize=12)
plt.grid(True)
plt.show()
```



In [48]:
```
Accuracy_Rate3 = accuracy_score(Y_test, Y_pred3)
Error_rate3 = 1 - Accuracy_Rate3
F1_score3 = f1_score(Y_test, Y_pred3)
Precision3 = precision_score(Y_test, Y_pred3)
Recall3 = recall_score(Y_test, Y_pred3)
CK3 = cohen_kappa_score (Y_test,Y_pred3)
MC3 = matthews_corrcoef(Y_test,Y_pred3)
auc3 = metrics.roc_auc_score(Y_test, y_pred_proba3)

print("Precision : {:.2f}".format(Precision3))
print("Recall : {:.2f}".format(Recall3))
print("Accuracy rate: ", Accuracy_Rate3)
print("Error rate: ",Error_rate3)
print("F1_score: ", F1_score_logreg3)
print("CK:", CK3)
print("MC:", MC3)
print("AUC:", auc3)
```

```
# create a list of metric names and values
metric_names = ["Precision", "Recall", "Accuracy Rate", "Error Rate", "F1 Score", "CK", "MC","AUC"]
metric_values = [Precision3, Recall3, Accuracy_Rate3, Error_rate3, F1_score3, CK3, MC3,auc3]

# create a bar chart
fig, ax = plt.subplots(figsize=(10,10))
ax.bar(metric_names, metric_values)
ax.set_ylabel('Value')
ax.set_ylim([0,1])
ax.set_title('Performance Metrics')
plt.show()
```

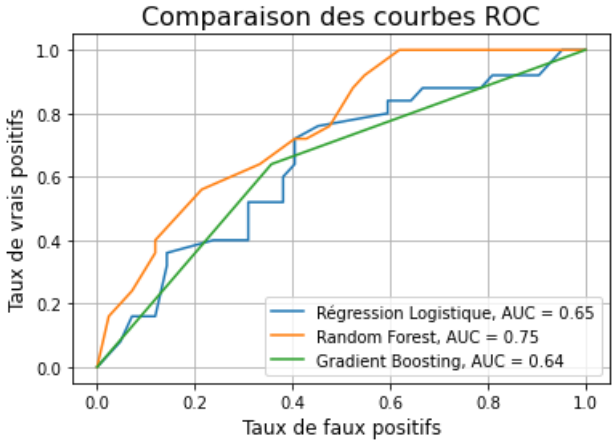
```
Precision : 0.52
Recall : 0.64
Accuracy rate:  0.6417910447761194
Error rate:  0.35820895522388063
F1_score:  0.5714285714285714
CK: 0.26975476839237056
MC: 0.27436562775947815
AUC: 0.6414285714285715
```



Comparaison des courbes roc

In [41]:
```
# Calculer les taux de faux positifs et les taux de vrais positifs pour chaque modèle
fpr_rl, tpr_rl, _ = metrics.roc_curve(Y_test, y_pred_proba1)
fpr_rf, tpr_rf, _ = metrics.roc_curve(Y_test, y_pred_proba2)
fpr_gb, tpr_gb, _ = metrics.roc_curve(Y_test, y_pred_proba3)

# Calculer les aires sous la courbe (AUC) pour chaque modèle
auc_rl = metrics.roc_auc_score(Y_test, y_pred_proba1)
auc_rf = metrics.roc_auc_score(Y_test, y_pred_proba2)
auc_gb = metrics.roc_auc_score(Y_test, y_pred_proba3)

# Tracer les courbes ROC pour chaque modèle
plt.plot(fpr_rl, tpr_rl, label="Régression Logistique, AUC = {:.2f}".format(auc_rl))
plt.plot(fpr_rf, tpr_rf, label="Random Forest, AUC = {:.2f}".format(auc_rf))
plt.plot(fpr_gb, tpr_gb, label="Gradient Boosting, AUC = {:.2f}".format(auc_gb))

# Afficher la légende et les titres
plt.legend(loc="lower right")
plt.title('Comparaison des courbes ROC', fontsize=16)
plt.xlabel('Taux de faux positifs', fontsize=12)
plt.ylabel('Taux de vrais positifs', fontsize=12)
plt.grid(True)
```

```
# Afficher le graphique
plt.show()
```



In [49]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, cohen_kappa_score, matthews_corrcoef, roc_auc_score

# Afficher les métriques
print("Métriques pour Régression Logistique :")
print("Accuracy rate: {:.2f}".format(Accuracy_Rate1))
print("Error rate: {:.2f}".format(Error_rate1))
print("Precision: {:.2f}".format(Precision1))
print("Recall: {:.2f}".format(Recall1))
print("F1 Score: {:.2f}".format(F1_score1))
print("CK: {:.2f}".format(CK1))
print("MC: {:.2f}".format(MC1))
print("AUC: {:.2f}".format(auc1))
print()

print("Métriques pour Random Forest :")
print("Accuracy rate: {:.2f}".format(Accuracy_Rate2))
print("Error rate: {:.2f}".format(Error_rate2))
print("Precision: {:.2f}".format(Precision2))
print("Recall: {:.2f}".format(Recall2))
print("F1 Score: {:.2f}".format(F1_score2))
print("CK: {:.2f}".format(CK2))
print("MC: {:.2f}".format(MC2))
print("AUC: {:.2f}".format(auc2))
print()

print("Métriques pour Gradient Boosting :")
print("Accuracy rate: {:.2f}".format(Accuracy_Rate3))
print("Error rate: {:.2f}".format(Error_rate3))
print("Precision: {:.2f}".format(Precision3))
print("Recall: {:.2f}".format(Recall3))
print("F1 Score: {:.2f}".format(F1_score3))
print("CK: {:.2f}".format(CK3))
print("MC: {:.2f}".format(MC3))
print("AUC: {:.2f}".format(auc3))
print()
```

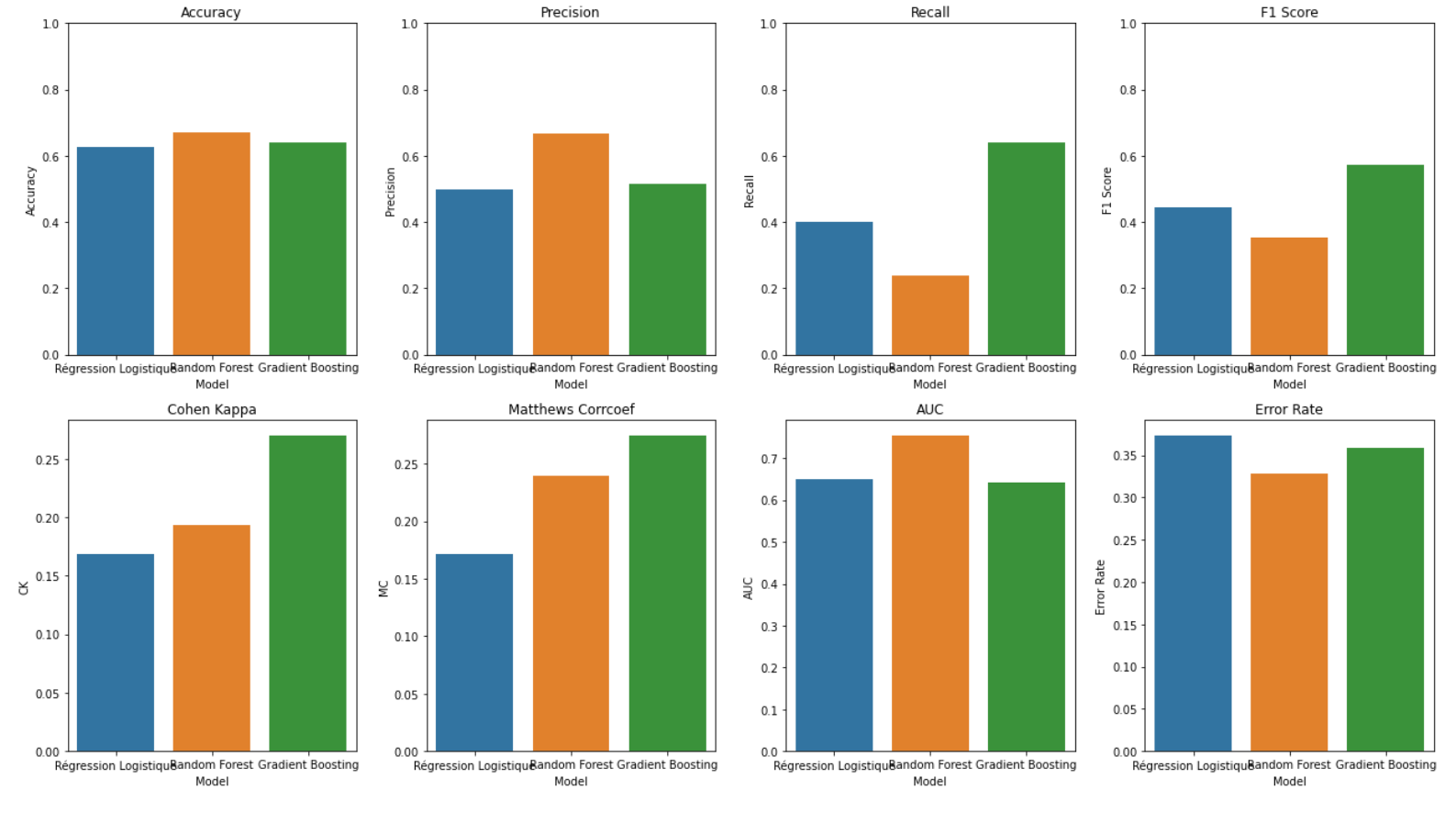
Métriques pour Régression Logistique :
Accuracy rate: 0.63
Error rate: 0.37
Precision: 0.50
Recall: 0.40
F1 Score: 0.44
CK: 0.17
MC: 0.17
AUC: 0.65

Métriques pour Random Forest :
Accuracy rate: 0.67
Error rate: 0.33
Precision: 0.67
Recall: 0.24
F1 Score: 0.35
CK: 0.19
MC: 0.24
AUC: 0.75

Métriques pour Gradient Boosting :
Accuracy rate: 0.64
Error rate: 0.36
Precision: 0.52
Recall: 0.64
F1 Score: 0.57
CK: 0.27
MC: 0.27
AUC: 0.64

In [50]:
```
# Calculer les métriques pour chaque modèle

Accuracy_Rate3 = accuracy_score(Y_test, Y_pred3)
Error_rate3 = 1 - Accuracy_Rate3
F1_score3 = f1_score(Y_test, Y_pred3)
Precision3 = precision_score(Y_test, Y_pred3)
Recall3 = recall_score(Y_test, Y_pred3)
CK3 = cohen_kappa_score (Y_test,Y_pred3)
MC3 = matthews_corrcoef(Y_test,Y_pred3)
auc3 = metrics.roc_auc_score(Y_test, y_pred_proba3)



metrics_rl = {
    'Model': 'Régression Logistique',
    'Accuracy': Accuracy_Rate1,
    'Precision': Precision1,
    'Recall':Recall1,
    'F1 Score':F1_score1,
    'CK':CK1,
    'MC': MC1,
    'AUC':auc1,
    'Error Rate':Error_rate1
}

metrics_rf = {
    'Model': 'Random Forest',
   'Accuracy': Accuracy_Rate2,
    'Precision': Precision2,
    'Recall':Recall2,
    'F1 Score':F1_score2,
    'CK':CK2,
    'MC': MC2,
    'AUC':auc2,
    'Error Rate':Error_rate2
}

metrics_gb = {
    'Model': 'Gradient Boosting',
    'Accuracy': Accuracy_Rate3,
    'Precision': Precision3,
    'Recall':Recall3,
    'F1 Score':F1_score3,
    'CK':CK3,
    'MC': MC3,
    'AUC':auc3,
    'Error Rate':Error_rate3
}

# Créer un dataframe avec les métriques
metrics_df = pd.DataFrame([metrics_rl, metrics_rf, metrics_gb])
```

```
# Ploter les métriques
plt.figure(figsize=(18, 10))

# Accuracy
plt.subplot(2, 4, 1)
sns.barplot(x='Model', y='Accuracy', data=metrics_df)
plt.title('Accuracy')
plt.ylim(0, 1)

# Precision
plt.subplot(2, 4, 2)
sns.barplot(x='Model', y='Precision', data=metrics_df)
plt.title('Precision')
plt.ylim(0, 1)

# Recall
plt.subplot(2, 4, 3)
sns.barplot(x='Model', y='Recall', data=metrics_df)
plt.title('Recall')
plt.ylim(0, 1)

# F1 Score
plt.subplot(2, 4, 4)
sns.barplot(x='Model', y='F1 Score', data=metrics_df)
plt.title('F1 Score')
plt.ylim(0, 1)

# Cohen Kappa
plt.subplot(2, 4, 5)
sns.barplot(x='Model', y='CK', data=metrics_df)
plt.title('Cohen Kappa')

# Matthews Corrcoef
plt.subplot(2, 4, 6)
sns.barplot(x='Model', y='MC', data=metrics_df)
plt.title('Matthews Corrcoef')

# AUC
plt.subplot(2, 4, 7)
sns.barplot(x='Model', y='AUC', data=metrics_df)
plt.title('AUC')

# Error Rate
plt.subplot(2, 4, 8)
sns.barplot(x='Model', y='Error Rate', data=metrics_df)
plt.title('Error Rate')

plt.tight_layout()
plt.show()
```



A Maitriser

Accuracy (Taux de précision) :

Formule mathématique : Accuracy = (TP + TN) / (TP + TN + FP + FN) L'Accuracy mesure la proportion de prédictions correctes parmi toutes les prédictions effectuées. Elle donne une idée globale de la performance du modèle en termes de prédictions correctes. La valeur de référence est généralement la proportion d'exemples de la classe majoritaire dans l'ensemble de données. Par exemple, si 70% des exemples appartiennent à la classe A et 30% à la classe B, alors l'accuracy de prédire toujours la classe majoritaire serait de 0,70.

Precision (Précision) :

Formule mathématique : Precision = TP / (TP + FP) La Precision mesure la proportion de prédictions positives correctes parmi toutes les prédictions positives effectuées. Elle met l'accent sur la capacité du modèle à éviter les faux positifs. La valeur de référence est souvent la précision associée à un classifieur qui prédit toujours la classe majoritaire. Dans l'exemple précédent, la précision pour prédire la classe majoritaire serait de 0,70.

Recall (Rappel) :

Formule mathématique : Recall = TP / (TP + FN)

Le Recall appelé Sensibilité ou Taux de Vrais Positifs mesure la proportion de vrais positifs correctement identifiés parmi tous les vrais positifs réels. Il met l'accent sur la capacité du modèle à éviter les faux négatifs.La valeur de référence est généralement le rappel associé à un classifieur qui prédit toujours la classe minoritaire. Dans l'exemple précédent, le rappel pour prédire la classe minoritaire serait de 1,00.

F1 Score :

Formule mathématique : F1 Score = 2 *(Precision* Recall) / (Precision + Recall) Le F1 Score est une métrique qui combine la Precision et le Recall en une seule valeur. Il donne une mesure de l'équilibre entre la Precision et le Recall. La valeur de référence est souvent le F1 Score associé à un classifieur qui prédit toujours la classe majoritaire. Dans l'exemple précédent, le F1 Score pour prédire la classe majoritaire serait de 0,82.

Cohen's Kappa (Kappa de Cohen) :

Formule mathématique : Kappa = (observed_accuracy - expected_accuracy) / (1 - expected_accuracy) Le Kappa de Cohen est une mesure de la concordance entre les prédictions du modèle et les observations réelles, corrigée pour tenir compte de la concordance due au hasard. Il tient compte de la possibilité d'accord simplement dû au hasard. Il n'y a pas de valeur de référence spécifique pour le kappa de Cohen. Cependant, une interprétation courante est que 0 représente un accord au hasard et 1 représente un accord parfait

Matthews Correlation Coefficient (Coefficient de corrélation de Matthews) :

Formule mathématique : MCC = (TP *TN - FP* FN) / sqrt((TP + FP) *(TP + FN)* (TN + FP) * (TN + FN)) Le MCC est une mesure de la qualité globale des prédictions du modèle, prenant en compte les quatre éléments de la matrice de confusion. Il est particulièrement utile lorsque les classes sont déséquilibrées. Il n'y a pas de valeur de référence spécifique pour le coefficient de corrélation de Matthews. Cependant, une interprétation courante est que -1 représente une prédiction complètement incorrecte, 0 représente une prédiction au hasard et 1 représente une prédiction parfaite.

AUC (Area Under the ROC Curve) :

L'AUC mesure la capacité du modèle à distinguer entre les classes positives et négatives. Il représente la probabilité qu'un exemple positif choisi au hasard par le modèle soit classé avec un score plus élevé que celui d'un exemple négatif choisi au hasard. Une valeur d'AUC proche de 1 indique une bonne capacité de discrimination du modèle. La valeur de référence pour l'AUC est généralement 0,5 qui correspond à une prédiction aléatoire. Un modèle avec un AUC supérieur à 0,5 est considéré comme meilleur que le hasard, tandis qu'un AUC de 1 indique une prédiction parfaite.

Error Rate (Taux d'erreur) :

Formule mathématique : Error Rate = 1 - Accuracy Le taux d'erreur mesure la proportion d'erreurs de classification commises par le modèle parmi toutes les prédictions effectuées. Il est complémentaire à l'Accuracy, fournissant une autre perspective sur la performance du modèle. Un Error Rate de zéro (0) signifierait qu'aucune erreur de prédiction n'a été commise par le modèle, ce qui indiquerait une performance parfaite. Cependant, dans la pratique, il est rare d'atteindre un Error Rate de zéro, car les modèles de classification ne sont pas parfaits et peuvent toujours commettre des erreurs. L'objectif est de réduire autant que possible l'Error Rate et de s'approcher le plus possible de zéro. Une valeur plus proche de zéro indique une meilleure performance du modèle, car cela signifie qu'il commet moins d'erreurs de prédiction.

AUC (Area Under the ROC Curve) :

La formule mathématique de l'AUC (Area Under the ROC Curve) est basée sur la construction de la courbe ROC (Receiver Operating Characteristic). La courbe ROC représente la relation entre le taux de vrais positifs (Sensibilité) et le taux de faux positifs (1 - Spécificité) pour différents seuils de classification.

Pour calculer l'AUC, on utilise la règle trapezoïdale pour approximer l'aire sous la courbe ROC. La formule mathématique est la suivante :

AUC = ∑ [(FP[i+1] - FP[i]) * (TP[i+1] + TP[i]) / 2]

où FP[i] représente le taux de faux positifs à l'indice i (i.e., 1 - spécificité) et TP[i] représente le taux de vrais positifs à l'indice i (i.e., sensibilité).

La formule calcule la somme des aires des trapèzes formés par chaque paire de points consécutifs de la courbe ROC. Chaque trapèze a une base égale à la différence entre les taux de faux positifs, et une hauteur égale à la moyenne des taux de vrais positifs.

L'AUC est une valeur comprise entre 0 et 1. Une valeur de 0,5 indique un modèle qui prédit au hasard, tandis qu'une valeur de 1 indique un modèle parfait qui classe correctement tous les exemples.

L'AUC est couramment utilisée pour évaluer la performance des modèles de classification, en particulier dans les problèmes où les classes sont déséquilibrées ou lorsque le seuil de classification optimal est inconnu. Une valeur d'AUC plus élevée indique une meilleure capacité de discrimination du modèle entre les classes positives et négatives.