Modèle Autisme :

- Ce notebook rassemble 3 datasets ; Après les avoir fusionnés, nous avons créé 3 dataframes et appliqué des méthodes de preprocessing sur chaque nouvel ensemble de données apres on a fusionné ces 3 dataframes.
- QChat-10 : A1...A10 : Ces questions évaluent des comportements et des traits spécifiques chez les enfants susceptibles d'être associés à l'ASD.

Import des biblio :

In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import classification_report, accuracy_score, f1_score, pre
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
import warnings
def ignore_warn(*args, **kwargs):
    pass
warnings.warn = ignore_warn

import os
for dirname, _, filenames in os.walk('C:/Users/yosser/OneDrive/Bureau/3DNI/Proje
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
C:/Users/yosser/OneDrive/Bureau/3DNI/Projets/IA/Dataset\autism_screening.csv
C:/Users/yosser/OneDrive/Bureau/3DNI/Projets/IA/Dataset\data_csv.csv
C:/Users/yosser/OneDrive/Bureau/3DNI/Projets/IA/Dataset\Toddler Autism dataset Ju
ly 2018.csv
```

Reading Datasets :

In [8]:
```
data1=pd.read_csv('C:/Users/yosser/OneDrive/Bureau/3DNI/Projets/IA/Dataset/data_
data2=pd.read_csv('C:/Users/yosser/OneDrive/Bureau/3DNI/Projets/IA/Dataset/Toddl
data3=pd.read_csv('C:/Users/yosser/OneDrive/Bureau/3DNI/Projets/IA/Dataset/autis
```

Affichage des premières lignes des 3 DataFrames :

In [11]:
```
data1.head()
```

Out[11]:

	CASE_NO_PATIENT'S	A1	A2	A3	A4	A5	A6	A7	A8	A9	...	Global developmental delay/intellectual disability
0	1	0	0	0	0	0	0	1	1	0	...	Yes
1	2	1	1	0	0	0	1	1	0	0	...	Yes
2	3	1	0	0	0	0	0	1	1	0	...	Yes
3	4	1	1	1	1	1	1	1	1	1	...	Yes
4	5	1	1	0	1	1	1	1	1	1	...	Yes

5 rows × 28 columns

In [13]:
```
data2.head()
```

Out[13]:

	Case_No	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	Age_Mons	Qchat-10-Score	Sex	Etl
0	1	0	0	0	0	0	0	1	1	0	1	28	3	f	e
1	2	1	1	0	0	0	1	1	0	0	0	36	4	m	Eu
2	3	1	0	0	0	0	0	1	1	0	1	36	4	m	e
3	4	1	1	1	1	1	1	1	1	1	1	24	10	m	H
4	5	1	1	0	1	1	1	1	1	1	1	20	9	f	Eu

In [15]:
```
data3.head()
```

Out[15]:

	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score
0	1	1	1	1	0	0	1	1
1	1	1	0	1	0	0	0	1
2	1	1	0	1	1	0	1	1
3	1	1	0	1	0	0	1	1
4	1	0	0	0	0	0	0	1

5 rows × 21 columns

Affichage des noms des colonnes de chaque DataFrame :

In [18]:

```
print(data1.columns)
print(data2.columns)
print(data3.columns)
```

```
Index(['CASE_NO_PATIENT'S', 'A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8',
       'A9', 'A10_Autism_Spectrum_Quotient', 'Social_Responsiveness_Scale',
       'Age_Years', 'Qchat_10_Score', 'Speech Delay/Language Disorder',
       'Learning disorder', 'Genetic_Disorders', 'Depression',
       'Global developmental delay/intellectual disability',
       'Social/Behavioural Issues', 'Childhood Autism Rating Scale',
       'Anxiety_disorder', 'Sex', 'Ethnicity', 'Jaundice',
       'Family_mem_with_ASD', 'Who_completed_the_test', 'ASD_traits'],
      dtype='object')
Index(['Case_No', 'A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9', 'A10',
       'Age_Mons', 'Qchat-10-Score', 'Sex', 'Ethnicity', 'Jaundice',
       'Family_mem_with_ASD', 'Who completed the test', 'Class/ASD Traits '],
      dtype='object')
Index(['A1_Score', 'A2_Score', 'A3_Score', 'A4_Score', 'A5_Score', 'A6_Score',
       'A7_Score', 'A8_Score', 'A9_Score', 'A10_Score', 'age', 'gender',
       'ethnicity', 'jundice', 'austim', 'contry_of_res', 'used_app_before',
       'result', 'age_desc', 'relation', 'Class/ASD'],
      dtype='object')
```

Creating dataframes with simular columns and features :

In [21]:

```
df1=pd.concat([data1.iloc[:,1:11],data1.iloc[:,[12,22,23,24,25,26,27]]],axis=1)
df1.head()
```

Out[21]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10_Autism_Spectrum_Quotient	Age_Years
0	0	0	0	0	0	0	1	1	0	1	2
1	1	1	0	0	0	1	1	0	0	0	3
2	1	0	0	0	0	0	1	1	0	1	3
3	1	1	1	1	1	1	1	1	1	1	2
4	1	1	0	1	1	1	1	1	1	1	2

In [23]:

```
df2=pd.concat([data2.iloc[:,1:12],data2.iloc[:,13:]],axis=1)
df2['Age_Mons']=(df2['Age_Mons']/12).astype(int)   #changement de month vers yea
df2.head()
```

Out[23]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	Age_Mons	Sex	Ethnicity	Jaundice
0	0	0	0	0	0	0	1	1	0	1	2	f	middle eastern	yes
1	1	1	0	0	0	1	1	0	0	0	3	m	White European	yes
2	1	0	0	0	0	0	1	1	0	1	3	m	middle eastern	yes
3	1	1	1	1	1	1	1	1	1	1	2	m	Hispanic	no
4	1	1	0	1	1	1	1	1	1	1	1	f	White European	no

In [25]:

```
df3=pd.concat([data3.iloc[:,0:15],data3.iloc[:,-2:]],axis=1)
df3.head()
```

Out[25]:

	A1_Score	A2_Score	A3_Score	A4_Score	A5_Score	A6_Score	A7_Score	A8_Score
0	1	1	1	1	0	0	1	1
1	1	1	0	1	0	0	0	1
2	1	1	0	1	1	0	1	1
3	1	1	0	1	0	0	1	1
4	1	0	0	0	0	0	0	1

In [27]:
```
order_test= pd.DataFrame({
    'df1': df1.columns,
    'df2': df2.columns ,
    'df3': df3.columns
})
order_test
```

Out[27]:

	df1	df2	df3
0	A1	A1	A1_Score
1	A2	A2	A2_Score
2	A3	A3	A3_Score
3	A4	A4	A4_Score
4	A5	A5	A5_Score
5	A6	A6	A6_Score
6	A7	A7	A7_Score
7	A8	A8	A8_Score
8	A9	A9	A9_Score
9	A10_Autism_Spectrum_Quotient	A10	A10_Score
10	Age_Years	Age_Mons	age
11	Sex	Sex	gender
12	Ethnicity	Ethnicity	ethnicity
13	Jaundice	Jaundice	jundice
14	Family_mem_with_ASD	Family_mem_with_ASD	austim
15	Who_completed_the_test	Who completed the test	relation
16	ASD_traits	Class/ASD Traits	Class/ASD

Maintenant , on va faire la concatenation vertical des 3 dataframes :

In [30]:
```
# Rename columns to have the same names in all DataFrames
df2.columns = df3.columns = df1.columns

# Concatenate the DataFrames
data_fin = pd.concat([df3, df2, df1], axis=0)
data_fin.head()
```

Out[30]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10_Autism_Spectrum_Quotient	Age_Years
0	1	1	1	1	0	0	1	1	0	0	26.0
1	1	1	0	1	0	0	0	1	0	1	24.0
2	1	1	0	1	1	0	1	1	1	1	27.0
3	1	1	0	1	0	0	1	1	0	1	35.0
4	1	0	0	0	0	0	0	1	0	0	40.0

In [32]:

```
data_fin.shape
```

Out[32]: (3743, 17)

Remarque : Ce code identifie et résume les colonnes de type objet dans le DataFrame data_fin, en affichant leurs noms, les valeurs uniques qu'elles contiennent, et le nombre de valeurs uniques. Cela aide à comprendre les données catégorielles avant le nettoyage et l'analyse.

In [35]:

```
# Get object type columns
object_cols = data_fin.select_dtypes('O').columns

# Create new DataFrame
object_df = pd.DataFrame({
    'Objects': object_cols,
    'Unique values': [data_fin[col].unique() for col in object_cols],
    'number of unique values':[data_fin[col].nunique()for col in object_cols]
})

object_df
```

Out[35]:

	Objects	Unique values	number of unique values
0	Sex	[f, m, F, M]	4
1	Ethnicity	[White-European, Latino, ?, Others, Black, Asi...	23
2	Jaundice	[no, yes, Yes, No]	4
3	Family_mem_with_ASD	[no, yes, No, Yes]	4
4	Who_completed_the_test	[Self, Parent, ?, Health care professional, Re...	11
5	ASD_traits	[NO, YES, No, Yes]	4

Standardisation des valeur de chaques colonnes :

In [38]:

```
replacements = {
    'f': 'F',
    'm': 'M',
```

```
}
data_fin['Sex'] = data_fin['Sex'].replace(replacements)
```

In [40]:
```
replacements = {
    'yes': 'Yes',
    'no': 'No',
}
data_fin['Jaundice'] = data_fin['Jaundice'].replace(replacements)
```

In [42]:
```
replacements = {
    'yes': 'Yes',
    'no': 'No',
}
data_fin['Family_mem_with_ASD'] = data_fin['Family_mem_with_ASD'].replace(replac
```

In [44]:
```
replacements = {
    'YES': 'Yes',
    'NO': 'No',
}
data_fin['ASD_traits'] = data_fin['ASD_traits'].replace(replacements)
```

In [46]:
```
replacements = {
    'middle eastern': 'Middle Eastern',
    'Middle Eastern ': 'Middle Eastern',
    'mixed': 'Mixed',
    'asian': 'Asian',
    'black': 'Black',
    'south asian': 'South Asian',
    'PaciFica':'Pacifica',
    'Pasifika':'Pacifica'

}
data_fin['Ethnicity'] = data_fin['Ethnicity'].replace(replacements)
```

In [48]:
```
replacements = {
    'Health care professional':'Health Care Professional',
    'family member':'Family Member',
    'Family member':'Family Member'
}
data_fin['Who_completed_the_test'] = data_fin['Who_completed_the_test'].replace(
```

In [50]:
```
# Get object type columns
object_cols = data_fin.select_dtypes('O').columns

# Create new DataFrame
object_df = pd.DataFrame({
    'Objects': object_cols,
    'Unique values': [data_fin[col].unique() for col in object_cols],
    'number of unique values':[data_fin[col].nunique()for col in object_cols]
})

object_df
```

Out[50]:

	Objects	Unique values	number of unique values
0	Sex	[F, M]	2
1	Ethnicity	[White-European, Latino, ?, Others, Black, Asi...	15
2	Jaundice	[No, Yes]	2
3	Family_mem_with_ASD	[No, Yes]	2
4	Who_completed_the_test	[Self, Parent, ?, Health Care Professional, Re...	8
5	ASD_traits	[No, Yes]	2

Remplacement des valeurs manquantes dans les colonnes par la valeur la plus fréquente :

In [53]:
```
count_question_marks = data_fin['Ethnicity'].str.contains(r'\?').sum()
print(f"Number of entries with '?': {count_question_marks}")
# Replace '?' with NaN in the 'Ethnicity' column
data_fin['Ethnicity'].replace('?', np.nan, inplace=True)
```

```
Number of entries with '?': 95
```

In [55]:
```
# Find the mode of the 'Ethnicity' column
mode_ethnicity = data_fin['Ethnicity'].mode()[0]
data_fin['Ethnicity'].fillna(mode_ethnicity, inplace=True)
```

In [57]:
```
count_question_marks_test = data_fin['Who_completed_the_test'].str.contains(r'\?
print(f"Number of entries with '?': {count_question_marks_test}")
data_fin['Who_completed_the_test'].replace('?', np.nan, inplace=True)
```

```
Number of entries with '?': 95
```

In [59]:
```
# Find the mode of the 'Who_completed_the_test' column
mode = data_fin['Who_completed_the_test'].mode()[0]

data_fin['Who_completed_the_test'].fillna(mode, inplace=True)
```

In [61]:
```
# Find the mode of the 'Age_Years' column
mode = data_fin['Age_Years'].mode()[0]

data_fin['Age_Years'].fillna(mode, inplace=True)
```

In [63]:
```
data_fin
```

Out[63]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10_Autism_Spectrum_Quotient	Age_Yea
0	1	1	1	1	0	0	1	1	0	0	26
1	1	1	0	1	0	0	0	1	0	1	24
2	1	1	0	1	1	0	1	1	1	1	27
3	1	1	0	1	0	0	1	1	0	1	35
4	1	0	0	0	0	0	0	1	0	0	40
...	...	...	...	...	...	...	...	...	...	...	
1980	0	0	0	0	0	0	0	0	0	1	7
1981	0	0	0	0	0	0	0	0	0	0	14
1982	0	0	0	0	0	0	0	0	0	1	7
1983	0	0	0	0	0	0	0	0	0	0	14
1984	0	0	0	0	0	0	0	0	0	1	7

3743 rows × 17 columns

Affichage du nombre de valeurs manquantes pour chaque colonne de data_fin :

In [66]:
```
count_question_marks = data_fin['Ethnicity'].str.contains(r'\?').sum()
print(f"Number of entries with '?': {count_question_marks}")
```

```
Number of entries with '?': 0
```

In [68]:
```
count_question_marks_test = data_fin['Who_completed_the_test'].str.contains(r'\?
print(f"Number of entries with '?': {count_question_marks_test}")
```

```
Number of entries with '?': 0
```

In [70]:
```
# Compter tous les '?' dans toutes les colonnes
count_question_marks_total = data_fin.applymap(lambda x: x == '?').sum().sum()

print(f"Total number of entries with '?': {count_question_marks_total}")
```

```
Total number of entries with '?': 0
```

In [72]:
```
print(data_fin.isnull().sum())
```

```
A1                              0
A2                              0
A3                              0
A4                              0
A5                              0
A6                              0
A7                              0
A8                              0
A9                              0
A10_Autism_Spectrum_Quotient    0
Age_Years                       0
Sex                             0
Ethnicity                       0
Jaundice                        0
Family_mem_with_ASD             0
Who_completed_the_test          0
ASD_traits                      0
dtype: int64
```

In [74]:
```python
df_missing = pd.DataFrame(data_fin.isnull().sum(), columns=["Missing Values"])
df_missing.style.bar(color="#84A9AC", vmin=0, vmax=1)
```

Out[74]:

	Missing Values
A1	0
A2	0
A3	0
A4	0
A5	0
A6	0
A7	0
A8	0
A9	0
A10_Autism_Spectrum_Quotient	0
Age_Years	0
Sex	0
Ethnicity	0
Jaundice	0
Family_mem_with_ASD	0
Who_completed_the_test	0
ASD_traits	0

In [76]:
```python
data_fin.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3743 entries, 0 to 1984
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   A1                              3743 non-null   int64  
 1   A2                              3743 non-null   int64  
 2   A3                              3743 non-null   int64  
 3   A4                              3743 non-null   int64  
 4   A5                              3743 non-null   int64  
 5   A6                              3743 non-null   int64  
 6   A7                              3743 non-null   int64  
 7   A8                              3743 non-null   int64  
 8   A9                              3743 non-null   int64  
 9   A10_Autism_Spectrum_Quotient    3743 non-null   int64  
 10  Age_Years                       3743 non-null   float64
 11  Sex                             3743 non-null   object 
 12  Ethnicity                       3743 non-null   object 
 13  Jaundice                        3743 non-null   object 
 14  Family_mem_with_ASD             3743 non-null   object 
 15  Who_completed_the_test          3743 non-null   object 
 16  ASD_traits                      3743 non-null   object 
dtypes: float64(1), int64(10), object(6)
memory usage: 526.4+ KB
```

Conversion de la colonne 'Age_Years' de float à integer

In [79]:
```python
# Conversion de la colonne 'Age_Years' de float à integer
data_fin['Age_Years'] = data_fin['Age_Years'].astype(int)
```

Transformation des variables catégorielles en valeurs numériques:

In [82]:
```python
data_fin.head()
```

Out[82]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10_Autism_Spectrum_Quotient	Age_Years
0	1	1	1	1	0	0	1	1	0	0	26
1	1	1	0	1	0	0	0	1	0	1	24
2	1	1	0	1	1	0	1	1	1	1	27
3	1	1	0	1	0	0	1	1	0	1	35
4	1	0	0	0	0	0	0	1	0	0	40

Mapping des valeurs de chaque colonnes :

In [85]:
```python
# Créer des instances de LabelEncoder pour chaque colonne
le_sex = LabelEncoder()
le_jaundice = LabelEncoder()
le_family = LabelEncoder()
le_who = LabelEncoder()
```

```
le_asd_traits = LabelEncoder()
le_ethnicity = LabelEncoder()

# Encoder les colonnes avec leurs encoders respectifs
data_fin["Sex"] = le_sex.fit_transform(data_fin["Sex"])
data_fin["Jaundice"] = le_jaundice.fit_transform(data_fin["Jaundice"])
data_fin["Family_mem_with_ASD"] = le_family.fit_transform(data_fin["Family_mem_w
data_fin["Who_completed_the_test"] = le_who.fit_transform(data_fin["Who_complete
data_fin["ASD_traits"] = le_asd_traits.fit_transform(data_fin["ASD_traits"])
data_fin["Ethnicity"] = le_ethnicity.fit_transform(data_fin["Ethnicity"])

# Afficher les mappings pour chaque colonne
print("Mapping for 'Sex':")
for i, label in enumerate(le_sex.classes_):
    print(f"{i}: {label}")

print("\nMapping for 'Jaundice':")
for i, label in enumerate(le_jaundice.classes_):
    print(f"{i}: {label}")

print("\nMapping for 'Family_mem_with_ASD':")
for i, label in enumerate(le_family.classes_):
    print(f"{i}: {label}")

print("\nMapping for 'Who_completed_the_test':")
for i, label in enumerate(le_who.classes_):
    print(f"{i}: {label}")

print("\nMapping for 'ASD_traits':")
for i, label in enumerate(le_asd_traits.classes_):
    print(f"{i}: {label}")

print("\nMapping for 'Ethnicity':")
for i, label in enumerate(le_ethnicity.classes_):
    print(f"{i}: {label}")
```

```
Mapping for 'Sex':
0: F
1: M

Mapping for 'Jaundice':
0: No
1: Yes

Mapping for 'Family_mem_with_ASD':
0: No
1: Yes

Mapping for 'Who_completed_the_test':
0: Family Member
1: Health Care Professional
2: Others
3: Parent
4: Relative
5: School and NGO
6: Self

Mapping for 'ASD_traits':
0: No
1: Yes

Mapping for 'Ethnicity':
0: Asian
1: Black
2: Hispanic
3: Latino
4: Middle Eastern
5: Mixed
6: Native Indian
7: Others
8: Pacifica
9: South Asian
10: Turkish
11: White European
12: White-European
13: others
```

In [87]:
```
data_fin.head()
```

Out[87]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10_Autism_Spectrum_Quotient	Age_Years
0	1	1	1	1	0	0	1	1	0	0	26
1	1	1	0	1	0	0	0	1	0	1	24
2	1	1	0	1	1	0	1	1	1	1	27
3	1	1	0	1	0	0	1	1	0	1	35
4	1	0	0	0	0	0	0	1	0	0	40

Séparation des données en deux ensembles d'entraînement (77%) et de test (23%) :

```
In [90]: # Séparer les caractéristiques et les étiquettes à partir du dataset augmenté
X = data_fin.drop(columns=["ASD_traits"])
y = data_fin["ASD_traits"]

# Diviser le dataset en ensembles d'entraînement et de test
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.23, random

# Vérifier la taille des ensembles d'entraînement et de test
print(f"Taille de l'ensemble d'entraînement : {x_train.shape[0]} lignes")
print(f"Taille de l'ensemble de test : {x_test.shape[0]} lignes")
```

```
Taille de l'ensemble d'entraînement : 2882 lignes
Taille de l'ensemble de test : 861 lignes
```

Normalisation des données avec MinMaxScaler :

ajuster les valeurs de chaque feature pour ameliorer la convergence

```
In [93]: sc = MinMaxScaler()
x_train_scaled = sc.fit_transform(x_train)
x_test_scaled = sc.transform(x_test)
```

Training the Model Using Logistic Regression :

```
In [96]: ## Définir la fonction d'entraînement
def train_model(model, X_train_scaled, y_train, X_test_scaled, y_test):

    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    score_df = pd.DataFrame([[accuracy, precision, recall, f1]],
                            columns=["accuracy", "precision", "recall", "f1"])
    return score_df
```

```
In [98]: #Instancier le modèle de régression logistique
model = LogisticRegression()
#Appeler la fonction pour entraîner le modèle et obtenir les résultats
results = train_model(model, x_train_scaled, y_train, x_test_scaled, y_test)

results.index = ["Logistic Regression"]
results
```

Out[98]:

	accuracy	precision	recall	f1
Logistic Regression	0.796748	0.835749	0.763797	0.798155

Ajustement des Hyperparamètres en utilisant le Classificateur Random Forest :

recherche d'hyperparamètres optimaux à l'aide de la méthode GridSearchCV de la bibliothèque scikit-learn.

In [102…

```
rf = RandomForestClassifier(random_state=42, n_jobs=-1)

params = {
    'max_depth': [2,3,5,10,20],
    'min_samples_leaf': [5,10,20,50,100,200],
    'n_estimators': [10,25,30,50,100,200]
}

from sklearn.model_selection import GridSearchCV

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rf,
                           param_grid=params,
                           cv = 4,
                           n_jobs=-1, verbose=1, scoring="accuracy")
```

In [104…

```
grid_search.fit(x_train, y_train)  #recherche d'hyperparamètres
```

```
Fitting 4 folds for each of 180 candidates, totalling 720 fits
```

Out[104…

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [105…

```
grid_search.best_score_   #Meilleure performance moyenne
```

Out[105…

```
0.9521165626444753
```

In [106…

```
rf_best = grid_search.best_estimator_
rf_best  #Modèle optimisé
```

Out[106…

```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=20, min_samples_leaf=5, n_jobs=-1,
                       random_state=42)
```

In [110…

```
# Obtenir les importances des caractéristiques du meilleur modèle Random Forest.
rf_best.feature_importances_
```

Out[110…

```
array([0.02935224, 0.04473345, 0.02329894, 0.06175604, 0.05743901,
       0.180788  , 0.09739925, 0.02140971, 0.12202275, 0.01503107,
       0.08350573, 0.0430701 , 0.12107013, 0.01107469, 0.0452435 ,
       0.0428054 ])
```

Après l'entraînement du modèle Random Forest et l'ajustement des hyperparamètres, il est essentiel d'évaluer la performance du modèle sur un ensemble de test.

In [113…

```
from sklearn.metrics import accuracy_score, classification_report, confusion_mat

y_pred = rf_best.predict(x_test)
print(f"Accuracy:{accuracy_score(y_test, y_pred) * 100} %" )
```

```python
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy:94.3089430894309 %
Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.94      0.94       408
           1       0.95      0.94      0.95       453

    accuracy                           0.94       861
   macro avg       0.94      0.94      0.94       861
weighted avg       0.94      0.94      0.94       861

Confusion Matrix:
 [[384  24]
 [ 25 428]]
```

In [115…

```python
import joblib
```

In [117…

```python
# Save the model
joblib.dump(rf_best, 'Autisme_prediction.pkl')
```

Out[117…

```
['Autisme_prediction.pkl']
```

In [ ]:

```python

```