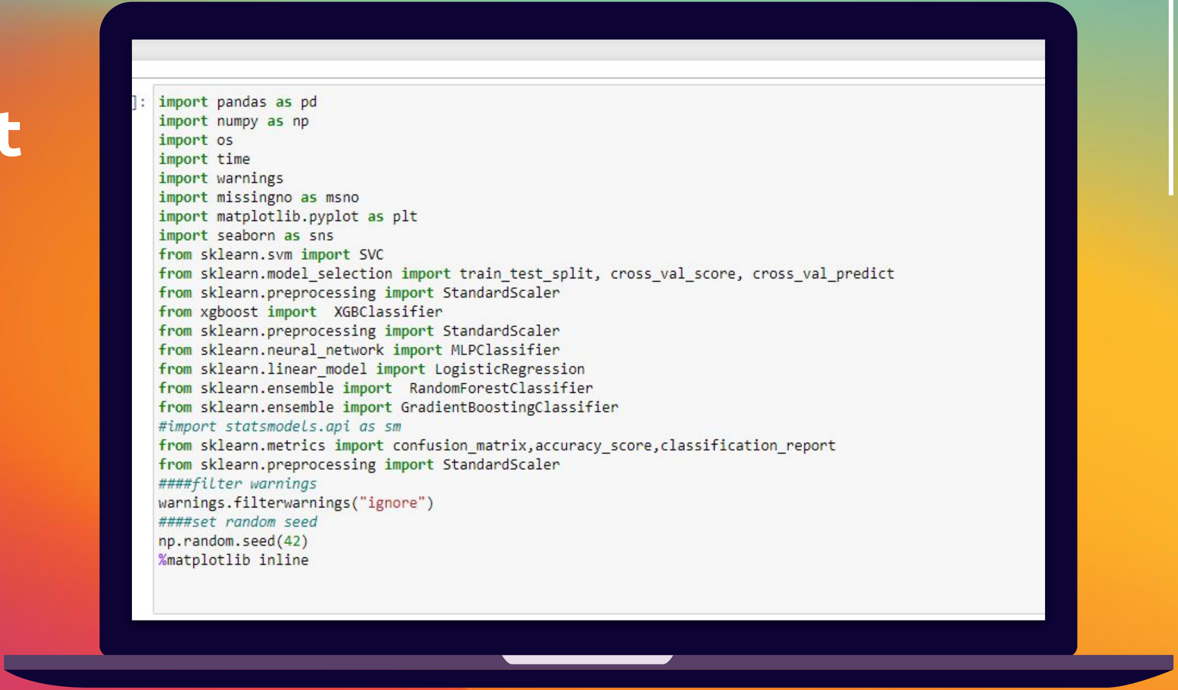
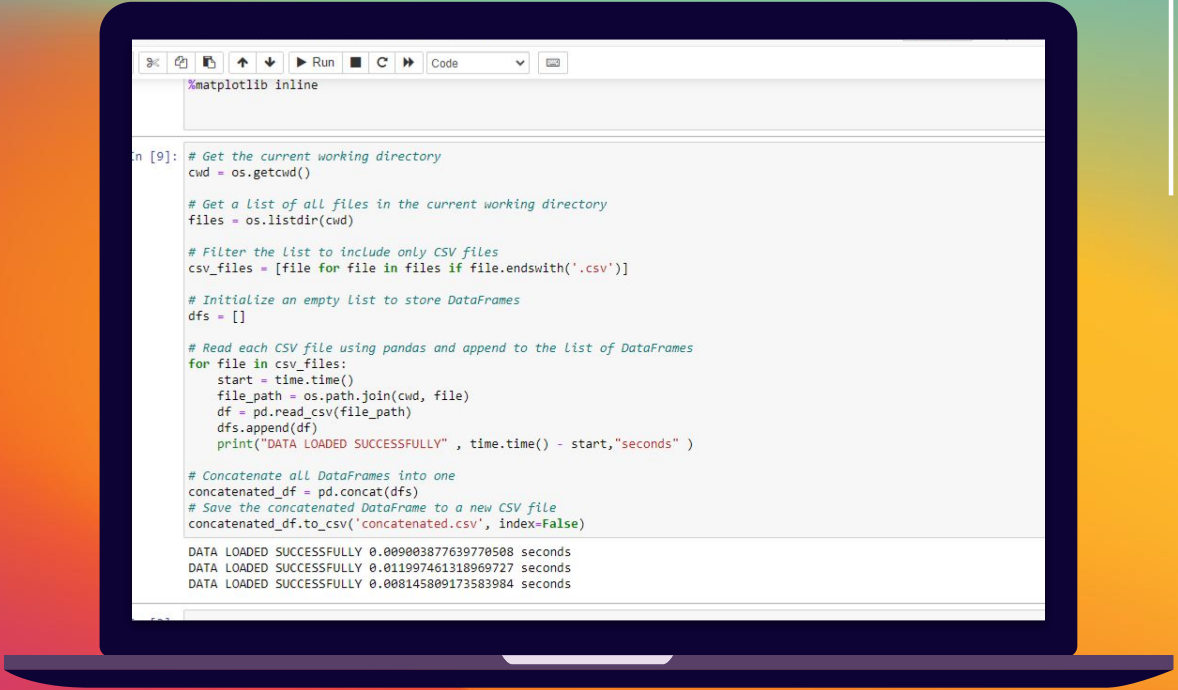
Realisation des modeles

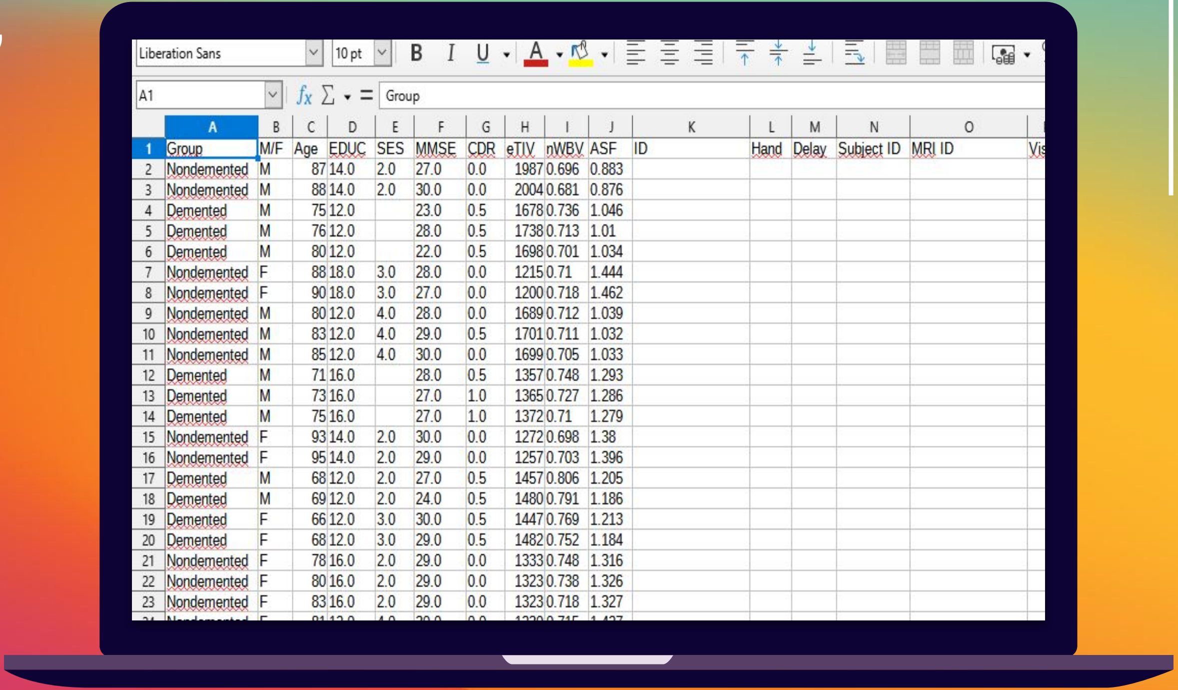
Afin de realiser nos modeles on doit premierement installer et importer les bibliotheques correspondant



Lire et concatener les fichiers csv telecharge depuis kaggle



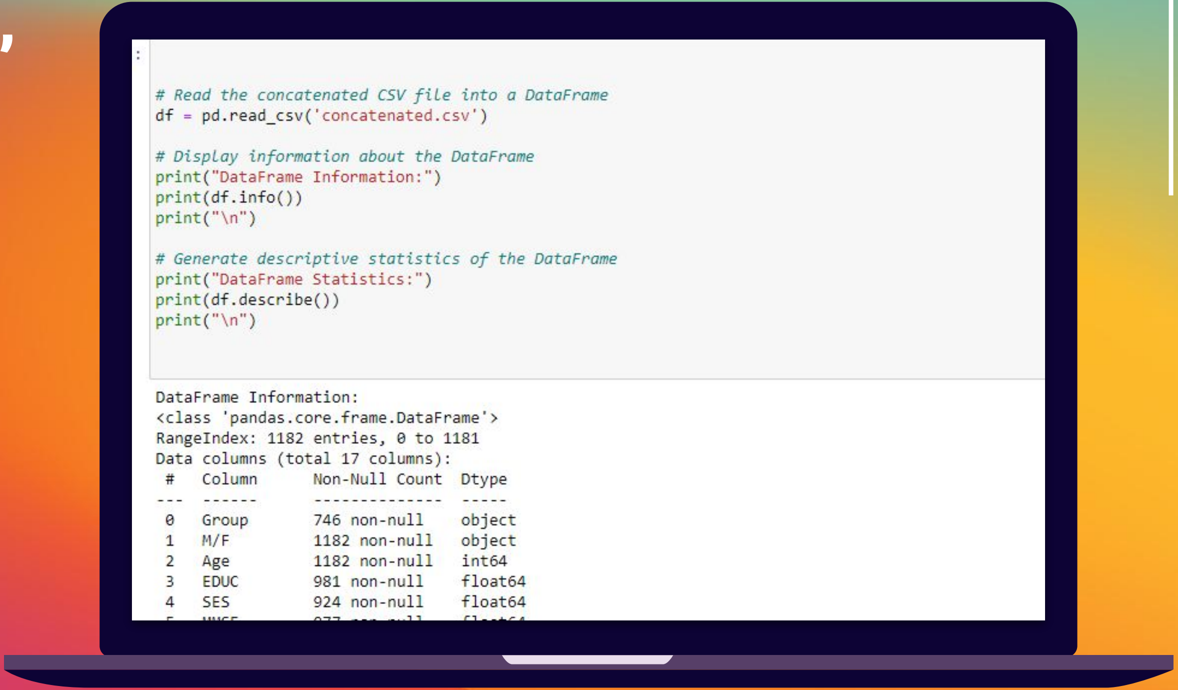
Ouvrir le fichier ‘concatenated.csv’



1	Group	M/F	Age	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF	ID	Hand	Delay	Subject ID	MRI ID	Vis
2	Nondemented	M	87	14.0	2.0	27.0	0.0	1987	0.696	0.883						
3	Nondemented	M	88	14.0	2.0	30.0	0.0	2004	0.681	0.876						
4	Demented	M	75	12.0		23.0	0.5	1678	0.736	1.046						
5	Demented	M	76	12.0		28.0	0.5	1738	0.713	1.01						
6	Demented	M	80	12.0		22.0	0.5	1698	0.701	1.034						
7	Nondemented	F	88	18.0	3.0	28.0	0.0	1215	0.71	1.444						
8	Nondemented	F	90	18.0	3.0	27.0	0.0	1200	0.718	1.462						
9	Nondemented	M	80	12.0	4.0	28.0	0.0	1689	0.712	1.039						
10	Nondemented	M	83	12.0	4.0	29.0	0.5	1701	0.711	1.032						
11	Nondemented	M	85	12.0	4.0	30.0	0.0	1699	0.705	1.033						
12	Demented	M	71	16.0		28.0	0.5	1357	0.748	1.293						
13	Demented	M	73	16.0		27.0	1.0	1365	0.727	1.286						
14	Demented	M	75	16.0		27.0	1.0	1372	0.71	1.279						
15	Nondemented	F	93	14.0	2.0	30.0	0.0	1272	0.698	1.38						
16	Nondemented	F	95	14.0	2.0	29.0	0.0	1257	0.703	1.396						
17	Demented	M	68	12.0	2.0	27.0	0.5	1457	0.806	1.205						
18	Demented	M	69	12.0	2.0	24.0	0.5	1480	0.791	1.186						
19	Demented	F	66	12.0	3.0	30.0	0.5	1447	0.769	1.213						
20	Demented	F	68	12.0	3.0	29.0	0.5	1482	0.752	1.184						
21	Nondemented	F	78	16.0	2.0	29.0	0.0	1333	0.748	1.316						
22	Nondemented	F	80	16.0	2.0	29.0	0.0	1323	0.738	1.326						
23	Nondemented	F	83	16.0	2.0	29.0	0.0	1323	0.718	1.327						

Lire le fichier ‘Concatenated.csv’

Et afficher ses informations



```python
# Read the concatenated CSV file into a DataFrame
df = pd.read_csv('concatenated.csv')

# Display information about the DataFrame
print("DataFrame Information:")
print(df.info())
print("\n")

# Generate descriptive statistics of the DataFrame
print("DataFrame Statistics:")
print(df.describe())
print("\n")
```

```
DataFrame Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1182 entries, 0 to 1181
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Group      746 non-null    object
 1   M/F        1182 non-null   object
 2   Age        1182 non-null   int64
 3   EDUC       981 non-null    float64
 4   SES        924 non-null    float64
```

```
print(df.describe())
print("\n")
```

```
DataFrame Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1182 entries, 0 to 1181
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Group       746 non-null    object
 1   M/F         1182 non-null   object
 2   Age         1182 non-null   int64
 3   EDUC        981 non-null    float64
 4   SES         924 non-null    float64
 5   MMSE        977 non-null    float64
 6   CDR         981 non-null    float64
 7   eTIV        1182 non-null   int64
 8   nWBV        1182 non-null   float64
 9   ASF         1182 non-null   float64
 10  ID          436 non-null    object
 11  Hand        809 non-null    object
 12  Delay       20 non-null     float64
 13  Subject ID  373 non-null    object
 14  MRI ID      373 non-null    object
 15  Visit       373 non-null    float64
 16  MR Delay    373 non-null    float64
dtypes: float64(9), int64(2), object(6)
memory usage: 157.1+ KB
None


DataFrame Statistics:
               Age        EDUC          SES         MMSE          CDR  \
```

On voit les nombre de valeurs dans chaque colonne
On se debarasse des colonne qui ont beaucoup de valeurs nulle et les information non necessaires pour notre modele

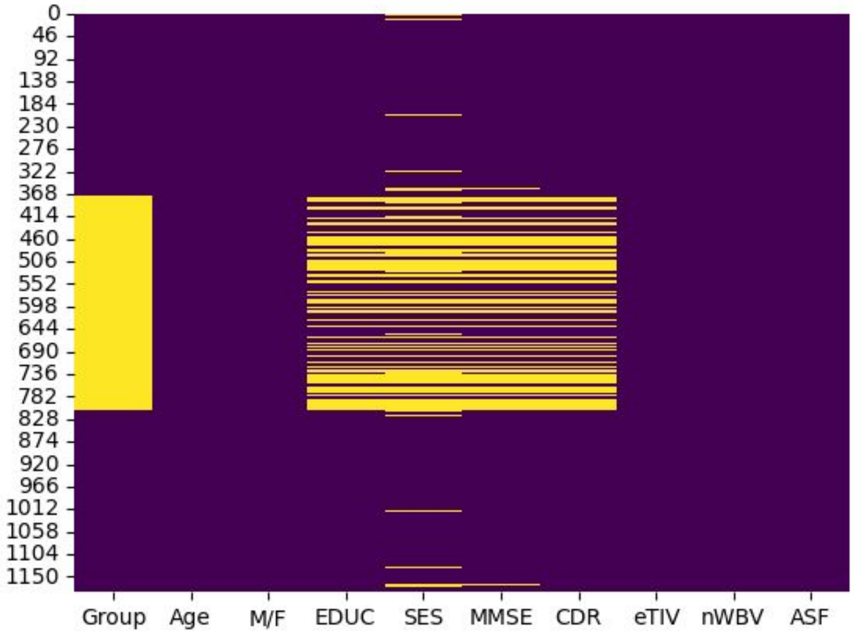
```
None


DataFrame Statistics:
               Age        EDUC         SES        MMSE         CDR  \
count  1182.000000  981.000000  924.000000  977.000000  981.000000
mean     67.549915   11.862385    2.467532   27.275333    0.289501
std      20.623978    5.519947    1.129735    3.684668    0.376317
min      18.000000    1.000000    1.000000    4.000000    0.000000
25%      65.000000    8.000000    2.000000   26.000000    0.000000
50%      74.000000   13.000000    2.000000   29.000000    0.000000
75%      81.000000   16.000000    3.000000   30.000000    0.500000
max      98.000000   23.000000    5.000000   30.000000    2.000000

              eTIV         nWBV          ASF     Delay       Visit  \
count  1182.000000  1182.000000  1182.000000  20.00000  373.000000
mean   1485.838409     0.752475     1.196728  20.55000    1.882038
std     169.810030     0.055593     0.134593  23.86249    0.922843
min    1106.000000     0.644000     0.876000   1.00000    1.000000
25%    1360.000000     0.709000     1.107000   2.75000    1.000000
50%    1474.000000     0.742000     1.190000  11.00000    2.000000
75%    1586.750000     0.788750     1.291000  30.75000    2.000000
max    2004.000000     0.893000     1.587000  89.00000    5.000000

          MR Delay
count   373.000000
mean    595.104558
std     635.485118
min       0.000000
25%       0.000000
50%     552.000000
75%     873.000000
max    2639.000000
```

Ici on voit les statistique du tableau:
Count: nombre de valeurs
Mean: moyenne
Std: la deviation
Min: le minimum
Max: le maximum
Les pourcentage 25%,50%(médiane)
Et 75% représente le pourcentage de l'apparition de la valeur

```
]: df = df[['Group','Age','M/F','EDUC','SES','MMSE','CDR','eTIV','nWBV','ASF']]
```

```
]: sns.heatmap(df.isnull(), cmap='viridis', cbar=False)
```

```
]: <Axes: >
```



On élimine les colonnes

On represente les valeurs manquees en jaune

```
df['M/F'] = df['M/F'].replace(['F','M'], [0,1])
df['Group'] = df['Group'].replace(['Converted'], ['Demented'])
df['Group'] = df['Group'].replace(['Demented', 'Nondemented'], [1,0])
```

On transforme les caracteres
En donnees numerique

```
# Import the necessary library for advanced imputation
from sklearn.impute import KNNImputer

# Create an instance of the KNNImputer
imputer = KNNImputer()

# Apply the imputation to fill missing values
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
```

```
# columns_to_convert = ['Group','Age','M/F','eTIV']

# for column in columns_to_convert:
#     df_imputed[column] = df_imputed[column].astype(int)

# df_imputed.to_csv('concatenated_imputed.csv', index=False)
```

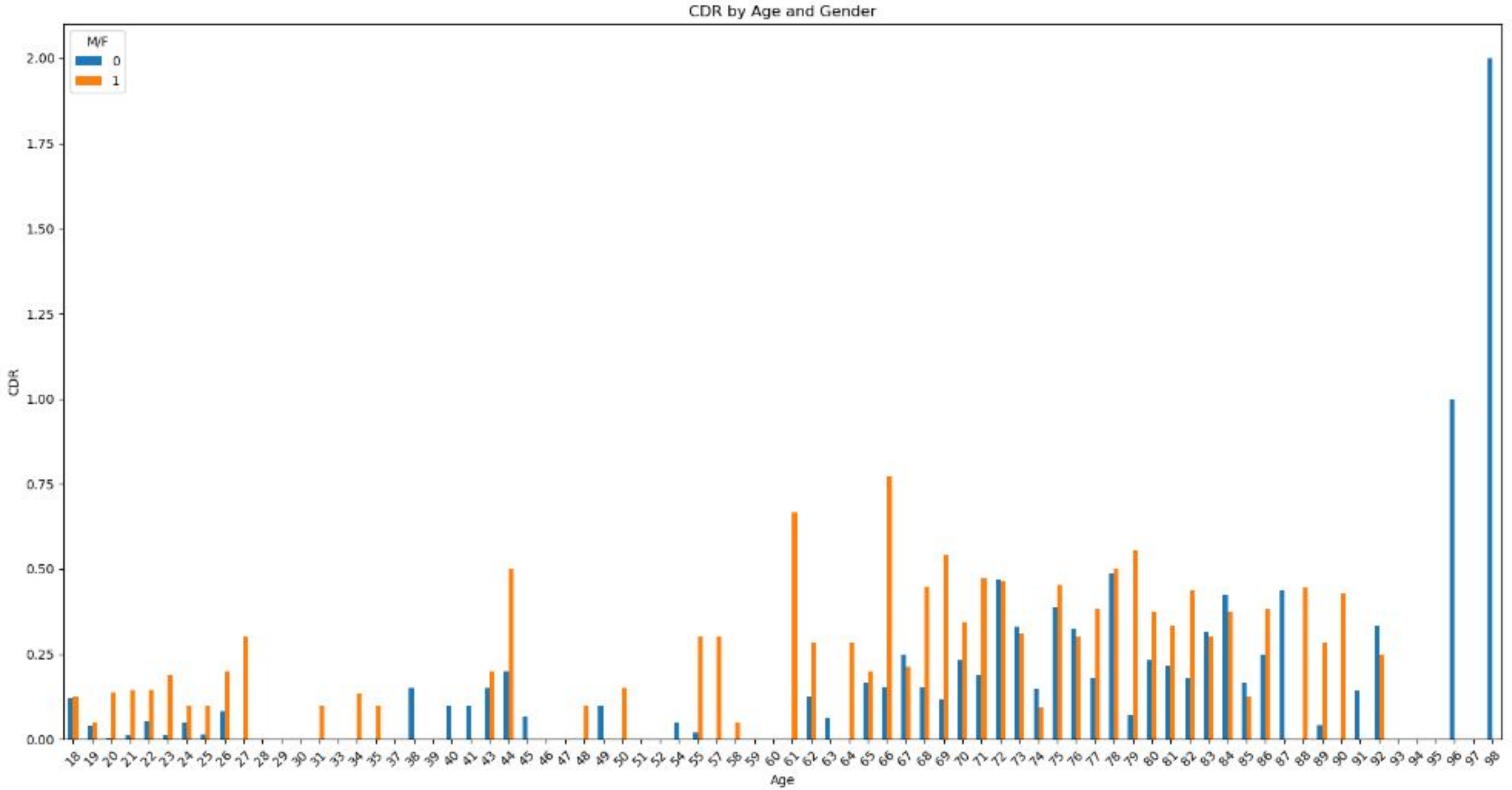
On remplit les valeur
nulles avec une module
d'esttimation

```
grouped_data = df_imputed.groupby(['Age', 'M/F'])['CDR'].mean().unstack()

# Set the plot size
# Plot the grouped data as a bar plot
grouped_data.plot(kind='bar',width=0.5,figsize=(20,10))
# Set plot labels and title
plt.xlabel('Age')
plt.ylabel('CDR')
```

Visualisation

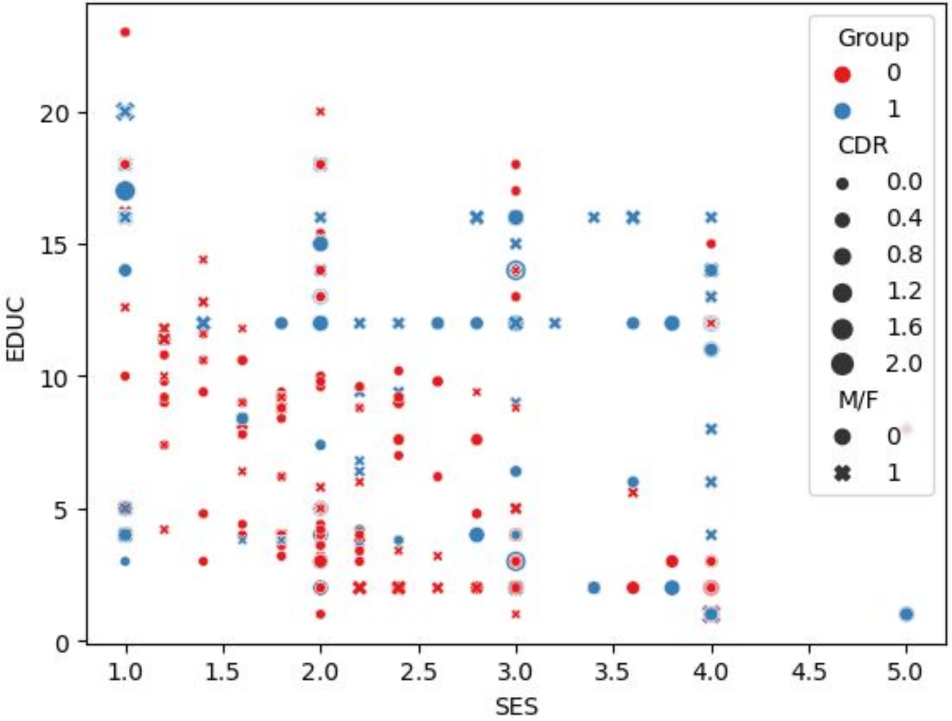
Après qu'on a ecarter les colonnes unnecessaires et rempli le tableau avec des estimations, on visualise les données du tableau. Pourquoi? Trouver les relation entes les colonnes et trouver les valeurs aberrantes

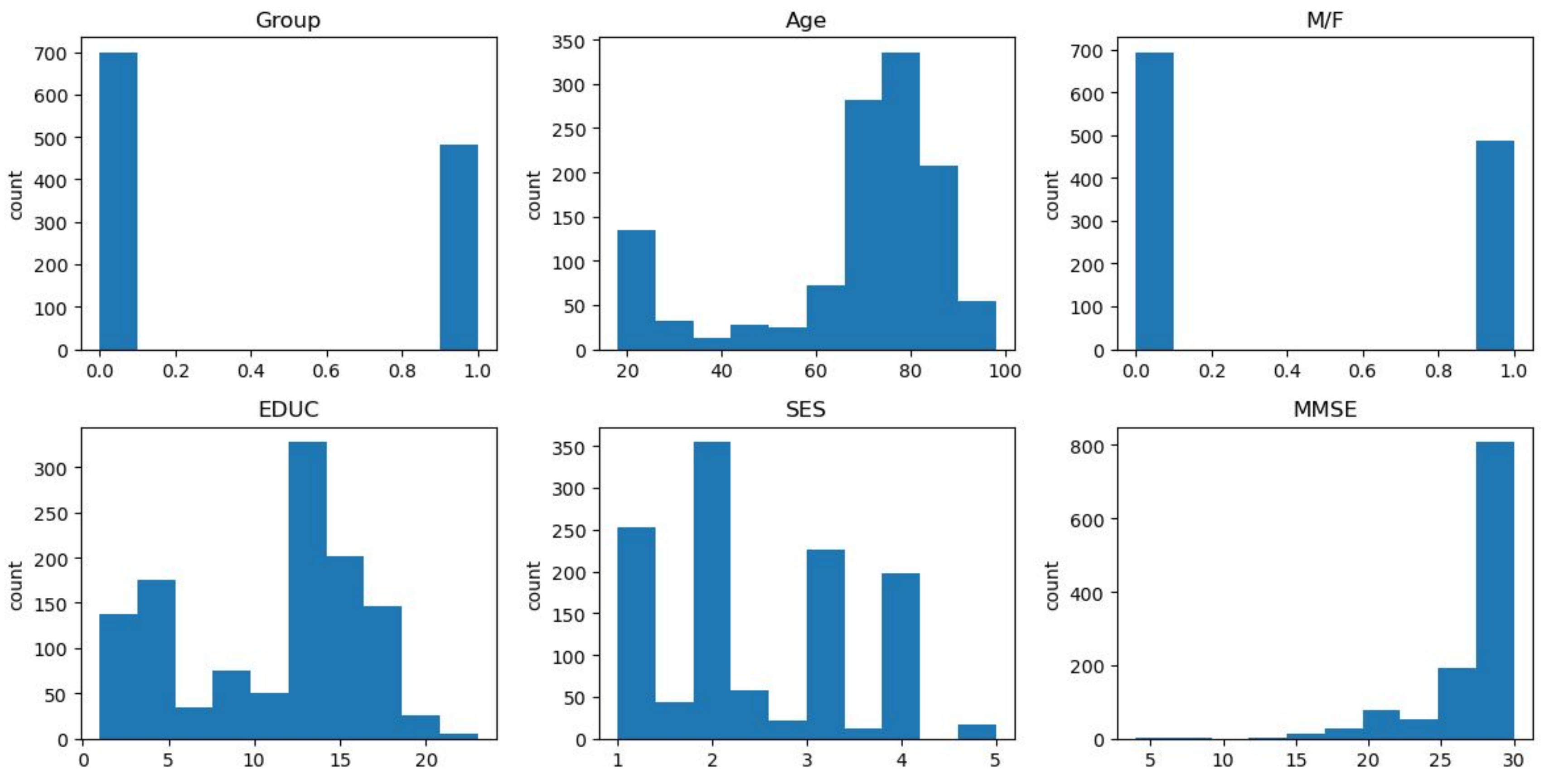
```
plt.show()
```


CDR by Age and Gender
M/F
0
1
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
CDR
Age
18 19 20 21 22 23 24 25 26 27 28 29 30 31 33 34 35 37 38 39 40 41 43 44 45 46 47 48 49 50 51 52 54 55 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98

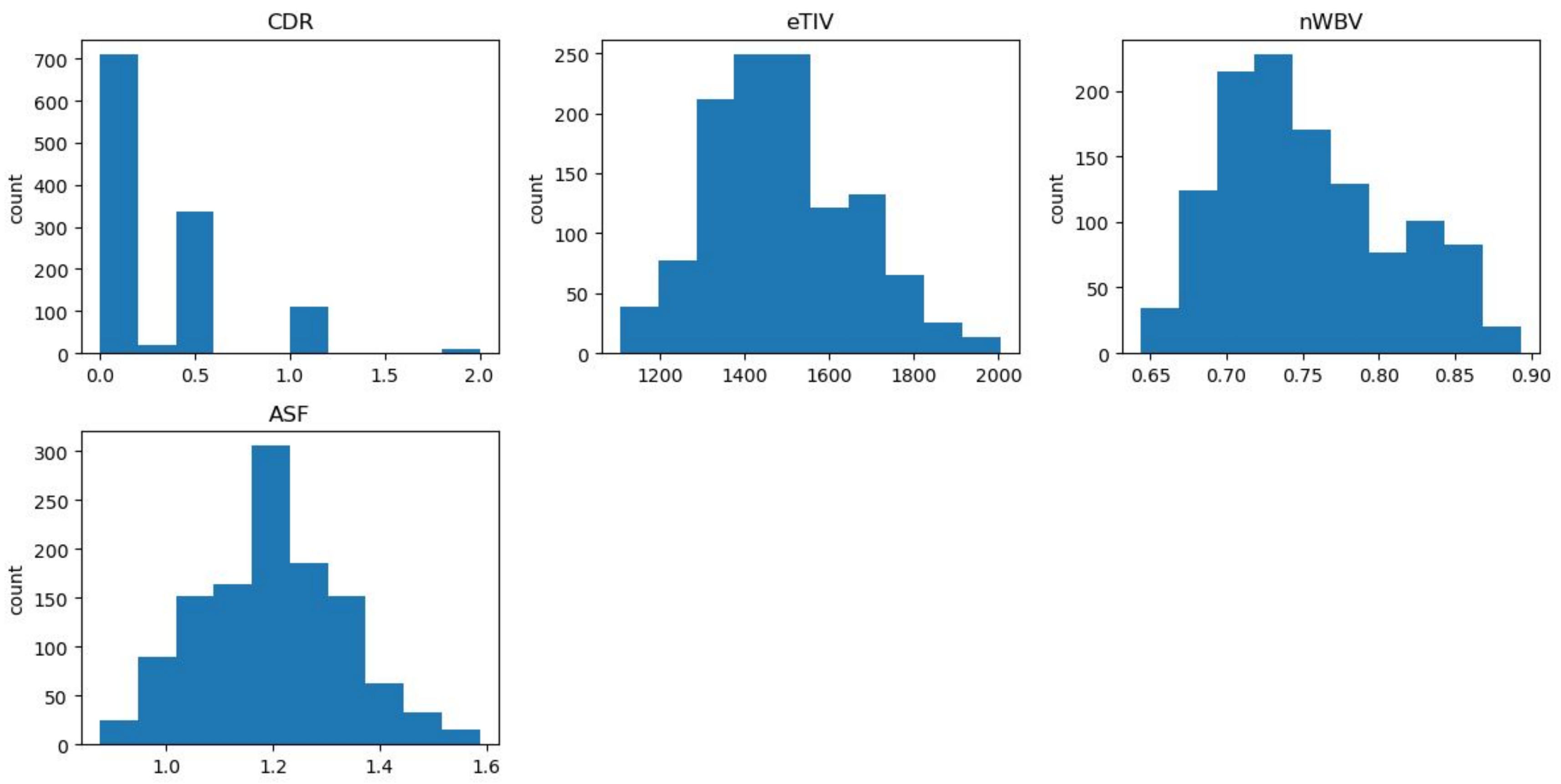
```
sns.scatterplot(data=df_imputed, x="SES", y="EDUC",hue="Group", style="M/F", size="CDR", palette="Set1")
```

Out[21]: <Axes: xlabel='SES', ylabel='EDUC'>

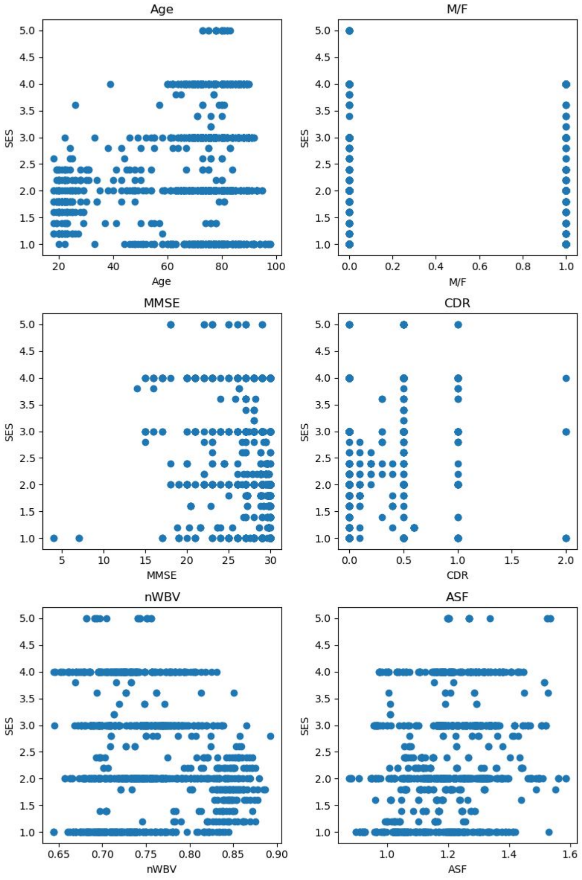


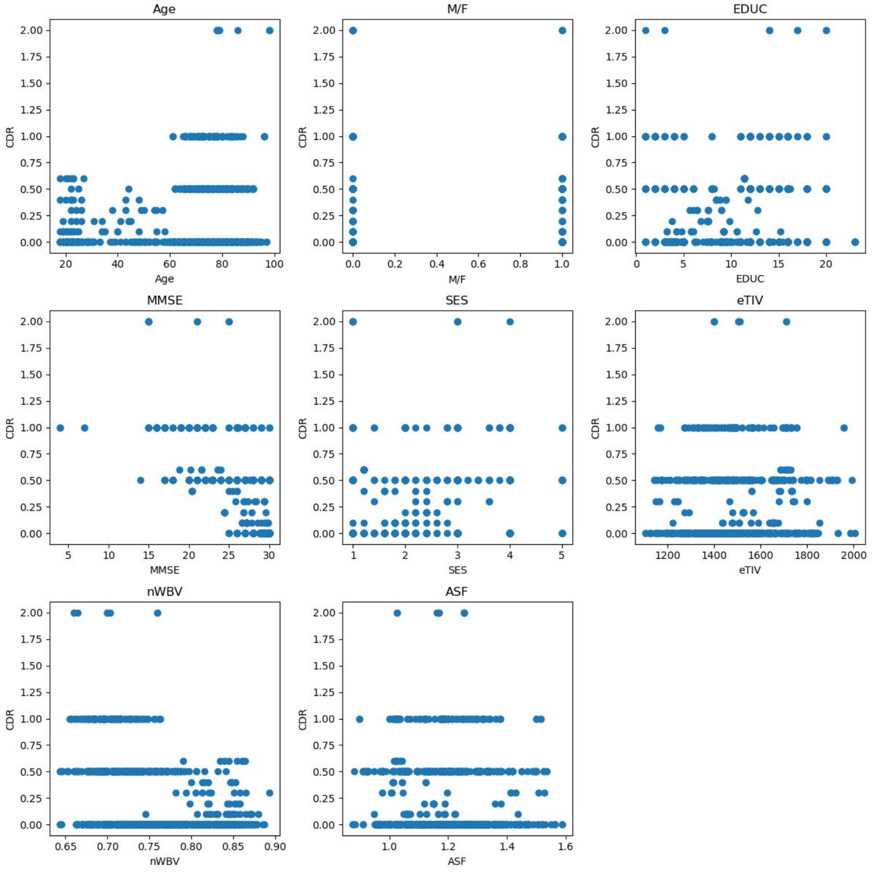


Group
Age
M/F
EDUC
SES
MMSE
count



CDR
count
700
600
500
400
300
200
100
0
0.0
0.5
1.0
1.5
2.0
eTIV
count
250
200
150
100
50
0
1200
1400
1600
1800
2000
nWBV
count
200
150
100
50
0
0.65
0.70
0.75
0.80
0.85
0.90
ASF
count
300
250
200
150
100
50
0
1.0
1.2
1.4
1.6


Age
M/F
EDUC
MMSE
CDR
eTIV
nWBV
ASF
SES


Age
M/F
EDUC
MMSE
SES
eTIV
nWBV
ASF
CDR

Mise en echelle et scinder

Pour traîner et augmenter la performance des algorithmes de machine learning

```
X =  df_imputed.drop('Group',axis=1)
y = df_imputed['Group']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
X_train[:3]
```

	Age	M/F	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF
650	67.0	1.0	4.0	2.0	23.0	0.5	1399.0	0.735	1.255
976	73.0	1.0	16.0	2.0	29.0	0.0	1931.0	0.722	0.909
1177	82.0	1.0	16.0	1.0	28.0	0.5	1693.0	0.694	1.037

On veut predire le groupe alors
On scinde le table en 20% pour les
Variables de test et le reste pour
pour trainer

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

```
X_train[:3]
```

	Age	M/F	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF
650	67.0	1.0	4.0	2.0	23.0	0.5	1399.0	0.735	1.255
976	73.0	1.0	16.0	2.0	29.0	0.0	1931.0	0.722	0.909
1177	82.0	1.0	16.0	1.0	28.0	0.5	1693.0	0.694	1.037

La mise en echelle des
variables

```
rfc = RandomForestClassifier(n_estimators=200)
rfc.fit(X_train,y_train)
pred_rfc = rfc.predict(X_test)
```

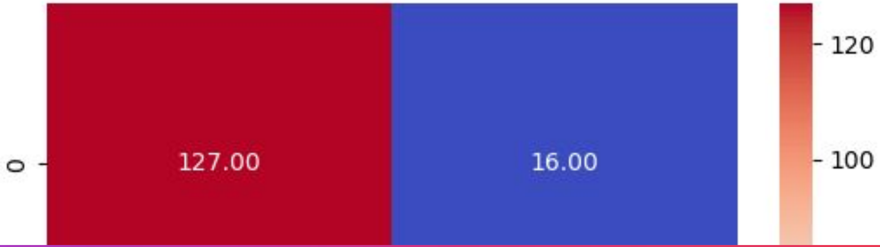
Trainer les modeles

On traine les modeles pour predire le variable ‘Group’

```
mlp = MLPClassifier(max_iter=500)
mlp.fit(X_train,y_train)
pred_mlp = mlp.predict(X_test)
print(classification_report(y_test,pred_mlp))
sns.heatmap(confusion_matrix(y_test,pred_mlp), annot=True, fmt=".2f", cmap="coolwarm")
print('ACCURACY SCORE',accuracy_score(y_test,pred_mlp))
plt.figure(figsize=(8, 6))
```

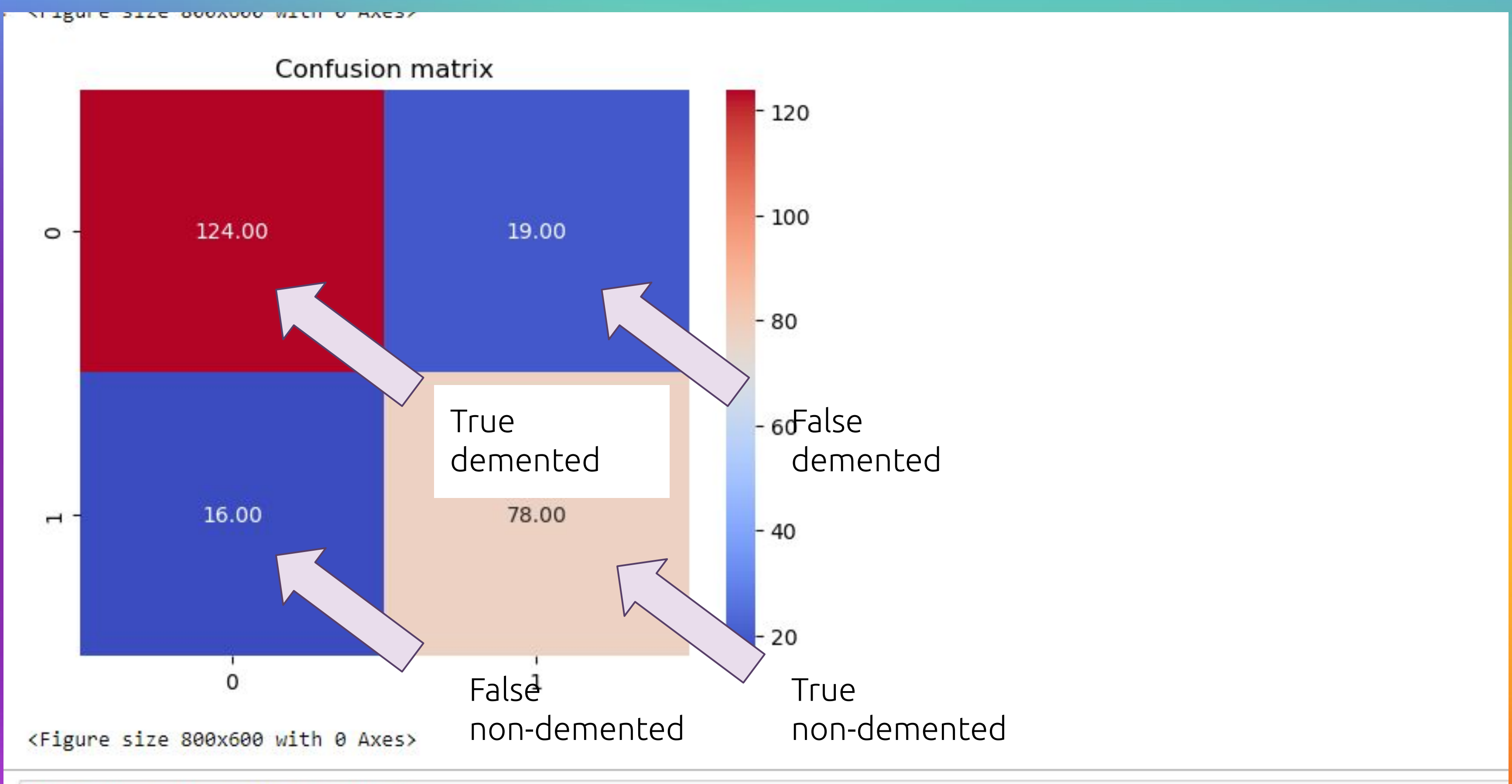
```
              precision    recall  f1-score   support

           0       0.88      0.89      0.88       143
           1       0.83      0.81      0.82        94

    accuracy                           0.86       237
   macro avg       0.85      0.85      0.85       237
weighted avg       0.86      0.86      0.86       237

ACCURACY SCORE 0.8565400843881856
```

```
<Figure size 800x600 with 0 Axes>
```



Le pourcentage de succès

Le rapport de classification
Precision measures how many of the predicted positive labels are actually correct.
Recall It measures the model's ability to correctly identify positive instances.
Fi-score measure of the model's performance by considering both precision and recall.
Support represents the number of instances in each class in the test dataset.


<Figure size 800x600 with 0 Axes>
Confusion matrix
124.00
19.00
16.00
78.00
0
1
0
1
120
100
80
60
40
20
True
demented
False
demented
False
non-demented
True
non-demented
<Figure size 800x600 with 0 Axes>

Sauvegarder les modeles

On utilise la module joblib pour sauvegarder les modeles haut performant en format model.joblib

```
import joblib
joblib.dump(mlp, 'model_mlp.pkl')
joblib.dump(logreg, 'model_logreg.pkl')
```

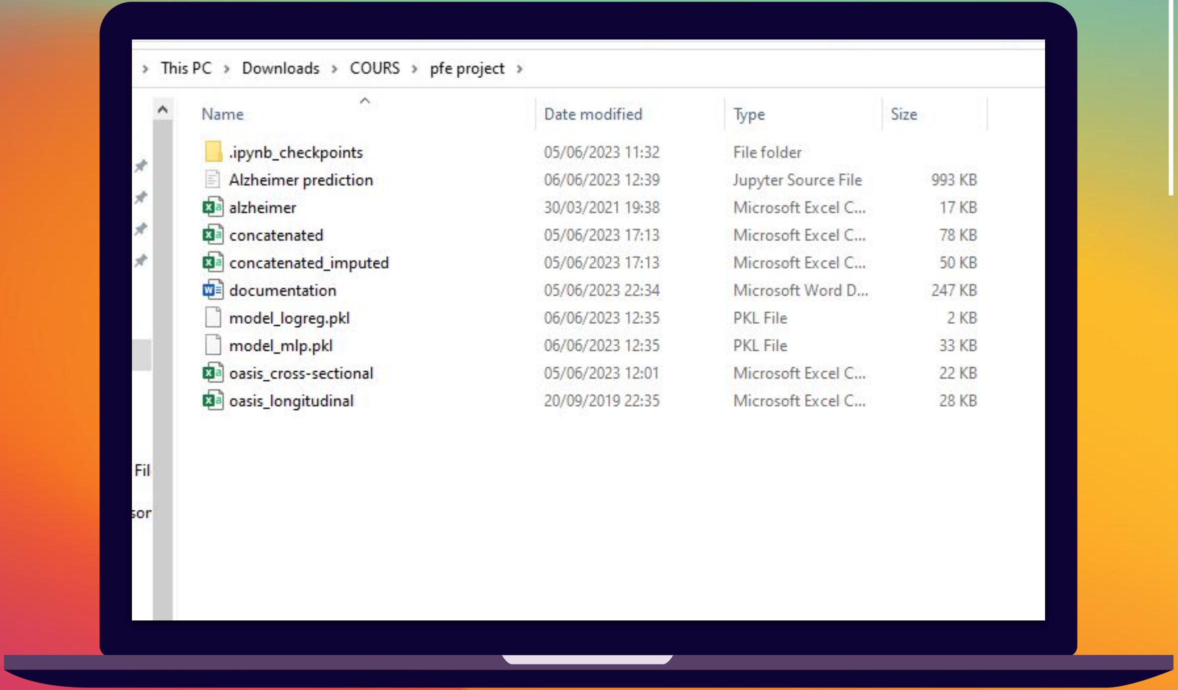
```
['model_logreg.pkl']
```

```
lg = joblib.load('model_logreg.pkl')
pred_log = lg.predict(X_test)
print(classification_report(y_test,pred_log))
sns.heatmap(confusion_matrix(y_test,pred_log), annot=True, fmt=".2f", cmap="coolwarm")
print('ACCURACY SCORE',accuracy_score(y_test,pred_log))
plt.figure(figsize=(8, 6))
```

```
              precision    recall  f1-score   support

           0       0.89      0.87      0.88       143
           1       0.80      0.83      0.82        94

    accuracy                           0.85       237
   macro avg       0.84      0.85      0.85       237
weighted avg       0.85      0.85      0.85       237

ACCURACY SCORE 0.8523206751054853
```

```
<Figure size 800x600 with 0 Axes>
```

Les fichiers a la fin De traitement



Thanks!

Any questions?