

# Alzheimer

February 26, 2024

## 1 Alzheimer disease training

## 2 Entraînement sur des données personnelles partagées dans le cadre du projet gratuit OASIS

Sources: [Haute Autorité Santé](#), [JACOB BOYSEN](#), [Oasis](#), [Association Daniel Goutaine](#), [Fares Sayah](#), [Shreyan Chaudhary](#)

### 2.1 Définition du problème

Alzheimer est une maladie qui pousse le sujet à progresser vers un état de démence dont notamment avec la perte de mémoire épisodique (mémoire des souvenirs) et sémantique (mémoire de connaissances générales sur le monde). Il a été observé plusieurs facteurs communs chez les malades: - Une accumulation anormale de protéines bêta-amyloïde. Une alteration de cette protéine protectrice pourrait être responsable de la formation de plaques séniles. Ces plaques conduiraient à des dégénérescences neurofibrillaires soit, la mort de neurones par interruption de transport intracellulaire. - un défaut du système productif de neurotransmetteur : l'acétylcholine. Ce neurotransmetteur jouerait un rôle important dans la mémorisation et l'apprentissage. Sa production semblerait être altérée chez plusieurs sujets touchés par la maladie.

Cependant, ces hypothèses sont remises en cause car elles ne seraient pas fortement corrélées à la sévérité ou à l'apparition des symptômes d'Alzheimer.

A partir du dataset du projet OASIS, nous essayerons de prédire à partir de ses données médicales si un individu est touché par la maladie d'Alzheimer.

### 2.2 Importations

```
[2]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import numpy as np
```

```
[2]: data_mri = pd.read_csv("../data/oasis_longitudinal.csv")
```

## 2.3 Explication des données

```
[3]: data_mri.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 373 entries, 0 to 372
Data columns (total 15 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Subject ID      373 non-null   object
1   MRI ID          373 non-null   object
2   Group           373 non-null   object
3   Visit           373 non-null   int64
4   MR Delay        373 non-null   int64
5   M/F             373 non-null   object
6   Hand            373 non-null   object
7   Age             373 non-null   int64
8   EDUC            373 non-null   int64
9   SES             354 non-null   float64
10  MMSE            371 non-null   float64
11  CDR              373 non-null   float64
12  eTIV            373 non-null   int64
13  nWBV            373 non-null   float64
14  ASF             373 non-null   float64
dtypes: float64(5), int64(5), object(5)
memory usage: 43.8+ KB
```

Nous possédons les données suivantes: les ID, le groupe de l'individu (attribut à prédire), son nombre de visites, le MR Delay, son genre, sa main d'écriture principale, son age, son niveau d'éducation, son statut social et économique, son état mental, sa note clinique de démence, la taille totale estimée du volume de sa boîte crânienne, le volume normalisé de son cerveau et enfin le facteur d'échelle de l'Atlas.

Explications approfondies des attributs: 1. "MR Delay": Délai de l'IRM alias le temps avant que l'image soit obtenue. 2. "SES": Le statut social et économique (sur une échelle de 1 à 5) où le niveau 1 est considéré comme basse-classe et 5 haute-classe. 3. "MMSE": Note de l'examen de l'état mental du patient. stade léger ( $MMSE > 20$ ), stade modéré ( $10 < MMSE < 20$ ), stade sévère ( $MMSE < 10$ ). 4. "CDR": note clinique de la démence du patient (sur une échelle de 0 à 3) où le niveau 0 est considéré comme absence de troubles et 3 troubles sévères. 5. "eTIV": volume normalisé de la boîte crânienne du patient 6. "nWBV": volume normalisé du cerveau du patient 7. "ASF": Facteur progressif de l'Atlas: permet de normaliser le volume du cerveau du patient par rapport à la moyenne humaine.

```
[4]: data_mri.describe()
```

```
[4]:
```

	Visit	MR Delay	Age	EDUC	SES \
count	373.000000	373.000000	373.000000	373.000000	354.000000
mean	1.882038	595.104558	77.013405	14.597855	2.460452
std	0.922843	635.485118	7.640957	2.876339	1.134005

min	1.000000	0.000000	60.000000	6.000000	1.000000
25%	1.000000	0.000000	71.000000	12.000000	2.000000
50%	2.000000	552.000000	77.000000	15.000000	2.000000
75%	2.000000	873.000000	82.000000	16.000000	3.000000
max	5.000000	2639.000000	98.000000	23.000000	5.000000

	MMSE	CDR	eTIV	nWBV	ASF
count	371.000000	373.000000	373.000000	373.000000	373.000000
mean	27.342318	0.290885	1488.128686	0.729568	1.195461
std	3.683244	0.374557	176.139286	0.037135	0.138092
min	4.000000	0.000000	1106.000000	0.644000	0.876000
25%	27.000000	0.000000	1357.000000	0.700000	1.099000
50%	29.000000	0.000000	1470.000000	0.729000	1.194000
75%	30.000000	0.500000	1597.000000	0.756000	1.293000
max	30.000000	2.000000	2004.000000	0.837000	1.587000

L'âge moyen des données est de **77 ans** et les données vont jusqu'à 98 ans.

On peut remarquer que le premier quartile est de **71 ans**, un âge assez important qui pourrait naturellement favoriser l'atrophie du cerveau.

On observe enfin que le troisième quartile est de **82 ans**.

```
[5]: fig, axs = plt.subplots(2, 2)
axs[0, 0].boxplot(data_mri["Age"])
axs[0, 0].set_title('Age')
axs[0, 0].set_xlabel("Patients")
axs[0, 0].set_ylabel("Age")

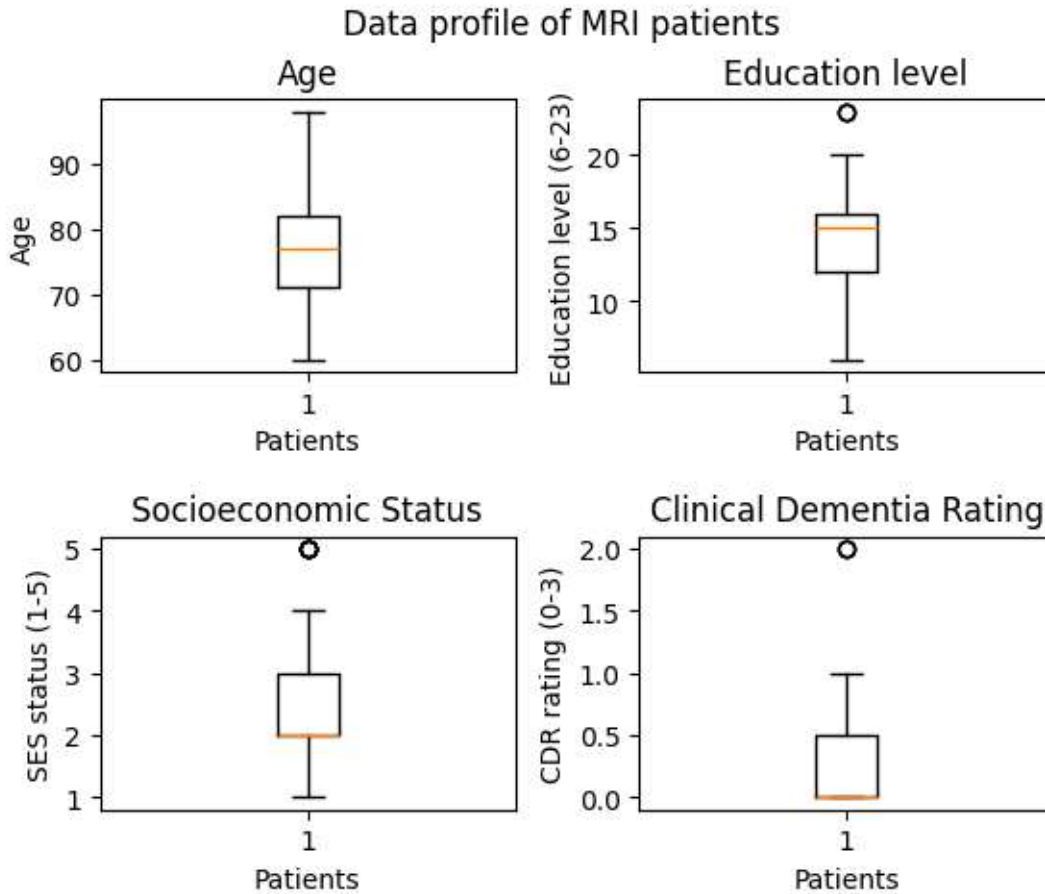
axs[0, 1].boxplot(data_mri["EDUC"])
axs[0, 1].set_title('Education level')
axs[0, 1].set_xlabel("Patients")
axs[0, 1].set_ylabel("Education level (6-23)")

axs[1, 0].boxplot(data_mri["SES"].dropna())
axs[1, 0].set_title('Socioeconomic Status')
axs[1, 0].set_xlabel("Patients")
axs[1, 0].set_ylabel("SES status (1-5)")

axs[1, 1].boxplot(data_mri["CDR"])
axs[1, 1].set_title('Clinical Dementia Rating')
axs[1, 1].set_xlabel("Patients")
axs[1, 1].set_ylabel("CDR rating (0-3)")

fig.subplots_adjust(hspace=0.6, wspace=0.3)
fig.suptitle('Data profile of MRI patients')

plt.show()
```



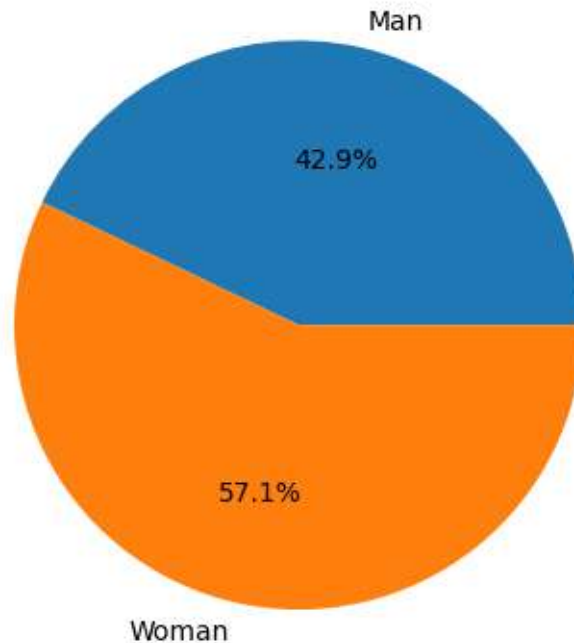
A partir de ces données, on peut observer que nous avons une moyenne du statut social et économique des patients qui se situe autour de 2/5. Les données ne semblent pas assez bien représenter l'étendu des niveaux socio-économiques. On peut aussi observer que le niveau d'éducation semble être en moyenne autour de 15/23. Nous avons des données qui semble presque représenter l'étendu des niveaux d'éducation.

Nous devons aussi vérifier si les données représentent correctement la population en regardant les distributions du sexe.

```
[6]: data_mri_man = data_mri[data_mri['M/F'] == 'M']
      data_mri_woman = data_mri[data_mri['M/F'] == 'F']

[7]: plt.pie(np.array([len(data_mri_man), len(data_mri_woman)]), labels=['Man', 'Woman'], autopct='%1.1f%%')
      plt.title('Gender distribution in dataset')
      plt.show()
```

Gender distribution in dataset



Nous avons donc une répartition majoritaire de femmes et une répartition minoritaire d'hommes. Les répartitions semblent convenables pour être interprétées dans le cadre du Machine Learning.

Cependant, nous devons aussi savoir s'il y a une distribution équivalente d'hommes déments, non déments et de femmes déments, non déments.

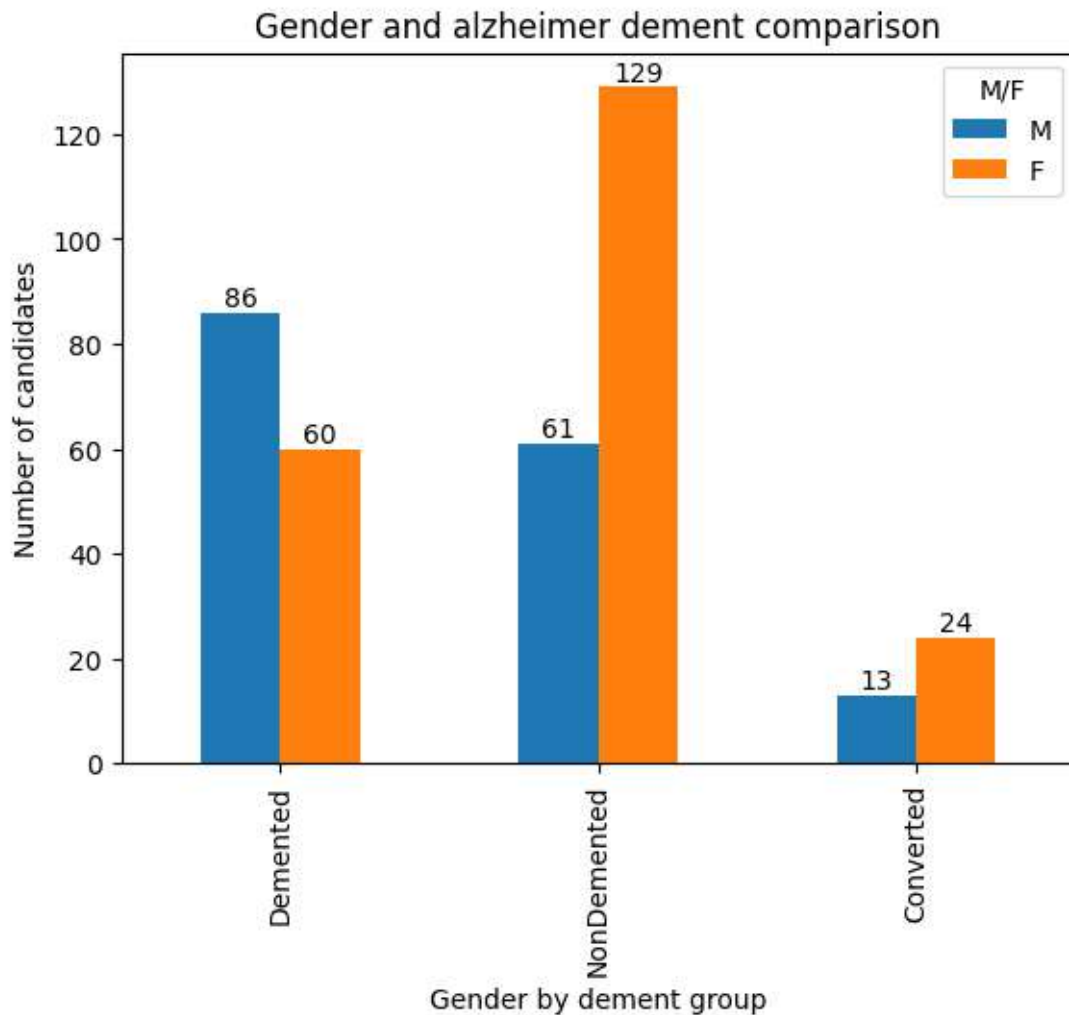
Pour cela, on crée un Dataframe qui réunira le nombre de femmes et hommes déments, non déments et convertis.

```
[8]: demented_data = data_mri[data_mri['Group'] == 'Demented']['M/F'].value_counts()
nondemented_data = data_mri[data_mri['Group'] == 'Nondemented']['M/F'].
    ↪value_counts()
converted_data = data_mri[data_mri['Group'] == 'Converted']['M/F'].
    ↪value_counts()

bar_data = pd.DataFrame([demented_data, nondemented_data, converted_data],
    ↪index=["Demented", "NonDemented", "Converted"])
```

```
[9]: ax = bar_data.plot(kind="bar")
plt.title("Gender and alzheimer dement comparison")
for container in ax.containers:
    ax.bar_label(container)
```

```
plt.xlabel("Gender by dement group")
plt.ylabel("Number of candidates")
plt.show()
```



D'après le diagramme en barres, on voit qu'il y a un grand nombre majoritaire d'hommes touchés par la maladie d'Alzheimer et une grande partie majoritaire de femmes non touchés par la maladie. Le nombre de convertis en dément sont assez rapprochés. Le modèle prédictif pourrait alors accorder un poids particulièrement important au genre pour prédire une démence. Cependant, nous voyons bien que cela reste une corrélation. Seul un genre ne pourrait pas déterminer la conversion vers la maladie d'Alzheimer.

## 2.4 Nettoyage des données

### 2.4.1 Problème 1: Features “Hand” et ID inutiles

```
[10]: data_mri.select_dtypes(include='object').describe()
```

```
[10]:
```

	Subject ID	MRI ID	Group	M/F	Hand
count	373	373	373	373	373
unique	150	373	3	2	1
top	OAS2_0070	OAS2_0001_MR1	Nondemented	F	R
freq	5	1	190	213	373

On a cinq colonnes qualitatives: les deux ID, la classification, le genre et la main préférée pour écrire.

On peut remarquer un autre problème: les données ne contiennent que des données de personnes dont la main droite est considérée comme forte.

*On la supprime des données.*

```
[11]: data_mri.drop('Hand', axis=1, inplace=True)
```

```
[12]: data_mri.drop('Subject ID', axis=1, inplace=True)
```

```
[13]: data_mri.drop('MRI ID', axis=1, inplace=True)
```

### 2.4.2 Problème 2: Features nominales

```
[14]: data_mri.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 373 entries, 0 to 372
Data columns (total 12 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Group       373 non-null    object
1   Visit       373 non-null    int64
2   MR Delay    373 non-null    int64
3   M/F        373 non-null    object
4   Age        373 non-null    int64
5   EDUC       373 non-null    int64
6   SES        354 non-null    float64
7   MMSE       371 non-null    float64
8   CDR        373 non-null    float64
9   eTIV       373 non-null    int64
10  nWBV       373 non-null    float64
11  ASF        373 non-null    float64
dtypes: float64(5), int64(5), object(2)
memory usage: 35.1+ KB
```

Nous avons 2 features nomiales. Afin que celles-ci puissent être utilisées, nous devons les normaliser. Nous n'avons pas besoin d'utiliser de fonctions pré-faites étant donné que nous connaissons les données et que les valeurs nominales possibles sont courtes: - Pour le group: nous avons "Demented" "Nondemented" ou "Converted". Nous considérerons que "Converted" est un "Demented". - Pour le genre: nous avons "F" ou "M".

On peut donc convertir ces deux features en features binaires.

```
[15]: data_mri["M/F"] = data_mri["M/F"].transform(lambda x: 1 if x == 'F' else 0)
```

```
[16]: data_mri["M/F"].describe()
```

```
[16]: count      373.000000
      mean        0.571046
      std         0.495592
      min         0.000000
      25%         0.000000
      50%         1.000000
      75%         1.000000
      max         1.000000
      Name: M/F, dtype: float64
```

```
[17]: data_mri["Group"] = data_mri["Group"].transform(lambda x: 1 if x == 'Demented'
      or x == 'Converted' else 0)
```

```
[18]: data_mri["Group"].describe()
```

```
[18]: count      373.000000
      mean        0.490617
      std         0.500583
      min         0.000000
      25%         0.000000
      50%         0.000000
      75%         1.000000
      max         1.000000
      Name: Group, dtype: float64
```

### 2.4.3 Problème 3: Valeurs manquantes.

```
[19]: data_mri.isna().sum()
```

```
[19]: Group          0
      Visit         0
      MR Delay      0
      M/F           0
      Age           0
      EDUC          0
      SES          19
```

```
MMSE          2
CDR            0
eTIV          0
nWBV          0
ASF           0
dtype: int64
```

On peut ainsi remarquer qu'il manque uniquement des données dans les deux catégories suivantes: SES et MMSE.

On a ainsi trois possibilités cohérentes pour nos données: \* Solution n°1: On remplace ces données par des valeurs manquantes '0'. \* Solution n°2: On supprime certaines ou toutes les features nominales dont nous possédons des données manquantes. \* Solution n°3: On supprime les lignes contenant ces valeurs manquantes.

**Solution n°1:** Dans notre cas, les valeurs manquantes sont des données qui n'ont pas pu être remplies durant les rendez-vous. Ce manque de données ne doit pas être un poids dans la classification. Solution réfutée.

**Solution n°2:** Avant de décider si on supprime les features nominales, on doit connaître leur poids sur la classification. Essayons alors de déterminer leur corrélation.

#### Analyse du lien SES et classification

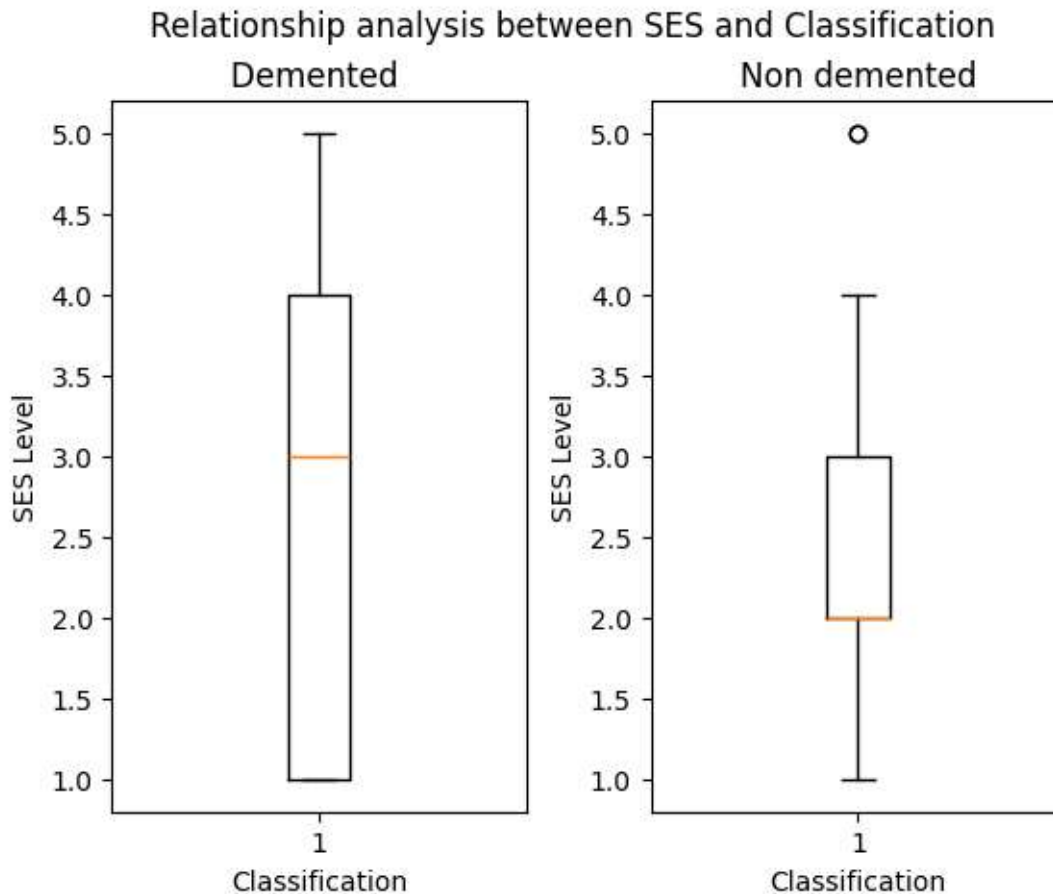
```
[20]: demented_data_SES = data_mri[data_mri['Group'] == 1]["SES"].dropna()
      nondemented_data_SES = data_mri[data_mri['Group'] == 0]["SES"].dropna()
```

```
[21]: fig, axs = plt.subplots(1, 2)
      axs[0].boxplot(demented_data_SES)
      axs[0].set_title('Demented ')
      axs[0].set_xlabel("Classification")
      axs[0].set_ylabel("SES Level")

      axs[1].boxplot(nondemented_data_SES)
      axs[1].set_title('Non demented')
      axs[1].set_xlabel("Classification")
      axs[1].set_ylabel("SES Level")

      fig.subplots_adjust(hspace=0.4, wspace=0.3)
      fig.suptitle('Relationship analysis between SES and Classification')

      plt.show()
```



D'après les données du projet OASIS dans la classification dément, on peut observer que la moyenne de la valeur socio-économique des candidats semblerait être à 3/5. Cette moyenne pourrait donner sens à l'hypothèse que le statut socio-économique n'influe pas sur la probabilité d'être touché par la maladie d'Alzheimer.

Pour le statut non dément, on peut voir que la moyenne semble être approximativement autour d'un niveau 2 socio-économique. Peut-être manquerait-il des données de personnes non démentes de niveau socio-économique supérieur à 4.

Hypothèse: le niveau socio-économique ne semble pas influencer sur la classification.

#### Analyse du lien MMSE et classification

```
[22]: demented_data_MMSE = data_mri[data_mri['Group'] == 1]["MMSE"].dropna()
      nondemented_data_MMSE = data_mri[data_mri['Group'] == 0]["MMSE"].dropna()
```

```
[23]: fig, axs = plt.subplots(1, 2)
      axs[0].boxplot(demented_data_MMSE)
      axs[0].set_title('Demented ')
      axs[0].set_xlabel("Classification")
```

```

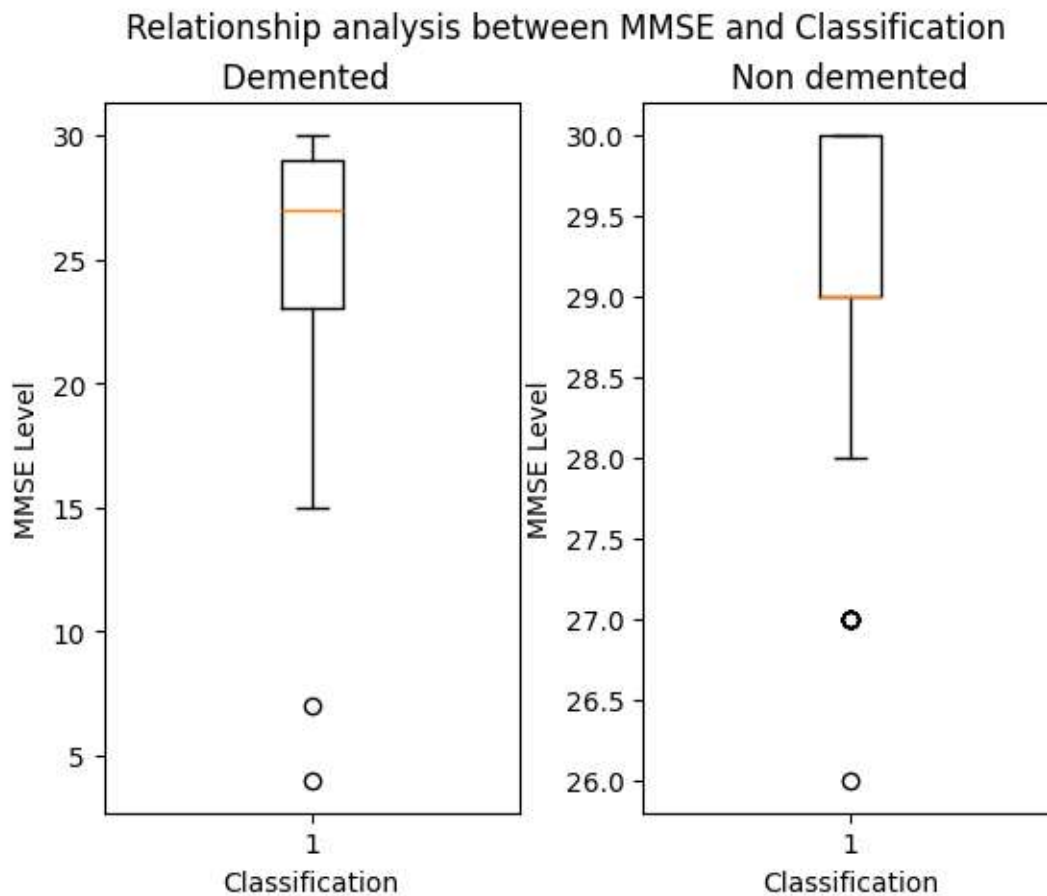
axs[0].set_ylabel("MMSE Level")

axs[1].boxplot(nondemented_data_MMSE)
axs[1].set_title('Non demented')
axs[1].set_xlabel("Classification")
axs[1].set_ylabel("MMSE Level")

fig.subplots_adjust(hspace=0.4, wspace=0.3)
fig.suptitle('Relationship analysis between MMSE and Classification')

plt.show()

```



D'après les données du projet OASIS, il semblerait que la grande majorité des données de sujets déments se place dès lors qu'un niveau MMSE obtenu est plus haut que 15. On peut émettre l'hypothèse que le projet OASIS n'ait pas pu récolter beaucoup de données de sujets au stage sévère.

Un MMSE sévère ( $<10$ ) veut-il forcément dire qu'une personne est démente?

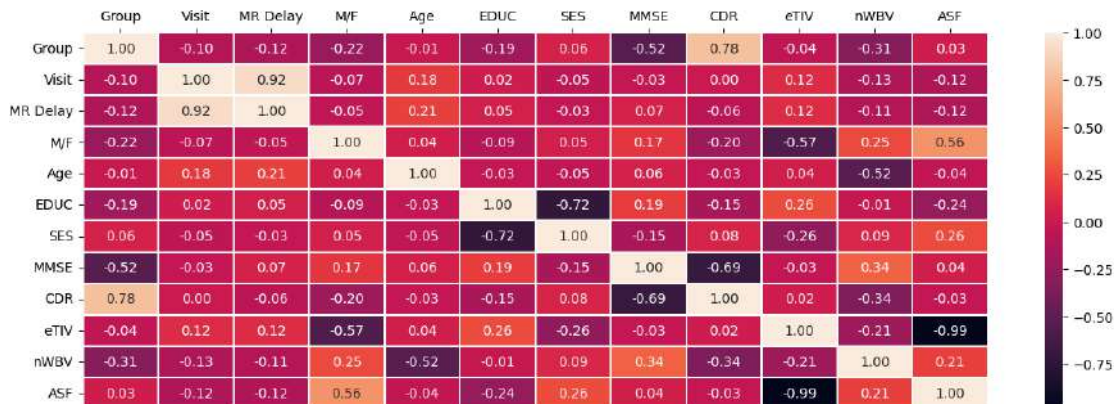
```
[24]: data_mri[(data_mri['Group'] == 'Nondemented') & (data_mri['MMSE']<10.0)].count()
```

```
[24]: Group      0
      Visit      0
      MR Delay    0
      M/F         0
      Age         0
      EDUC        0
      SES         0
      MMSE        0
      CDR         0
      eTIV        0
      nWBV        0
      ASF         0
      dtype: int64
```

D'après les données du projet OASIS, aucune personne ne semble pas être démente et avoir un MMSE sévère. Le MMSE serait alors possiblement une valeur importante pour définir la probabilité qu'un individu soit touché par la maladie d'Alzheimer.

### Matrice de corrélation

```
[25]: data_mri_corr = data_mri.corr()
      plt.subplots(figsize=(15,5))
      ax = sb.heatmap(data_mri_corr,annot=True, fmt='.2f', linewidth=.20)
      ax.xaxis.tick_top()
      plt.show()
```



On observe alors, avec la table de corrélation: - Un MMSE qui tend vers un score élevé semble influencer la non démente. Cette corrélation est cohérente puisque plus un MMSE est élevé plus l'état mental du sujet est stable. - Un CDR qui tend vers un score élevé semble influencer la démente. Ce résultat semble lui aussi cohérent puisque plus la note élevée, plus le patient possède de troubles mentaux.

Concernant nos précédentes variables nominatives: - Concernant la note SES, celle-ci semble très

peu influencer sur la classification du patient.

Etant donné que le MMSE semble complètement influencer la classification, nous gardons cette feature.

Pour le SES, nous comparerons les résultats prédictifs des algorithmes afin de choisir si l'on supprime la feature ou non.

### Suppression des données manquantes dans le MMSE

```
[26]: data_mri["MMSE"].isna().sum()
```

```
[26]: 2
```

```
[27]: data_mri["MMSE"].count()
```

```
[27]: 371
```

Nous avons deux sujets qui semblent ne pas avoir de score MMSE. Si nous leur attribuons un -1 pour les mettre de côté par rapport aux autres scores MMSE, nous risquons d'influencer les décisions de l'algorithme de classification qui faire en sorte que le -1 soit une note participant à la classification.

Etant donné le peu de données non connues en MMSE, nous supprimons ces deux sujets des données.

```
[28]: data_mri.dropna(subset=['MMSE'], inplace=True)
```

```
[29]: data_mri["MMSE"].isna().sum()
```

```
[29]: 0
```

```
[30]: data_mri.isnull().sum()
```

```
[30]: Group          0
      Visit        0
      MR Delay     0
      M/F          0
      Age          0
      EDUC         0
      SES         17
      MMSE         0
      CDR          0
      eTIV         0
      nWBV         0
      ASF          0
      dtype: int64
```

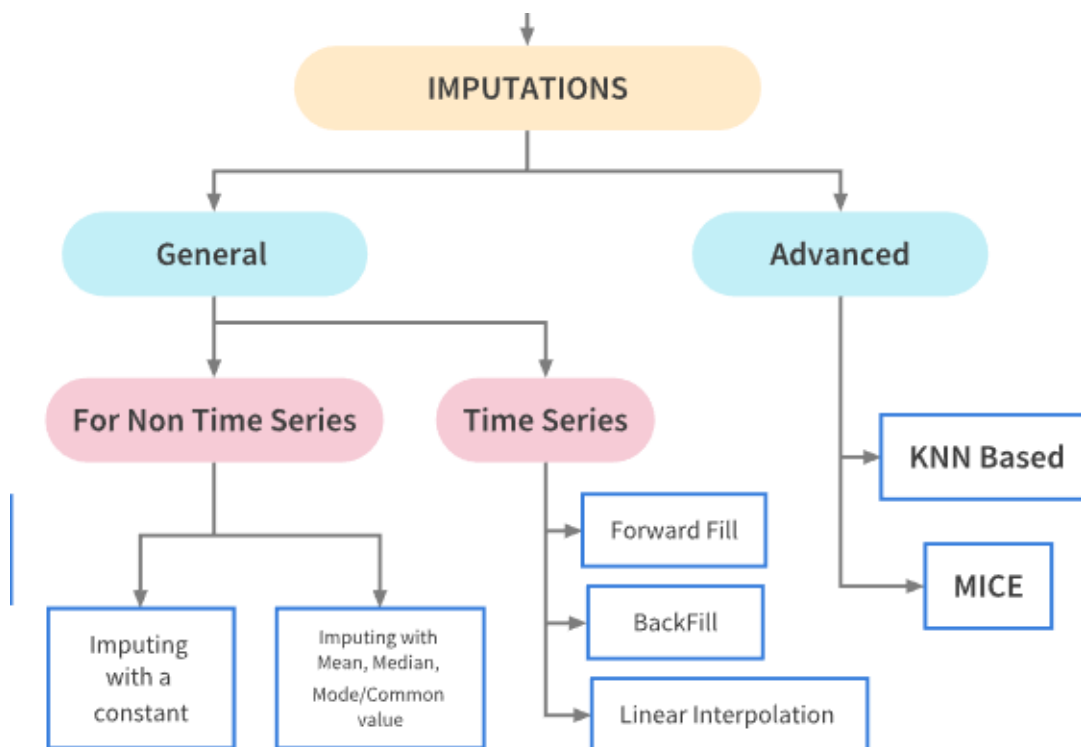
Nous remarquons qu'il y a encore 17 données manquantes dans le SES.

```
[31]: 17/data_mri['SES'].count()
```

[31]: 0.0480225988700565

Supprimer ces données nous ferait perdre 5% de nos données. On ne peut pas se permettre cela. Nous devons alors trouver un autre moyen de régler ces données manquantes.

Si on suit par exemple ce graphique [source: Parul Pandey](#) , nous devrions affecter la moyenne ou la médiane aux données manquantes. Cependant, cette affectation pourrait influencer la précision du modèle. Or, ces données manquantes ne devraient pas être un poids dans la classification. Nous utiliserons alors l'algorithme de KNN (K-Nearest Neighbor) pour faire en sorte que les données manquantes soient complétées en fonction de leurs voisins les plus proches. Cela permettrait ainsi de ne pas modifier (ou très très peu) la précision. Tout en réglant le problème des données.



```
[32]: data_mri['SES'].describe()
```

```
[32]: count    354.000000
      mean      2.460452
      std       1.134005
      min       1.000000
      25%       2.000000
      50%       2.000000
      75%       3.000000
      max       5.000000
      Name: SES, dtype: float64
```

```
[33]: #Algorithme KNN pour remplacer les données manquantes par leurs voisins les
      ↪ plus proches.
```

```
from sklearn.impute import KNNImputer

knn_imputer = KNNImputer(n_neighbors=2, weights="uniform")
data_mri['SES'] = knn_imputer.fit_transform(data_mri[['SES']])
```

```
[34]: data_mri['SES'].isnull().sum()
```

```
[34]: 0
```

```
[35]: data_mri['SES'].describe()
```

```
[35]: count      371.000000
      mean        2.460452
      std         1.107647
      min         1.000000
      25%         2.000000
      50%         2.000000
      75%         3.000000
      max         5.000000
      Name: SES, dtype: float64
```

Ainsi, nous remarquons que la moyenne n'a pas changé, il semblerait donc que le remplacement par l'algorithme KNN ait été un bon choix. Nous sommes dorénavant prêts pour l'entraînement.

## 2.5 Entraînement de la classification avec plusieurs algorithmes

Pour un entraînement de classification, nous utiliserons les algorithmes suivants:

- Logistic Regression
- KNN Classifier
- SVM
- Decision Tree
- Random Forest Classifier
- Gradient Classifier

### 2.5.1 Séparation des données

Afin que le score de précision ne soit pas truqué, nous allons séparer les données en deux données: une d'entraînement et une autre pour tester l'algorithme.

```
[36]: from sklearn.model_selection import train_test_split

Y = data_mri['Group']
X = data_mri.drop('Group', axis=1)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
                                                    random_state=42)
```

### 2.5.2 Mesure de précision

Avant d'entrer dans l'implémentation de ces algorithmes, nous allons définir une fonction de précision qui nous donnera le taux de précision de l'entraînement et des tests.

```
[37]: from sklearn.metrics import accuracy_score, confusion_matrix, \
      ↪ classification_report

def algorithm_result(classifier, isTest=True):
    prediction = None
    accuracy = None
    confusion = None
    if isTest:
        print("Test results:")
        prediction = classifier.predict(X_test)
        accuracy = accuracy_score(Y_test, prediction)
        confusion = confusion_matrix(Y_test, prediction)
    else:
        print("Train results:")
        prediction = classifier.predict(X_train)
        accuracy = accuracy_score(Y_train, prediction)
        confusion = confusion_matrix(Y_train, prediction)
    print("Accuracy:")
    print(accuracy)
    print("Confusion matrix:")
    print(confusion)
    print("=====")
```

### 2.5.3 Entraînement et tests

#### Logistic Regression

```
[38]: from sklearn.linear_model import LogisticRegression

logistic_clf = LogisticRegression(solver='liblinear')
```

Algorithm	Train Accuracy	Test Accuracy
Logistic Clf	.95	.902

#### KNN Classifier

```
[39]: from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, Y_train)

algorithm_result(knn_clf, isTest=False)
algorithm_result(knn_clf)
```

Train results:  
 Accuracy:  
 0.6911196911196911  
 Confusion matrix:  
 [[101 37]  
 [ 43 78]]

=====  
 Test results:  
 Accuracy:  
 0.5178571428571429  
 Confusion matrix:  
 [[29 23]  
 [31 29]]

Algorithm	Train Accuracy	Test Accuracy
Logistic Clf	.95	.902
KNN Clf	.69	.518

### Support Vector Classifier

```
[40]: from sklearn.pipeline import make_pipeline
      from sklearn.preprocessing import StandardScaler
      from sklearn.svm import SVC
      svc_clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
      svc_clf.fit(X_train, Y_train)

      algorithm_result(svc_clf, isTest=False)
      algorithm_result(svc_clf)
```

Train results:  
 Accuracy:  
 0.9575289575289575  
 Confusion matrix:  
 [[137 1]  
 [ 10 111]]

=====  
 Test results:  
 Accuracy:  
 0.9196428571428571  
 Confusion matrix:  
 [[51 1]  
 [ 8 52]]

Algorithm	Train Accuracy	Test Accuracy
Logistic Clf	.95	.902

Algorithm	Train Accuracy	Test Accuracy
KNN Clf	.69	.518
Support Vector Clf	.96	.920

### Decision Tree Classifier

```
[41]: from sklearn.tree import DecisionTreeClassifier
```

```
decision_clf = DecisionTreeClassifier()
decision_clf.fit(X_train, Y_train)

algorithm_result(decision_clf, isTest=False)
algorithm_result(decision_clf)
```

Train results:

Accuracy:

1.0

Confusion matrix:

```
[[138  0]
 [  0 121]]
```

=====

Test results:

Accuracy:

0.8928571428571429

Confusion matrix:

```
[[45  7]
 [ 5 55]]
```

=====

Algorithm	Train Accuracy	Test Accuracy
Logistic Clf	.95	.902
KNN Clf	.69	.518
Support Vector Clf	.96	.920
Decision Tree Clf	1	.902

### Random Forest Classifier

```
[42]: from sklearn.ensemble import RandomForestClassifier
```

```
forest_clf = RandomForestClassifier()
forest_clf.fit(X_train, Y_train)

algorithm_result(forest_clf, isTest=False)
algorithm_result(forest_clf)
```

Train results:

Accuracy:

1.0

Confusion matrix:

```
[[138  0]
 [  0 121]]
```

Test results:

Accuracy:

0.9196428571428571

Confusion matrix:

```
[[51  1]
 [ 8 52]]
```

Algorithm	Train Accuracy	Test Accuracy
Logistic Clf	.95	.902
KNN Clf	.69	.518
Support Vector Clf	.96	.920
Decision Tree Clf	1	.902
Random Forest Clf	1	.911

### Gradient Boosting Classifier

```
[43]: from sklearn.ensemble import GradientBoostingClassifier
      gradient_clf = GradientBoostingClassifier()
      gradient_clf.fit(X_train, Y_train)

      algorithm_result(gradient_clf, isTest=False)
      algorithm_result(gradient_clf)
```

Train results:

Accuracy:

1.0

Confusion matrix:

```
[[138  0]
 [  0 121]]
```

Test results:

Accuracy:

0.9017857142857143

Confusion matrix:

```
[[49  3]
 [ 8 52]]
```

Algorithm	Train Accuracy	Test Accuracy
Logistic Clf	.95	.902
KNN Clf	.69	.518
Support Vector Clf	.96	.920

Algorithm	Train Accuracy	Test Accuracy
Decision Tree Clf	1	.902
Random Forest Clf	1	.911
Gradient Boosting Clf	1	.902

### 2.5.4 Amélioration des algorithmes

Certains algorithmes nécessitent un ajustement des hyper-paramètres. Trouvons les meilleurs hyper-paramètres.

```
[44]: grid_X_train, grid_X_test, grid_Y_train, grid_Y_test = train\_test\_split(X_train, Y_train, test_size=0.5, random_state=42)
      from sklearn.model\_selection import GridSearchCV
      from sklearn.pipeline import Pipeline
```

#### Logistic Regression

```
[45]: params = {"clf__max_iter": (50,100,500)}

pipeline = Pipeline([( "scaler" , StandardScaler()),
                      ("clf",LogisticRegression(solver='liblinear'))])

logistic_clf = GridSearchCV(pipeline, params, cv=5)
logistic_clf = logistic_clf.fit(X_train, Y_train)

algorithm_result(logistic_clf, isTest=False)
algorithm_result(logistic_clf)

print(f"Best params: {logistic_clf.best_params_}")
```

Train results:

Accuracy:

0.9575289575289575

Confusion matrix:

```
[[137  1]
 [ 10 111]]
```

=====

Test results:

Accuracy:

0.9196428571428571

Confusion matrix:

```
[[51  1]
 [ 8 52]]
```

=====

Best params: {'clf\_\_max\_iter': 50}

Algorithm	Train Accuracy	Test Accuracy
Logistic Clf	.96	.919

### KNN Classifier

```
[46]: params = {"clf__leaf_size": (30,50,60),
               "clf__n_neighbors": (5,10,20)}

pipeline = Pipeline([("scaler", StandardScaler()),
                     ("clf", KNeighborsClassifier())])

knn_clf = GridSearchCV(pipeline, params, cv=5)
knn_clf = knn_clf.fit(X_train, Y_train)

algorithm_result(knn_clf, isTest=False)
algorithm_result(knn_clf)

print(f"Best params: {knn_clf.best_params_}")
```

Train results:

Accuracy:

0.9266409266409267

Confusion matrix:

```
[[138  0]
 [ 19 102]]
```

Test results:

Accuracy:

0.8571428571428571

Confusion matrix:

```
[[52  0]
 [16 44]]
```

Best params: {'clf\_\_leaf\_size': 30, 'clf\_\_n\_neighbors': 10}

Algorithm	Train Accuracy	Test Accuracy
Logistic Clf	.96	.919
KNN Clf	.93	.871

### Support Vector Classifier

```
[47]: params = {"clf__C": (0.1, 0.5, 1, 2, 5, 10, 20),
               "clf__gamma": (0.001, 0.01, 0.1, 0.25, 0.5, 0.75, 1)}

pipeline = Pipeline([("scaler", StandardScaler()),
                     ("clf", SVC(gamma='auto'))])

svc_clf = GridSearchCV(pipeline, params, cv=5)
```

```

svc_clf.fit(X_train, Y_train)

algorithm_result(svc_clf, isTest=False)
algorithm_result(svc_clf)

print(f"Best params: {svc_clf.best_params_}")

```

Train results:

Accuracy:

0.9575289575289575

Confusion matrix:

```

[[137   1]
 [ 10 111]]

```

Test results:

Accuracy:

0.9196428571428571

Confusion matrix:

```

[[51   1]
 [ 8  52]]

```

Best params: {'clf\_\_C': 0.5, 'clf\_\_gamma': 0.1}

Algorithm	Train Accuracy	Test Accuracy
Logistic Clf	.96	.919
KNN Clf	.93	.871
Support Vector Clf	.96	.919

## Decision Tree Classifier

```

[48]: params = {"clf__splitter":("best", "random"),
               "clf__max_depth":(list(range(1, 20))),
               "clf__min_samples_split":[2, 3, 4],
               "clf__min_samples_leaf":list(range(1, 20))}

pipeline = Pipeline([( "scaler" , StandardScaler()),
                     ("clf",DecisionTreeClassifier())])

decision_clf = GridSearchCV(pipeline, params, cv=5)
decision_clf.fit(X_train, Y_train)

algorithm_result(decision_clf, isTest=False)
algorithm_result(decision_clf)

print(f"Best params: {decision_clf.best_params_}")

```

Train results:

Accuracy:  
0.9343629343629344  
Confusion matrix:  
[[136 2]  
[ 15 106]]

=====

Test results:

Accuracy:  
0.9017857142857143  
Confusion matrix:  
[[49 3]  
[ 8 52]]

=====

Best params: {'clf\_\_max\_depth': 8, 'clf\_\_min\_samples\_leaf': 5,  
'clf\_\_min\_samples\_split': 4, 'clf\_\_splitter': 'random'}

Algorithm	Train Accuracy	Test Accuracy
Logistic Clf	.96	.919
KNN Clf	.93	.871
Support Vector Clf	.96	.919
Decision Tree Clf	.96	.919

## Random Forest Classifier

```
[49]: params = {
    'clf__n_estimators': [100, 500, 1000],
    'clf__max_depth': [2, 3, 5, 10, 15, None]}

pipeline = Pipeline([("scaler", StandardScaler()),
                      ("clf", RandomForestClassifier())])

forest_clf = GridSearchCV(pipeline, params, cv=5)
forest_clf.fit(X_train, Y_train)

algorithm_result(forest_clf, isTest=False)
algorithm_result(forest_clf)

print(f"Best params: {forest_clf.best_params_}")
```

Train results:  
Accuracy:  
0.9575289575289575  
Confusion matrix:  
[[137 1]  
[ 10 111]]

=====

Test results:

Accuracy:  
0.9196428571428571  
Confusion matrix:  
[[51 1]  
[ 8 52]]

=====

Best params: {'clf\_\_max\_depth': 2, 'clf\_\_n\_estimators': 100}

Algorithm	Train Accuracy	Test Accuracy
Logistic Clf	.96	.919
KNN Clf	.93	.871
Support Vector Clf	.96	.919
Decision Tree Clf	.96	.919
Random Forest Clf	0.96	.919

### Gradient Boosting Classifier

```
[50]: params = {
    'clf__learning_rate': [0.1, 0.2, 0.3, 0.4, 0.5],
    'clf__n_estimators': [100,500,1000],
    'clf__min_samples_split': [2, 5, 10],
    'clf__min_samples_leaf': [1, 2, 4]}

pipeline = Pipeline([( "scaler" , StandardScaler()),
                      ("clf",GradientBoostingClassifier())])

gradient_clf = GridSearchCV(pipeline, params, cv=5)
gradient_clf.fit(X_train, Y_train)

algorithm_result(gradient_clf, isTest=False)
algorithm_result(gradient_clf)

print(f"Best params: {gradient_clf.best_params_}")
```

Train results:  
Accuracy:  
1.0  
Confusion matrix:  
[[138 0]  
[ 0 121]]

=====

Test results:  
Accuracy:  
0.9017857142857143  
Confusion matrix:  
[[49 3]  
[ 8 52]]

=====

Best params: {'clf\_\_learning\_rate': 0.1, 'clf\_\_min\_samples\_leaf': 1, 'clf\_\_min\_samples\_split': 10, 'clf\_\_n\_estimators': 100}

Algorithm	Train Accuracy	Test Accuracy
Logistic Clf	.96	.919
KNN Clf	.93	.871
Support Vector Clf	.96	.919
Decision Tree Clf	.96	.919
Random Forest Clf	0.96	.919
Gradient Boosting Clf	1	.902

### 2.5.5 Conclusions

Comparons les résultats avant optimisation des hyper-paramètres et normalisation

Algorithm	Train Accuracy	Test Accuracy
Logistic Clf	.95	.902
KNN Clf	.69	.518
Support Vector Clf	.96	.920
Decision Tree Clf	1	.902
Random Forest Clf	1	.911
Gradient Boosting Clf	1	.902

A ceux optimisés:

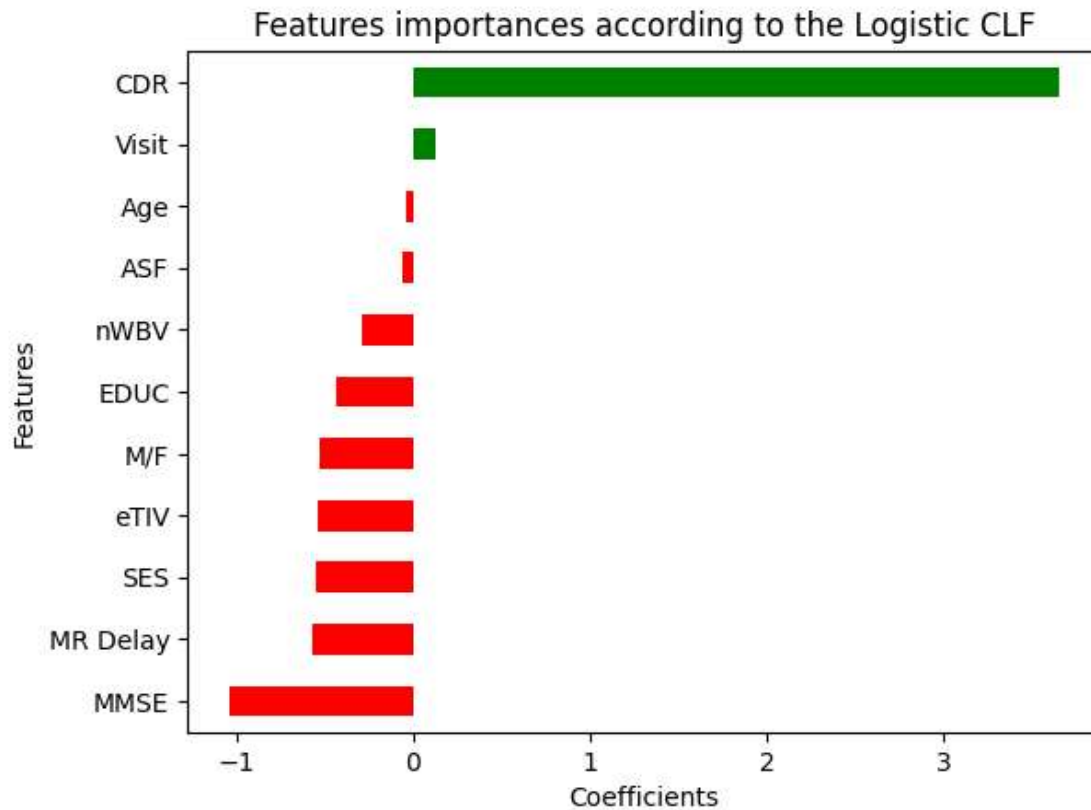
Algorithm	Train Accuracy	Test Accuracy
Logistic Clf	.96	.919
KNN Clf	.93	.871
Support Vector Clf	.96	.919
Decision Tree Clf	.96	.919
Random Forest Clf	0.96	.919
Gradient Boosting Clf	1	.902

On peut observer une grande amélioration de l'algorithme KNN après normalisation et c'est logique: l'algorithme kNN cherche à calculer la distance entre des points. Si jamais il y a un point qui est trop éloigné des autres alors l'algorithme pourrait comprendre que l'étendu général des distances est beaucoup trop grand. Ainsi, nous pouvons par exemple choisir l'algorithme logistique classifier qui est rapide et qui possède une précision de 0.919.

```
[51]: logistic_coefs = pd.Series(data=logistic_clf.best_estimator_.named_steps.clf.
    ↪coef_[0], index=data_mri.drop('Group',axis=1).columns)

logistic_coefs.sort_values(ascending=True, inplace=True)
col = ['red' if c < 0 else 'green' for c in logistic_coefs]
```

```
logistic_coefs.plot(kind="barh", color=col)
plt.title("Features importances according to the Logistic CLF")
plt.xlabel("Coefficients")
plt.ylabel("Features")
plt.show()
```



D'après le diagramme en barre ci-dessus, la note de CDR (note clinique de démence) semble être le facteur le plus important pour qu'un patient tende à être non dément. Son nombre de visites semble aussi légèrement influencer à tendre vers une non démence. On peut aussi voir que le facteur le plus important pour qu'un patient ait plus de probabilité d'être dément semble être la note MMSE (note de l'examen mental du patient). Derrière, nous avons le délai d'IRM qui, plus le délai est long, tend à favoriser la démence du patient. Nous avons aussi le facteur socio-économique qui semble être important. Plus le facteur est grand, plus le risque de démence est important. Cette interprétation est possiblement à remettre en question puisque les données ont été récoltées à partir de patients qui ont accepté de partager leurs données mais aussi, qui ont pu investir dans une lutte contre la maladie. Cette condition financière pourrait être la cause d'un manque de patients déments à bas niveau socio-économique.

[ ]: