Master 2 Data Science et Société Numérique

MACHINE LEARNING 2

Professeur : BERCHER Jean-François

Prédire la note sur dossier des élèves pour un concours d'entrée à un groupe d'écoles d'ingénieurs

PHAM Dong Pha - M2 D2SN

Problématique

Dans ce travail, nous allons prédire la note de dossier obtenue par un candidat. Celle-ci est calculée dans le cadre d'un concours d'entrée à un groupe d'écoles d'ingénieurs. Le but est de prédire la note sur dossier en fonction de l'ensemble des données fournies. Au-delà de cela, il s'agit de comprendre quels sont les paramètres déterminants dans l'évaluation du dossier, voire d'approcher ou retrouver l'algorithme. Les données ont été complètement anonymisées et les différentes notes décalées de quantités aléatoires. Le but est donc, à partir de l'ensemble des données disponibles, de prédire la note de dossier obtenue par un candidat.

Nous choisissons les tâches principales du projet:

- Lire le jeu de données disponible

- Prétraiter le jeu de données (Imputation, suppression des colonnes indésirables sans affecter la précision, traitement des valeurs aberrantes, …etc.)

- Visualiser le jeu de données

- Effectuer une analyse exploratoire des données (statistiques descriptives, corrélation, sélection de caractéristiques, réduction de dimensionnalité, …etc)

- Utiliser 2 algorithmes d'apprentissage automatique différents pour prédire la note

- Après avoir identifié le meilleur algorithme, utiliser toutes les données d'entraînement (train.csv) pour créer le modèle et prédire la note finale de l'examen pour les données de test fournies (test.csv)

Table des matières

I) Exploratory Data Analysis (EDA)

1) Identification des variables

D'abord, nous identifions variable de prévision (Input) et variable cible (Output). Ensuite, nous identifions le type de données et la catégorie des variables. En effet, vu qu'il y a des 246 variables, il est donc trop long à lister dans le tableau, nous utilisons les exemples pour chaque élément pour illustrer, mais la variable cible est unique!

Nous voulons prédire les scores des élèves sur la base de toutes les données fournies. Ici, nous devons identifier les variables prédictives, la variable cible, le type de données des variables et la catégorie de variables. Et voici ce que nous pouvons proposer :

Type de variables	Type de données	Catégorie de variable
Variables prédictives - Genre - Bac - Pondération - Méthode de travail - Moyenne candidat en Mathématiques Trimestre 1 Terminale - …. etc, etc	**Caractère** - Genre - Bac - Avis sur la capacité à réussir - Niveau de la classe - Méthode de travail - …etc, etc	**Catégorique** - Genre - Pondération - Méthode de travail - Autonomie - …etc, etc
Variable cible - Points	**Numérique** - Moyenne candidat en Mathématiques Trimestre 1 Terminale - Note à l'épreuve de Oral de Français - …etc, etc	**Continue** - Points - Note à l'épreuve de Oral de Français (épreuve anticipée) - Note à l'épreuve de Ecrit de Français (épreuve anticipée) - Moyenne candidat en Mathématiques Trimestre 1 Terminale - ...etc, etc

La première étape pour résoudre un problème de science des données est l'analyse exploratoire des données (EDA). Il s'agit d'un processus ouvert dans lequel nous recherchons des anomalies, des tendances ou des modèles intéressants et des corrélations dans un ensemble de données. Ceux-ci peuvent être intéressants en eux-mêmes et ils peuvent éclairer notre modélisation. En gros, nous utilisons l'EDA pour découvrir les informations que nos données peuvent nous révéler.

Tout d'abord, nous examinons les données en tant que dataframe pandas et nous supprimons les points-virgules (;) dans les lignes et les colonnes.

```
df=pd.read_csv('/content/train.csv', sep=';')
df.head()
```

	id	Bac	Nom	Prénom	Genre	Aménagements Validés	Pondération	Points	Méthode de travail	Autonomie	Engagement, Esprit d'initiative	Capacité à s'investir	Niveau de la classe	Cohéren projet formati
0	0	scientific	LECLERC	Bernadette	F	NaN	1	111	Très satisfaisante	Très satisfaisante	Non	Très satisfaisante	Très bon	Na
1	1	scientific	TEXIER	Rémy	H	NaN	0	78	Très satisfaisante	Très satisfaisante	Non	Très satisfaisante	Très bon	Na
2	2	scientific	BERNIER	Olivier	H	NaN	0	98	Satisfaisante	Satisfaisante	Oui	Assez satisfaisante	Assez bon	Na
3	3	scientific	PASCAL	Isaac	H	NaN	0	93	Satisfaisante	Satisfaisante	Oui	Très satisfaisante	Bon	Na
4	4	scientific	PAUL	Arnaude	F	NaN	0	105	Très satisfaisante	Très satisfaisante	Non	Très satisfaisante	Bon	Na

L'une des caractéristiques de cet ensemble de données est qu'il est multivarié. L'ensemble de données multivarié composé de deux ou plusieurs variables est un ensemble multivarié. Il est supervisé car nous avons un ensemble de données d'entraînement avec des objectifs connus et, pendant l'entraînement, nous voulons que notre modèle apprenne à prédire la note à partir des autres variables.
Dans ce dataset (train.csv), il y a un total de 2423 observations avec 246 variables. Chaque ligne correspond à un élève, chaque colonne contenant une caractéristique de différence. La colonne "Points" est notre variable cible (également appelée réponse), ce qui en fait une tâche d'apprentissage automatique de régression supervisée. Il est supervisé car nous avons un ensemble de données d'entraînement avec des objectifs connus et, pendant l'entraînement, nous voulons que notre modèle apprenne à prédire la note à partir des autres variables. Nous traiterons les "Points" comme continus, ce qui en fait un problème de régression.

2) Nettoyage du dataframe :

- Nous éliminons les colonnes inutiles: "Prénom" et "Nom".

```
#Eliminons les colonnes inutiles
df.drop(columns=['Nom','Prénom'],axis=1,inplace=True)
df.head()
```

	id	Bac	Genre	Aménagements Validés	Pondération	Points	Méthode de travail	Autonomie	Engagement, Esprit d'initiative	Capacité à s'investir	Niveau de la classe	Cohérence projet de formation	Av... is sur la ...cité à ...ussir	l'... de ... F... (... ant...
0	0	scientific	F	NaN	1	111	Très satisfaisante	Très satisfaisante	Non	Très satisfaisante	Très bon	NaN	Satisf...aisante	
1	1	scientific	H	NaN	0	78	Très satisfaisante	Très satisfaisante	Non	Très satisfaisante	Très bon	NaN	Assez satisf...aisante	
2	2	scientific	H	NaN	0	98	Satisfaisante	Satisfaisante	Oui	Assez satisfaisante	Assez bon	NaN	Assez satisf...aisante	
3	3	scientific	H	NaN	0	93	Satisfaisante	Satisfaisante	Oui	Très satisfaisante	Bon	NaN	Satisf...aisante	
4	4	scientific	F	NaN	0	105	Très satisfaisante	Très satisfaisante	Non	Très satisfaisante	Bon	NaN	Très satisf...aisante	

5 rows × 244 columns

- Nous éliminons les colonnes dont les NaNs occupent plus de 50 % parce que l'information contenue dans la variable n'est pas élevée, nous supprimons la variable si elle a plus de 50% de valeurs manquantes. Dans cette méthode, nous supprimons des colonnes avec des valeurs nulles au-dessus d'un certain seuil.

```
#Eliminons le colonnes dont les NaNs occupent plus de 50 %
print('Avant le nettoyage' + str(df.shape))
features=df.columns.values
for feature in features :
  nan=df[feature].isnull().sum()
  if nan > 1300 :
    df.drop(columns=[feature],axis=1,inplace=True)
print('Après le nettoyage :'+ str(df.shape))
```

```
Avant le nettoyage(2423, 244)
Après le nettoyage :(2423, 73)
```

Après l'étape de nettoyage, nous avons un dataset avec 2423 observations et 73 variables.

- Nous poussons vers la droite le dtype des colonnes de "objet" au "float" pour examiner, calculer la moyenne et le tranning model, …etc, ensuite, nous convertissons les NaN en moyenne dans chaque colonne.

```
#Poussons le dtype vers la droite et convertissons les nan en moyenne dans chaque colonne
new_features=df.columns.values
inapropriate_col=new_features[11:]
inapropriate_col=np.append(inapropriate_col,new_features[3])
for col in inapropriate_col :
    df[col]=df[col].str.replace(',','.')
    df[col]=df[col].replace(np.nan,'0')
    df[col]=df[col].astype(float)
    df[col]=df[col].replace(0.,np.mean(df[col]))
```

- Nous trouvons et comptons le nombre de valeurs manquantes dans toutes les colonnes en utilisant les fonctions: isnull(), sum().

```
#Comptons les Nans dans le dataset
df.isnull().sum()
```

```
id                                                          0
Bac                                                         0
Genre                                                       0
Pondération                                                 0
Points                                                      0
                                                           ..
Moyenne classe en Langue vivante 1 Trimestre 3 Première     0
Moyenne candidat en Histoire/Géographie Trimestre 3 Première    0
Moyenne classe en Histoire/Géographie Trimestre 3 Première      0
Moyenne candidat en Français Trimestre 3 Première           0
Moyenne classe en Français Trimestre 3 Première             0
Length: 73, dtype: int64
```

Ici, toutes les variables sont bien représentées.

- Nous déterminons la catégorie de chaque variable (type de données : catégorie).

```
# Déterminons la categorie de chaque variable
print(df['Bac'].value_counts())
print(df['Genre'].value_counts())
print(df['Méthode de travail'].value_counts())
print(df['Autonomie'].value_counts())
print(df["Engagement, Esprit d'initiative"].value_counts())
print(df["Capacité à s'investir"].value_counts())
print(df["Niveau de la classe"].value_counts())
print(df['Avis sur la capacité à réussir'].value_counts())
```

```
scientific    2244
sti2d          179
Name: Bac, dtype: int64
H    2087
F     336
Name: Genre, dtype: int64
Satisfaisante          936
Très satisfaisante     864
Assez satisfaisante    459
Peu démontrée          107
Name: Méthode de travail, dtype: int64
Très satisfaisante     1183
Satisfaisante           790
Assez satisfaisante     312
Peu démontrée            81
Name: Autonomie, dtype: int64
Oui    1340
Non    1026
Name: Engagement, Esprit d'initiative, dtype: int64
Très satisfaisante     1237
Satisfaisante           726
Assez satisfaisante     316
Peu démontrée            87
Name: Capacité à s'investir, dtype: int64
Bon          941
Très bon     789
Assez bon    423
Moyen        174
Faible         6
Name: Niveau de la classe, dtype: int64
Très satisfaisante     1193
Satisfaisante           592
Assez satisfaisante     262
Peu démontrée           128
Name: Avis sur la capacité à réussir, dtype: int64
```

Nous voyons que ce sont des valeurs qui ne sont pas encodées et non-numériques, par exemple: ‘Scientific’, ‘sti2d’, ‘H’, ‘F’, ‘Satisfaisante’, ‘Très satisfaisante’, ‘Assez satisfaisante’, ‘Peu démontrée’. Nous devons donc faire un encodage numérique de ces valeurs afin que le modèle puisse comprendre et modéliser les entrées.

Ici, nous mettons:

‘H’ : 1

‘F’ : 0

‘Non’ : 0

‘Oui’ : 1

‘Scientific’ : 1

‘sti2d’ : 0

'Bon' : 1

'Assez bon' : 2

'Très bon' : 3

'Faible' : 4

'Moyen' : 5

‘Satisfaisante’ : 1

‘Très satisfaisante’ : 2

‘Assez satisfaisante’ : 3

‘Peu démontrée’ : 4

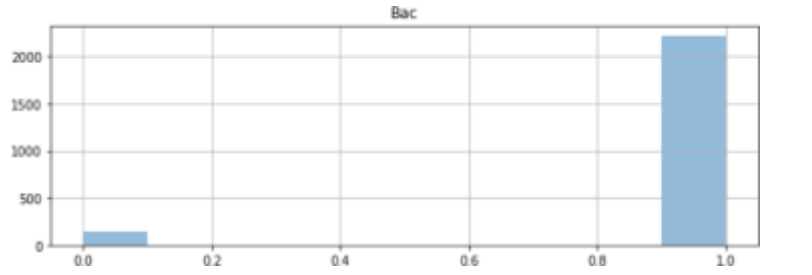
```
# Faisons un encodage numérique pour modéliser les entrées
df['Genre']=df['Genre'].map({'H':1,'F':0})
df['Bac']=df['Bac'].map({'scientific':1,'sti2d':0})
df['Méthode de travail']=df['Méthode de travail'].map({'Satisfaisante':1,'Très satisfaisante':2,'Assez satisfaisante':3,'Peu démontrée':4})
df['Avis sur la capacité à réussir']=df['Avis sur la capacité à réussir'].map({'Satisfaisante':1,'Très satisfaisante':2,'Assez satisfaisante':3,'Peu démontrée':4})
df['Niveau de la classe']=df['Niveau de la classe'].map({'Bon':1,'Assez bon':2,'Très bon':3,'Faible':4,'Moyen':5})
df["Engagement, Esprit d'initiative"]=df["Engagement, Esprit d'initiative"].map({'Non':0,'Oui':1})
df['Autonomie']=df['Autonomie'].map({'Satisfaisante':1,'Très satisfaisante':2,'Assez satisfaisante':3,'Peu démontrée':4})
df["Capacité à s'investir"]=df["Capacité à s'investir"].map({'Satisfaisante':1,'Très satisfaisante':2,'Assez satisfaisante':3,'Peu démontrée':4})
```

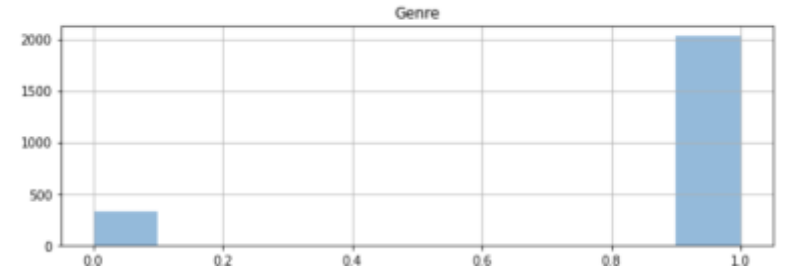
```
df.head()
```

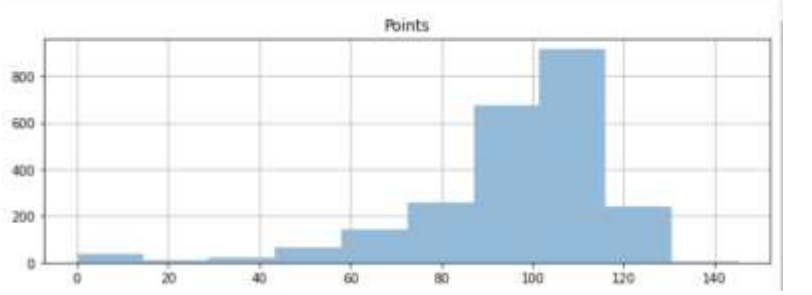
	id	Bac	Genre	Pondération	Points	Méthode de travail	Autonomie	Engagement, Esprit d'initiative	Capacité à s'investir	Niveau de la classe	Avis sur la capacité à réussir	Note à l'épreuve de Oral de Français (épreuve anticipée)	Note à l'épreuve de Ecrit de Français (épreuve anticipée)	Moyenne candidat en Mathématiques Trimestre 1 Terminale	Moyenne classe en Mathématiques Trimestre 1 Terminale	Moyenne candidat en Physique/Chimie Trimestre 1 Terminale	Moyenne c Physique/C Trimes Term
0	0	1	0	1.000000	111	2.0	2.0	0.0	2.0	3.0	1.0	11.0	11.0	11.38	9.78	11.05	
1	1	1	1	0.193562	78	2.0	2.0	0.0	2.0	3.0	3.0	13.0	13.0	8.68	13.06	7.75	
2	2	1	1	0.193562	98	1.0	1.0	1.0	3.0	2.0	3.0	10.0	9.0	8.90	11.18	11.17	
3	3	1	1	0.193562	93	1.0	1.0	1.0	2.0	1.0	1.0	13.0	8.0	12.18	12.36	8.10	
4	4	1	0	0.193562	105	2.0	2.0	0.0	2.0	1.0	2.0	13.0	11.0	14.28	11.49	10.96	

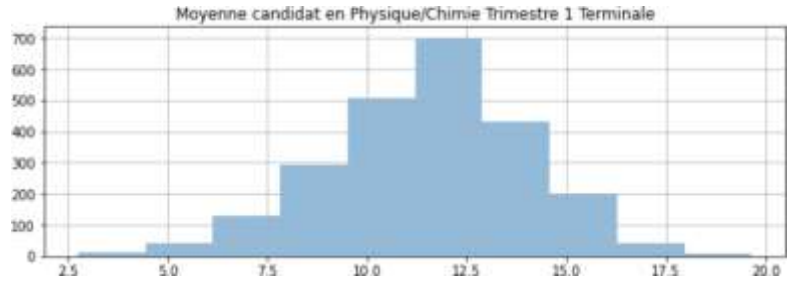
3) Visualisation du dataset

- Nous affichons des histogrammes des 20 premières caractéristiques (Nous n'en prenons que 5 comme exemple).

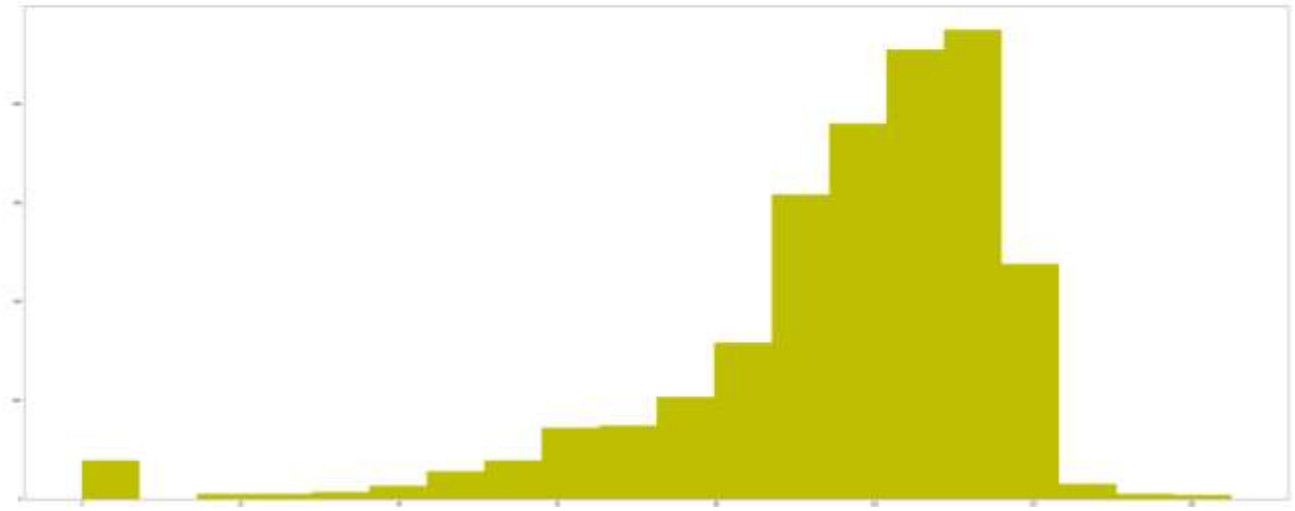









```
#Affichons la colonne "Points"
plt.hist(df['Points'],bins=20,color='y')
plt.show()
```



Par exemple, ici, nous voyons que le nombre de BAC (scientific) est majoritaire, au nombre de 2244 contre 179 (sti2d) et il y a un grand écart entre les sexes mâle et femelle (2087 contre 336). La majorité des élèves ont des scores allant de 90 au 120 (plus de 1400 étudiants).

À propos de la moyenne en Physique du trimestre 1 de terminale, il y avait plus de 700 élèves avec des scores de 11 à 12,5, environ 500 élèves avec des scores de 10 à 11 et plus de 400 élèves avec des scores entre 13 et 15.

Pour l'histogramme de la variable cible "Points", nous voyons que la majorité des notes sont comprises entre 60 et 130.

4) Analyse descriptive

Tout d'abord, nous examinons la fonction de description qui calcule les statistiques de base pour toutes les variables continues de l'ensemble de données.

La fonction de description nous montre:

- le nombre de cette variable

- la moyenne

- l'écart type

- les valeurs maximales et minimales

- l'IQR (intervalle interquartile: 25%, 50%, 75%)

Nous appliquons la méthode de description comme suit:

```
#Analyse descriptive EDA
df.describe(include='all')
```

	id	Bac	Genre	Pondération	Points	Méthode de travail	Autonomie	Engagement, Esprit d'initiative	Capacité à s'investir	Niveau de la classe	Avis sur la capacité à réussir
count	2423.000000	2423.000000	2423.000000	2423.000000	2423.000000	2366.000000	2366.000000	2366.000000	2366.000000	2333.000000	2175.000000
mean	1211.000000	0.926125	0.861329	0.354051	95.887742	1.888842	1.866441	0.566357	1.900254	2.163738	1.965977
std	699.604174	0.261622	0.345674	0.380165	21.708046	0.870898	0.764928	0.495682	0.760121	1.181884	0.791945
min	0.000000	0.000000	0.000000	0.193562	0.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000
25%	605.500000	1.000000	1.000000	0.193562	88.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000
50%	1211.000000	1.000000	1.000000	0.193562	101.000000	2.000000	2.000000	1.000000	2.000000	2.000000	2.000000
75%	1816.500000	1.000000	1.000000	0.193562	110.000000	2.000000	2.000000	1.000000	2.000000	3.000000	2.000000
max	2422.000000	1.000000	1.000000	1.500000	145.000000	4.000000	4.000000	1.000000	4.000000	5.000000	4.000000

-count: affiche la valeur de la variable, dans le tableau c'est 2423. Le nombre est utilisé pour afficher le nombre d'entrées que nous avons dans un ensemble de données.

-mean: la valeur moyenne de la variable.

-std: écart type, une quantité exprimant la différence entre les membres d'un groupe et la valeur moyenne du groupe. En termes simples, il s'agit d'une mesure de la répartition des nombres.

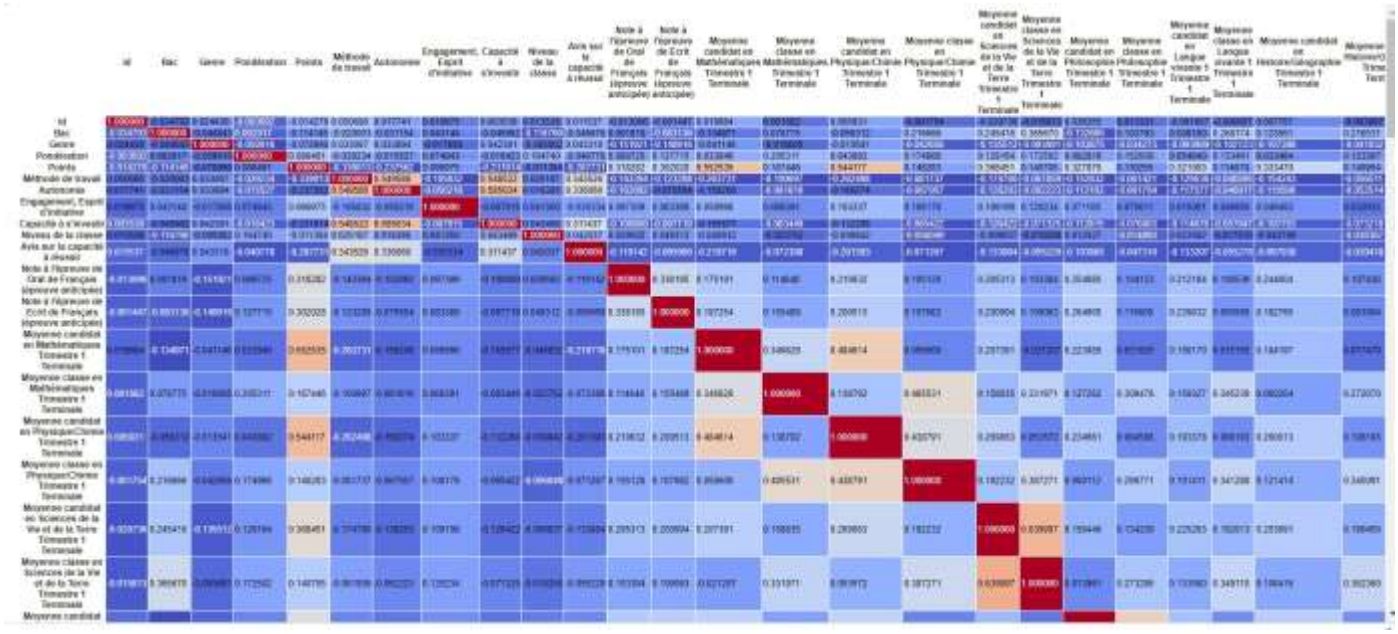
-min: valeur minimale de la variable

-IQR: L'intervalle interquartile nous montre la valeur de la variable aux percentiles respectifs, c'est-à-dire 25%, 50% et 75%.

-max: valeur maximale de la variable.

5) Matrice de corrélation

```
# Matrice de corrélation
cor = df.corr()
cor.style.background_gradient(cmap='coolwarm')
```



-Une matrice de corrélation est un tableau indiquant la valeur du coefficient de corrélation (les coefficients de corrélation sont utilisés dans les statistiques pour mesurer la force d'une relation entre deux variables.) entre des ensembles de variables. Chaque attribut de l'ensemble de données est comparé aux autres attributs pour connaître le coefficient de corrélation. Cette analyse nous permet de voir quelles paires ont la corrélation la plus élevée, les paires qui sont fortement corrélées représentent la même variance de l'ensemble de données, nous pouvons donc les analyser plus en détail pour comprendre quel attribut parmi les paires est le plus significatif pour la construction du modèle.

Ici, nous pouvons voir le réseau de corrélation de toutes les variables sélectionnées, la valeur de corrélation est comprise entre -1 et +1. Les variables hautement corrélées auront une valeur de corrélation proche de +1 et les variables moins corrélées auront une valeur de corrélation proche de -1.

- Montrons la moyenne des valeurs de toutes les colonnes, la médiane des valeurs de toutes les colonnes, la variance de toutes les colonnes, l'écart type de toutes les colonnes.

```
#la moyenne des valeurs de toutes les colonnes
df.mean()
```

```
id                                                          1Z11.000000
Bac                                                            0.9Z61Z5
Genre                                                          0.8613Z9
Pondération                                                    0.354051
Points                                                        95.887742

Moyenne classe en Langue vivante 1 Trimestre 3 Première       13.73Z949
Moyenne candidat en Histoire/Géographie Trimestre 3 Première  11.657363
Moyenne classe en Histoire/Géographie Trimestre 3 Première    12.147901
Moyenne candidat en Franpais Trimestre 3 Premiere             11.327923
Moyenne classe en Franpais Trimestre 3 Première               12.053865
Length: 73, dtype: float64
```

```
#la médiane des valeurs de toutes les colonnes
dfmedian)
```

```
id                                                          1Z11.000000
Bac                                                            1.000000
Genre                                                          1.000000
Pondération                                                    0.193562
Points                                                       101.000000

Moyenne classe en Langue vivante 1 Trimestre 3 Première       13.6Z0000
Moyenne candidat en Histoire/Géographie Trimestre 3 Première  11.560000
Moyenne classe en Histoire/Géographie Trimestre 3 Première    12.060000
Moyenne candidat en Franpais Trimestre 3 Premiere             11.260000
Moyenne classe en Franpais Trimestre 3 Première               12.090000
Length: 73, dtype: float64
```

```
#Ia variance de toutes les colonnes
df.var()
```

```
id                                                         489446.000000
Bac                                                            6. Osg 6
Genre                                                          6.119491
Pondération                                                    0.144526
Points                                                       471. 239251

Moyenne classe en Langue vivante 1 T imestre 3 Première       3.020237
Moyenne candidat en Histoire/Géographie Trimestre 3 Premiere  6.539311
Moyenne classe en Histoire/Géographie Trimestre 3 Première    2.Z51861
Moyenne candidat en *ranpais Trimestre 3 Premiere             5.3T3504
Moyenne classe en Franpais Trimestre s Première               1.817209
Length: 73, dtype: float64
```

```
#I'écart type de toutes les colonnes
df.std()
```

```
id                                                           699.6041T4
Bac                                                            0.261622
Genre                                                          6.345674
Pondération                                                    0.380165
Points                                                        Z1.7080d6

Moyenne classe en Langue vivante 1 T imestre 3 Première       1.737883
Moyenne candidat en Histoire/Géographie Trimestre 3 Premiere  2.557z0g
Moyenne classe en Histoire/Géographie Trimestre 3 Première    1.500620
Moyenne candidat en *ranpais Trimestre 3 Premiere             2.318082
Moyenne classe en Franpais Trimestre 3 Première               1.348039
Length: 73, dtype: float64
```

6) La fonction de corrélation la plus élevée

Trouvons la fonction de corrélation la plus élevée et sa colonne.

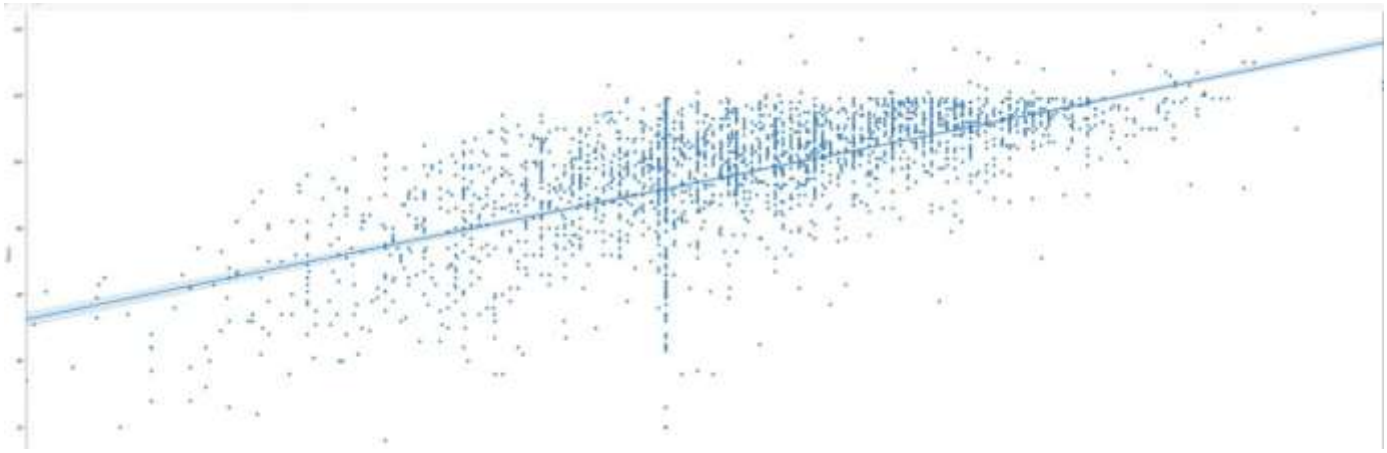
```
# Trouvons la fonction de corrélation la plus élevée
corr=df.corr().abs()
rst=corr[['Points']].sort_values(by=['Points'])
rst=rst.drop(rst[rst['Points']==1],axis=0)
print('La correlation la plus élévée :',rst.max().values)
print(rst[rst['Points']==rst["Points"].max()].index.values)
```

```
La correlation la plus élévée : [0.58081075]
['Moyenne candidat en Physique/Chimie Trimestre 2 Terminale']
```

Ici, rst est la liste des fonctionnalités corrélées avec Y = 'Points'. Nous voyons que la correlation la plus élévée est "Moyenne candidat en Physique/Chimie Trimestre 2 Terminale", elle est d'environ 0.58.

7) Visualisons la corrélation entre la meilleure fonctionnalité et Y

```
# Visualisons la corrélation entre la meilleure fonctionnalité et Y
sns.regplot(x=df['Moyenne candidat en Physique/Chimie Trimestre 2 Terminale'],y=df['Points'])
plt.title('Linear Regression model of the best feature with r at :'+str(rst.max().values))
plt.show()
```



II) Sélection de fonctionnalité (Feature selection)

La sélection d'entités est un processus dans lequel nous sélectionnons automatiquement les entités des données qui contribuent le plus à la variable de prédiction ou au résultat qui nous intéresse. Le fait d'avoir des caractéristiques non-pertinentes dans nos données peut diminuer la précision de nombreux modèles, en particulier les algorithmes linéaires tels que la régression linéaire et logistique. Et les trois avantages de la sélection des fonctionnalités avant de modéliser nos données sont les suivants:

Réduire le surajustement: Moins de données redondantes signifie moins de possibilités de prendre des décisions basées sur le bruit.

Améliorer la precision (Accuracy): Moins de données trompeuses signifie que la précision de la modélisation s'améliore.

Réduire le temps d'entraînement: Moins de données signifie que les algorithmes s'entraînent plus rapidement.

Nous souhaitons conserver seulement les fonctionnalités ayant une forte corrélation avec la variable cible. Cela implique que la fonction d'entrée a une forte influence sur la prédiction de la variable cible. Nous avons fixé le seuil à la valeur absolue de 0,4. Nous conservons les entités d'entrée uniquement si la corrélation de l'entité d'entrée avec la variable cible est supérieure à 0,4

Sélection de fonctionnalités avec toutes les fonctionnalités >= 0,4 et standardisation

```
# Sélection de fonctionnalités avec toutes les fonctionnalités >= 0,4 et standardisation
print('Features with r >=0.4 :')
print(rst[rst['Points']>=0.4])
selected_features=list(df.loc[:,rst[rst['Points']>=0.4].index.values].columns.values)
X_train=df.loc[:,rst[rst['Points']>=0.4].index.values]
scaler = StandardScaler().fit(X_train)
X_train=scaler.transform(X_train)
```

```
Features with r >=0.4 :
                                                  Points
Moyenne candidat en Français Trimestre 2 Première   0.427229
Moyenne candidat en Français Trimestre 3 Première   0.437966
Moyenne candidat en Physique/Chimie Trimestre 1...  0.476634
Moyenne candidat en Mathématiques Trimestre 1 P...  0.488494
Moyenne candidat en Mathématiques Trimestre 3 P...  0.519173
Moyenne candidat en Physique/Chimie Trimestre 2...  0.520310
Moyenne candidat en Mathématiques Trimestre 2 P...  0.532108
Moyenne candidat en Physique/Chimie Trimestre 3...  0.534765
Moyenne candidat en Physique/Chimie Trimestre 1...  0.544117
Moyenne candidat en Mathématiques Trimestre 1 T...  0.552535
Moyenne candidat en Mathématiques Trimestre 2 T...  0.579967
Moyenne candidat en Physique/Chimie Trimestre 2...  0.580811
```

Les informations contenues dans la variable n'étant pas élevées, nous supprimons les fonctionnalités si elles ont plus de 50% de valeurs manquantes.

III) Traitement des valeurs aberrantes

Qu'est-ce qu'une valeur aberrante?

La valeur aberrante est une terminologie couramment utilisée par les analystes et les scientifiques des données, car elle nécessite une attention particulière, sinon elle peut entraîner des estimations extrêmement fausses. Pour parler simplement, un outlier est une observation qui apparaît loin et diverge d'un modèle global dans un échantillon.

Comment la détecter?

La méthode la plus couramment utilisée pour détecter les valeurs aberrantes est la visualisation. Nous utilisons diverses méthodes de visualisation, telles que le box-plot, l'histogramme, le nuage de point ou scatter plot (ci-dessus, nous avons utilisé un box plot et un scatter plot pour la visualisation). Certains analystes ont également diverses règles empiriques pour détecter les valeurs aberrantes. Certains d'entre eux sont:

Toute valeur, qui est au-delà de la plage de -1,5 x IQR à 1,5 x IQR et utilise des méthodes de plafonnement. Toute valeur hors de la plage du 5e et du 95e centile peut être considérée comme aberrante.

Les points de données: Au moins trois écarts-types par rapport à la moyenne sont considérés comme des valeurs aberrantes.

La détection des valeurs aberrantes n'est qu'un cas particulier de l'examen des données pour les points de données influents et dépend également de la compréhension de l'entreprise.

Les valeurs aberrantes bivariées et multivariées sont généralement mesurées à l'aide d'un indice d'influence ou de levier, ou de distance. Des indices populaires tels que la distance de Mahalanobis et le D de Cook sont fréquement utilisés pour détecter les valeurs aberrantes.

Supprimer des observations

Nous supprimons les valeurs aberrantes si elles sont dues à une erreur de saisie de données, à une erreur de traitement des données ou à des observations aberrantes en très petit nombre. Nous pouvons également utiliser la coupe aux deux extrémités pour supprimer les valeurs aberrantes.

Transformer et classer les valeurs

La transformation des variables peut également éliminer les valeurs aberrantes. Le logarithme naturel d'une valeur réduit la variation causée par les valeurs extrêmes. Le binning est également une forme de transformation variable. L'algorithme de l'arbre de décision permet de bien gérer les valeurs aberrantes en raison du regroupement des variables. Nous pouvons également utiliser le processus d'attribution de poids à différentes observations.

Imputation

À l'instar de l'imputation des valeurs manquantes, nous pouvons également imputer des valeurs aberrantes. Nous pouvons utiliser des méthodes d'imputation moyenne, médiane et modale. Avant d'imputer des valeurs, nous devons analyser s'il s'agit d'une valeur aberrante naturelle ou artificielle. Si c'est artificiel, nous pouvons aller avec des valeurs imputées. Nous pouvons également utiliser un modèle statistique pour prédire les valeurs de l'observation aberrante et après cela, nous pouvons les imputer avec des valeurs prédites.

Traiter séparément

S'il existe un nombre significatif de valeurs aberrantes, nous devons les traiter séparément dans le modèle statistique. L'une des approches consiste à traiter les deux groupes comme deux groupes différents et à créer un modèle individuel pour les deux groupes, puis à combiner le résultat.

Le code ci-dessous peut être utilisé pour se débarrasser des valeurs aberrantes en utilisant la méthode 1.5 * IQR:

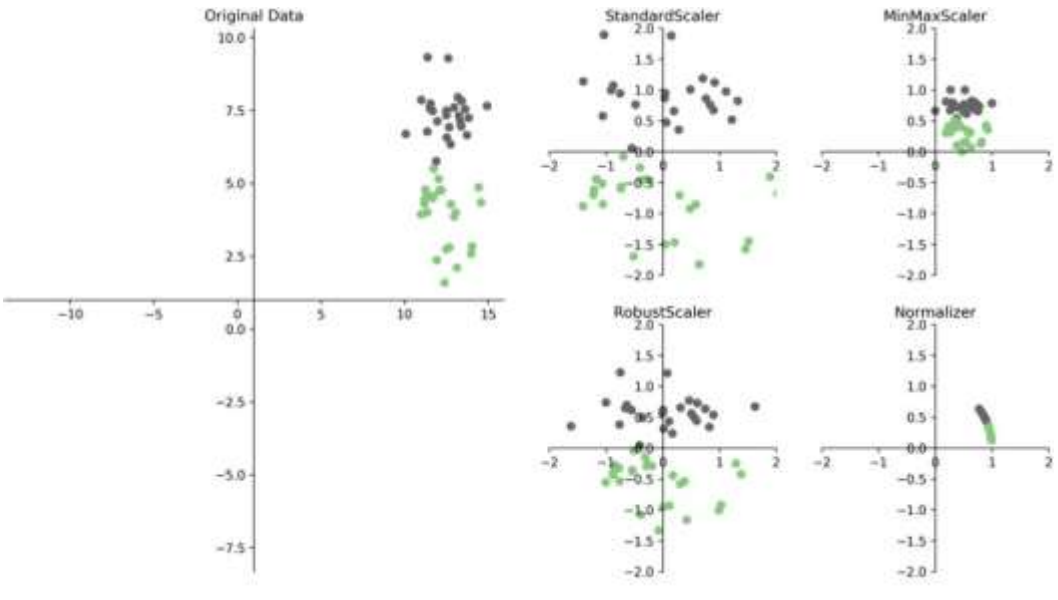
```python
def remove_outlier(df_in, col):
    for col_name in col :
      q1 = df_in[col_name].quantile(0.25)
      q3 = df_in[col_name].quantile(0.75)
      iqr = q3-q1 #Gamme interquartile
      fence_low  = q1-1.5*iqr
      fence_high = q3+1.5*iqr
      df_out = df_in.loc[(df_in[col_name] > fence_low) & (df_in[col_name] < fence_high)]
    return df_out
df=remove_outlier(df,df.columns.values)
print('Forme de la df après la suppression des valeurs aberrantes :')
df.shape
```

```
Forme de la df après la suppression des valeurs aberrantes :
(2366, 73)
```

Nous avons maintenans un dataset avec 2366 observations et 73 fonctionnalités après la suppression des valeurs aberrantes.

IV) Méthodes de mise à l'échelle des données:

Il existe plusieurs façons de mettre à l'échelle nos données, comme illustré dans la figure ci-dessous. Chacune de ces méthodes est implémentée dans une classe Python dans scikit-learn. L'un des moyens les plus courants de mettre à l'échelle les données est de s'assurer que les données ont une moyenne nulle et une variance unitaire après la mise à l'échelle (également appelée standardisation ou parfois z-scoring), qui est implémentée dans StandardScaler.



Semblable à l'ajustement d'un modèle, nous pouvons faire le “fit” de notre mise à l'échelle aux données en utilisant la méthode d'ajustement sur l'ensemble d'apprentissage. Cela évalue simplement la

moyenne et l'écart type. Vous pourriez considérer la mise à l'échelle comme une procédure d'apprentissage non-supervisée très simple, qui ne nécessite pas certainement la cible y, il suffit donc de passer notre fonctionnalité X_train à la méthode “fit”:

Afin d'éviter que des informations sur la distribution de l'ensemble de test ne fuient dans notre modèle, nous devons adapter le scaler à nos données d'entraînement uniquement, puis normaliser à la fois les ensembles d'entraînement et de test avec ce scaler. Nous devrions utiliser StandardScaler avant de diviser les données en train / test

Ici, en d'autres termes, nous divisons un ensemble d'entraînement en ensembles Train et Validation, et nous apprenons l'ajustement uniquement sur le train, puis nous l'appliquons aux ensembles de validation et de test.

V) Preprocessing

Ici, nous utilisons ‘Scaler” sur Train data, et après nous les transférons aux Test data.

```
X_train=df.loc[:,rst[rst['Points']>=0.4].index.values]
scaler = StandardScaler().fit(X_train)
X_train=scaler.transform(X_train)
X_train.shape
```

```
(2366, 12)
```

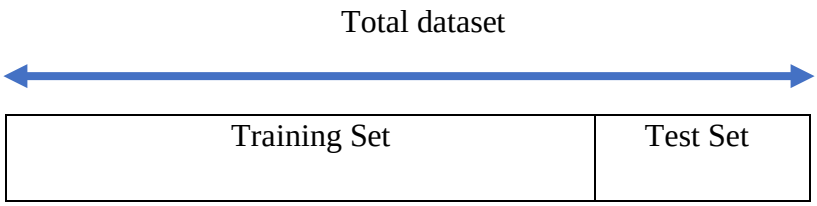
Maintenant, nous avons un dataset avec 2366 observations et 12 fonctionnalités (features)

VI) Train/Test Split

Comme nous le savons, dans les statistiques et l'apprentissage automatique, nous divisons généralement nos données en deux sous-ensembles: les données d'entraînement et les données de test (et parfois en trois pour: former, valider et tester), et ajuster notre modèle sur les données du train, afin de faire des predictions sur les données de test. Lorsque nous faisons cela, une de ces deux choses peut arriver: nous sur-ajustons notre modèle ou nous sous-ajustons notre modèle. Nous ne voulons pas des choses se produisent, car elles affectent la prévisibilité de notre modèle. Nous utilisons peut-être un modèle moins précis et/ou non généralisé (ce qui signifie que vous ne pouvez pas généraliser vos prédictions sur d'autres données).

Il est à noter que le sous-ajustement n'est pas aussi répandu que le sur-ajustement. Néanmoins, nous voulons éviter ces deux problèmes lors de l'analyse des données. Comme nous le verrons, le fractionnement train/test et la validation croisée permettent d'éviter le surajustement plus que le sous-ajustement.

Comme nous l'avons déjà dit, les données que nous utilisons sont généralement divisées en données d'entraînement et données de test. L'ensemble d'apprentissage contient une sortie connue et le modèle apprend sur ces données afin d'être généralisé à d'autres données ultérieurement. Nous avons le jeu de données de test (ou sous-ensemble) afin de tester la prédiction de notre modèle sur ce sous-ensemble.



Nous utilison la bibliothèque Scikit-Learn et en particulier la méthode train_test_split avec l'importation des bibliothèques nécessaires:

```
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV,train_test_split
from sklearn.metrics import r2_score
from scipy import stats
```

Lors de la création d'un modèle machine learning, nous devons toujours diviser les données entre l'ensemble d'apprentissage et l'ensemble de test. Nous pouvons maintenant utiliser la fonction train_test_split pour effectuer le fractionnement. C'est généralement autour de 80/20 ou 70/30. Le test_size = 0.3 à l'intérieur de la fonction indique le pourcentage des données qui doivent être conservées pour les tests. Cela signifie que nous utilisons 70% de l'ensemble de données pour l'ensemble d'apprentissage (train data). Si tout se passe bien avec le modèle de construction plus tard, nous reviendrons en arrière pour obtenir 100% train data pour entraîner le modèle.

```
#Divisons (split) train, validate (70/30)
Y=df.loc[:,'Points'].to_frame()
x_train,x_validate,y_train,y_validate=train_test_split(X_train,Y,test_size=0.3, random_state=5)
print('forme de x_train :'+ str(x_train.shape))
print('forme de y_validate :'+ str(y_validate.shape))
```

```
forme de x_train :(1656, 12)
forme de y_validate :(710, 1)
```

Le modèle de régression linéaire et le modèle SVR seront formés avec les valeurs de l'ensemble d'apprentissage et les prédictions sont testées sur l'ensemble de test.

- Modèle de régression linéaire

La régression linéaire tente de modéliser la relation entre deux variables en ajustant une équation linéaire aux données observées. Une variable est considérée comme une variable explicative et l'autre est considérée comme une variable dépendante.

La régression linéaire est un modèle linéaire, par exemple, un modèle qui suppose une relation linéaire entre les variables d'entrée (X) et la variable de sortie unique (Y). Plus spécifiquement, ce y peut être calculé à partir d'une combinaison linéaire des variables d'entrée (X).

Lorsqu'il n'y a qu'une seule variable d'entrée (X), la méthode est appelée régression linéaire simple. Lorsqu'il y a plusieurs variables d'entrée, la littérature statistique se réfère souvent à la méthode comme une régression linéaire multiple.

- Modèle non linéaire : Support Vector Regression (SVR)

Support Vector Machine (SVM) est un algorithme d'apprentissage automatique très populaire utilisé à la fois dans la régression et la classification. La régression vectorielle de support est similaire à la régression linéaire en ce que l'équation de la ligne est y = wx + b En SVR, cette ligne droite est appelée hyperplan. Les points de données de chaque côté de l'hyperplan qui sont les plus proches de l'hyperplan sont appelés vecteurs de support qui sont utilisés pour tracer la ligne de démarcation. Contrairement à d'autres modèles de régression qui tentent de minimiser l'erreur entre la valeur réelle et la valeur prédite, le SVR essaie d'ajuster la meilleure ligne dans une valeur de seuil (distance entre l'hyperplan et la ligne de démarcation), a. Ainsi, nous pouvons dire que le modèle SVR essaie de satisfaire la condition -a < y-wx + b < a. Il a utilisé les points avec cette limite pour prédire la valeur.

Nous testons les 2 modèles sur les données d'entraînement et nous montrons les scores de précision:

```
# Testons les 2 modèles
linear_model=LinearRegression()
non_linear_model=SVR()

linear_model.fit(x_train,y_train)
non_linear_model.fit(x_train,y_train)
```

```
# Trouvons la score précision de l'approche linéaire et l'approche non linéaire
Y_hat_linear=linear_model.predict(x_validate)
Y_hat_nonlinear=non_linear_model.predict(x_validate)
r2_lin=r2_score(y_validate,Y_hat_linear)
r2_nonlin=r2_score(y_validate,Y_hat_nonlinear)
print('Score précision de l''approche linéaire:'+str(r2_score(y_validate,Y_hat_linear)))
print('Score de précision de l''approche non linéaire :'+str(r2_score(y_validate,Y_hat_nonlinear)))
if r2_lin > r2_nonlin :
  print('Le modèle le plus optique est la régression linéaire')
else :
  print('Le modèle le plus optique est SVR')
```

```
Score précision de lapproche linéaire:0.6770783211372393
Score de précision de lapproche non linéaire :0.5957954765016689
Le modèle le plus optique est la régression linéaire
```

y_hat.flatten est pour l'étirement le “array” est unidimensionnel. Puis, nous l'avons transformé en liste car il doit correspondre au dataset pour les étapes suivantes. Nous voyons que le score de précision du modèle de régression linéaire est supérieur au score du modèle SVR. Donc, le modele le plus optique est la régression linéare.

Nous voyons que le score de précision (accuracy) des 2 modele, et nous choisissons le modèle de régression linéaire parce que sa score de précision (accuracy) est plus grand que le modèle SVR.

VII) Calcul le “r2 score” et RMSE pour le modèle de régression linéaire et le modèle non linéaire SVR

Nous importons la bibliothèque:

```
from sklearn.metrics import mean_squared_error, r2_score
```

```
def calculateModel(real, predict):
    rmse = np.sqrt(mean_squared_error(real, predict))
    r2 = r2_score(real, predict)
    print("rmse:",rmse)
    print("r2 score:",r2)
```

- Nous calculons r2 score et RMSE pour le modèle SVR

```
#Calculons r2 score et RMSE pour le modèle SVR
print("Modèle non linéaire SVR")
print("----")
lr = SVR(kernel='rbf').fit(x_train, y_train)
lr_pred = lr.predict(x_train)
print("Train set de SVR")
calculateModel(y_train,lr_pred)
```

```
Modèle non linéaire SVR
----
/usr/local/lib/python3.7/dist-packages/sklearn/utils/v
  y = column_or_1d(y, warn=True)
Train set de SVR
rmse: 15.764004410795222
r2 score: 0.4807510827101442
```

- Nous calculons r2 score et RMSE pour le Modèle de régression linéaire

```
#Calcullons r2 score et RMSE pour le modèle de régression linéaire
print("Model de Régression linéaire")
print("------")
lr = LinearRegression(normalize=True).fit(x_train, y_train)
lr_pred = lr.predict(x_train)
print("Train set de régression linéaire")
calculateModel(y_train,lr_pred)
```

```
Model de Régression linéaire
------
Train set de régression linéaire
rmse: 14.395220719232771
r2 score: 0.5670087095643493
```

Nous voyons que des valeurs plus faibles de RMSE indiquent un meilleur ajustement. RMSE est une bonne mesure de la précision avec laquelle le modèle prédit la réponse, et c'est le critère d'ajustement le plus important si l'objectif principal du modèle est la prédiction. La meilleure mesure de l'ajustement du modèle dépend des objectifs du chercheur, et plusieurs sont souvent utiles. Puisqu'il n'y a pas de réponse correcte, la valeur de base de RMSE est de sélectionner un modèle de prédiction plutôt qu'un autre.

De même, nous n'avons pas non plus de réponse correcte quant à ce que devrait être le score R2. Pourtant, il existe des modèles avec un faible score R2 qui sont toujours de bons modèles.

Ici, nous voyons que RMSE du modèle de Régression linéaire est inférieur à celui du modèle SVR et le score R2 de modèle de Régression linéaire est supérieur à celui de modèle SV.

Par conséquent, en tenant compte du fait que le score de précision du modèle de Régression linéaire est supérieure au modèle SVR dans l'étape précédant, et RMSE, le score R2 dans cette étape, nous voyons que le modèle de Régression linéaire est plus optimal que le modèle SVR.

VIII) Sélectionner les meilleurs paramètres pour le modèle de régression à l'aide de Gridsearch

- Objectif de la recette

Bien souvent, en travaillant sur un ensemble de données et en utilisant un modèle d'apprentissage automatique, nous ne savons pas quel ensemble d'hyperparamètres nous donnera le meilleur résultat. Passer manuellement tous les ensembles d'hyperparamètres dans le modèle et vérifier le résultat peut être un travail intense et impossible à faire.

Pour obtenir le meilleur ensemble d'hyperparamètres, nous pouvons utiliser Grid Search. Grid Search transmet toutes les combinaisons d'hyperparamètres une à une dans le modèle et vérifie le résultat. Enfin il nous donne l'ensemble des hyperparamètres qui donne le meilleur résultat après passage dans le modèle.

GridSearchCV doit être utilisé pour trouver les paramètres optimaux pour entraîner notre modèle final. En règle générale, nous devrions exécuter GridSearchCV puis examiner les paramètres qui ont donné au modèle le meilleur score. Nous devrions ensuite prendre ces paramètres et former notre modèle final sur toutes les données. Il est important de noter que si nous avons formé notre modèle final sur toutes nos données, nous ne pouvons pas le tester. Pour tout test correct, nous devons réserver certaines des données.

```
# Les modèles optiques utilisant X_train et Y
best_model=LinearRegression()
params= {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':[True, False]}
grid=GridSearchCV(best_model,params)
grid.fit(X_train,Y)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=LinearRegression(copy_X=True, fit_intercept=True,
                                        n_jobs=None, normalize=False),
             iid='deprecated', n_jobs=None,
             param_grid={'copy_X': [True, False],
                         'fit_intercept': [True, False],
                         'normalize': [True, False]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

IX) TEST DATA

A) Prétraitons les données de test

- Nous prétraitons le Test data

```
# Prétraitons les données de test
test=pd.read_csv('/content/test.csv',sep=';')
test
```

	id	Bac	Nom	Prénom	Genre	Aménagements Validés	Pondération	Méthode de travail	Autonomie	Engagement, Esprit d'initiative	Capacité à s'investir	Niveau de la classe	Cohérence projet de formation
0	0	scientific	MASSON	Valérie	F	NaN	0	Satisfaisante	Assez satisfaisante	Oui	Satisfaisante	Bon	NaN
1	1	scientific	PAGES	Bertrand	H	NaN	0	Très satisfaisante	Très satisfaisante	Non	Très satisfaisante	Bon	NaN
2	2	scientific	SEGUIN	Émilie	F	NaN	0	Satisfaisante	Satisfaisante	Oui	Assez satisfaisante	Moyen	NaN
3	3	scientific	REMY	Alphonse	H	NaN	1	Satisfaisante	Très satisfaisante	Oui	Très satisfaisante	Bon	NaN
4	4	scientific	LE ROUX	Lorraine	F	NaN	0	Assez satisfaisante	Satisfaisante	Oui	Satisfaisante	Moyen	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
845	845	scientific	BOUCHER	Célina	F	NaN	0	Assez satisfaisante	Satisfaisante	Non	Satisfaisante	Bon	NaN

- Nous éliminons les colonnes dont les NaNs occupent plus de 50 %

```
test.drop(columns=['Nom','Prénom'],axis=1,inplace=True)
print('Avant le nettoyage :' + str(test.shape))
features=test.columns.values
for feature in features :
  nan=test[feature].isnull().sum()
  if nan > 1300 :
    test.drop(columns=[feature],axis=1,inplace=True)
print('Après le nettoyage :'+ str(test.shape))
```

```
Avant le nettoyage :(850, 243)
Après le nettoyage :(850, 243)
```

- Nous faisons un encodage numérique pour modéliser les entrées

```
# Faisons un encodage numérique pour modéliser les entrées
df['Genre']=df['Genre'].map({'H':1,'F':0})
df['Bac']=df['Bac'].map({'scientific':1,'sti2d':0})
df['Méthode de travail']=df['Méthode de travail'].map({'Satisfaisante':1,'Très satisfaisante':2,'Assez satisfaisante'
df['Avis sur la capacité à réussir']=df['Avis sur la capacité à réussir'].map({'Satisfaisante':1,'Très satisfaisante'
df['Niveau de la classe']=df['Niveau de la classe'].map({'Bon':1,'Assez bon':2,'Très bon':3,'Faible':4,'Moyen':5})
df["Engagement, Esprit d'initiative"]=df["Engagement, Esprit d'initiative"].map({'Non':0,'Oui':1})
df['Autonomie']=df['Autonomie'].map({'Satisfaisante':1,'Très satisfaisante':2,'Assez satisfaisante':3,'Peu démontrée'
df["Capacité à s'investir"]=df["Capacité à s'investir"].map({'Satisfaisante':1,'Très satisfaisante':2,'Assez satisfai
```

- Nous poussons le dtype vers la droite et convertissons les NaN en moyenne dans chaque colonne

```
new_features=test.columns.values
for i in new_features :
    test[i]=test[i].astype(object)
```

```
new_feat=test.columns.values
for col in new_feat :
  try :
      test[col]=test[col].str.replace(',','.')
      test[col]=test[col].replace(np.nan,'0')
      test[col]=test[col].astype(float)
      test[col]=test[col].replace(0.,np.mean(test[col]))
  except :
    print(col)
```

- Nous remplaçons les valeurs “NaN” restantes dans les colonnes au lieu de la valeur moyenne (mean)

```
for feature in new_feat :
  nan=test[feature].isnull().sum()

  if nan > 0 :
    test[feature]=test[feature].replace(np.nan,0)
    test[feature]=test[feature].replace(0,np.mean(test[feature]))
print([i for i in test.columns if test[i].isnull().any()])
```

- Nous testons sur Test data

```
X_test=test.loc[:,selected_features]
X_test=scaler.transform(X_test)
X_test.shape
```

```
(850, 12)
```

```
y_hat=grid.predict(X_test)
y_hat=list(y_hat.flatten())
print('Prediction :',y_hat)
```

```
Prediction : [74.11765360632204, 108.80658117743992, 72.31061357119077, 109.82481620536134, 43.784863
```

```
rst=pd.DataFrame({
    'ID':test['id'],
    'Points':y_hat
})
print(rst)
rst=rst.set_index('ID')
rst
```

Résultat:

```
      ID      Points
0      0   74.117654
1      1  108.806581
2      2   72.310614
3      3  109.824816
4      4   43.784864
..   ...         ...
845  845   94.076054
846  846   95.110920
847  847  121.118562
848  848   91.220759
849  849  109.171609

[850 rows x 2 columns]
```

ID	Points
0	74.117654
1	108.806581
2	72.310614
3	109.824816
4	43.784864
...	...
845	94.076054
846	95.110920
847	121.118562
848	91.220759
849	109.171609

850 rows × 1 columns

Nous enregistrons le fichier de prévision dans le répertoire

```
rst.to_csv('ML_result.csv')
```

X) Conclusion

La prédiction de la note d'un élève pour un concours lui donne un aperçu et une évaluation de ses capacités, afin qu'il puisse intégrer de manière proactive des mesures pour améliorer ses études et ses notes. L'étude a abordé des questions de recherche pour trouver un modèle à prédire la note du concours.

Sur la base des points des données collectées auprès de 2423 étudiants en trois ans, un modèle de régression linéaire et un modèle non-linéaire (SVR) sont comparés pour choisir le modèle le plus optimal dans l'étude pour prédire les notes de l'élève pour un concours.

Les entrées (prédicteurs / variables indépendantes) des modèles incluent 246 fonctionnalités : id, Bac, Genre, Aménagements Validés, Pondération, Méthode de travail, Autonomie, Engagement, Esprit d'initiative, Capacité à s'investir, Niveau de la classe, Cohérence projet de formation, Avis sur la capacité à réussir, Note à l'épreuve de Oral de Français (épreuve anticipée, Note à l'épreuve de Ecrit de Français (épreuve anticipée), Moyenne générale au bac (N-1), Moyenne candidat en Mathématiques etc etc. Après, nous avons utilisé des techniques de l'ingénierie des fonctionnalités et nous avons supprimé des valeurs aberrantes pour avoir un dataset "propre" pour l'entrainnement. Plusieurs critères ont été utilisés pour évaluer et valider les modèles prédictifs développés, l'un d'eux est l'exactitude ou la précision (accurary).

En termes de précision de prédiction et de pourcentage de bonnes prédictions, le modèle de régression linéaire - est apparemment le modèle optimal parmi les deux modèles. Les résultats de la validation interne et externe montrent que les modèles prédictifs développés ont une précision de prédiction moyenne de 69% à 70%.

Notre travaille se concentre sur la prédiction des notes des élèves pour un concours à l'aide de techniques d'apprentissage automatique. Des fonctionnalités supplémentaires sont ajoutées à notre ensemble de données pour acquérir une meilleure exactitude (accuracy) et des limites de notre projet,

Limites

Nous savons que le modèle de régression linéaire n'est peut-être pas la méthode la plus optimale, il peut y avoir des méthodes optimales telles que:

- Avec un dataset ayant de nombreuses fonctionnalités comme celle-ci, nous devrions utiliser la méthode Analyse en composantes principales (ACP ou PCA en anglais) et clustering de pour le feature engineering.

- Pour l'algorithme, nous pouvons utiliser les modèle Randomforest et Xgboost pour prédire la note, peut-être qu'il est mieux d'utiliser le modèle de Régression Linéaire, alors l'exactitude (accuracy) peut être comprise entre 70 et 85. Après, si nous utilisons la recherche baysienne et le feature engineering, l'exactitude (accuracy) peut-être compris peut être comprise entre 85 et 90.

Cependant, nous sommes deux étudiants en sociologie avec un background en économie, pas des informaticiens, nous avons donc appliqué ces méthodes ci-dessus pour essayer d'y parvenir mais nous ne sommes pas arrivés à destination, nous avons donc décidé de choisir 2 modèles régression linéaire et SVR à comparer et nous avons enfin choisi le modèle de régression linéaire pour prédire la note car il est le plus simple modèle à prédire et sa prédiction n'est pas mauvaise non plus: soit près de 70%.

À l'avenir, avec plus de temps, nous approfondirons la recherche sur ces algorithmes pour optimiser notre système de prédiction de la note.

XI) Réferences

1) Fundamentals of Machine Learning for Predictive Data Analytics, second edition: Algorithms, Worked Examples, and Case Studies Hardcover – October 20, 2020, by John D. Kelleher

2) Applied Predictive Modeling, 2018 Edition, by Max Kuhn

3) Data Science for Business: Predictive Modeling, Data Mining, Data Analytics, Data Warehousing, Data Visualization, Regression Analysis, Database Querying, and Machine Learning for Beginners, 2018, by Herbert Jones