Intelligence Artificielle - Anthony Quere

January 21, 2022

This is a Jupyter notebook builded is a python project with the code available on GitHub. (https://github.com/Anthony-Jhoiro/projet_ia_s1). Only the main functions will be presented in this file. If you want to see the implementation of those functions, I recommand checking directly the repository.

0.1 Préparation des données

Erratum

Scikit ne supporte que les valeurs continuent pour les différents algorithmes utilisés. Par conséquent les champ sde catégories ont été passés en champs booléens puis les champs booléens sont passé en 0 et 1. C'est pourquoi nos retrouvons dans nos centre des valeurs continues pour des champs booléens.

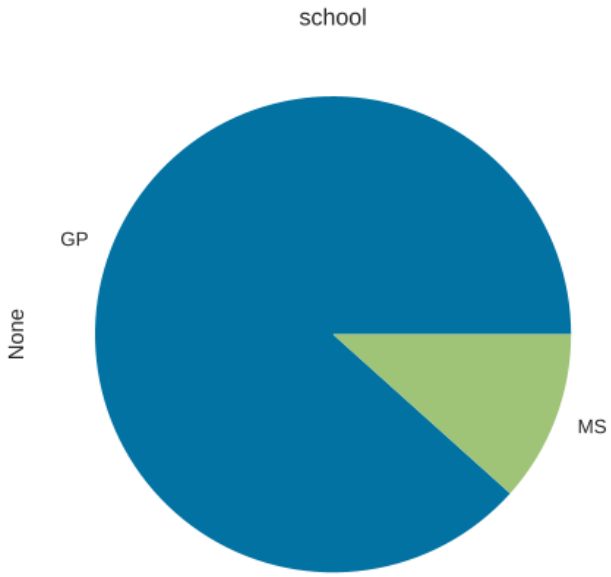
[1]:
```
from dataset_utils import add_score_g3_classes_columns, add_ratio_g1pg2_column

import csv_parser
import pandas as pd
import student_dataset as sd

# Define float output
pd.options.display.float_format = '{:,.4f}'.format

# Load DataFrame
mat_original = csv_parser.parse_student_csv("student-mat.csv", separator=";",
                                            cast_types=sd.dataset_types)
por_original = csv_parser.parse_student_csv("student-por.csv", separator=";",
                                            cast_types=sd.dataset_types)

# Create missing columns
add_score_g3_classes_columns(mat_original)
add_score_g3_classes_columns(por_original)
add_ratio_g1pg2_column(mat_original)
add_ratio_g1pg2_column(por_original)

mat_formatted = csv_parser.format_dataset(mat_original, sd.target_columns, sd.
 ↪string_fields, sd.boolean_fields)
```

```
por_formatted = csv_parser.format_dataset(por_original, sd.target_columns, sd.
  ↪string_fields, sd.boolean_fields)
```

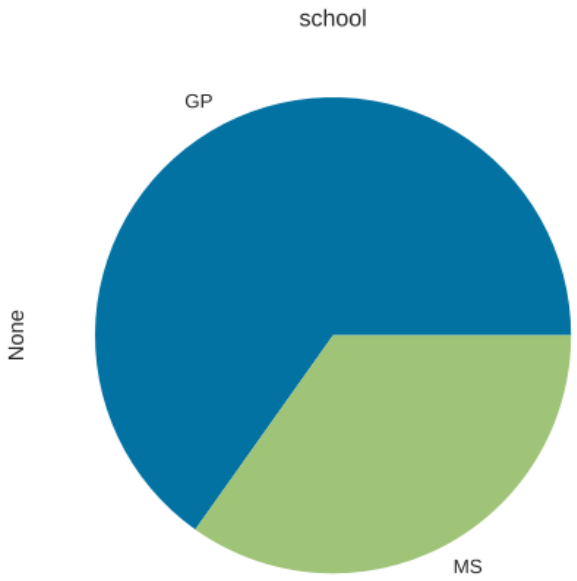
0.2 Partie 1 - Analyse des données

0.2.1 Les écoles

Les élèves de mathématique



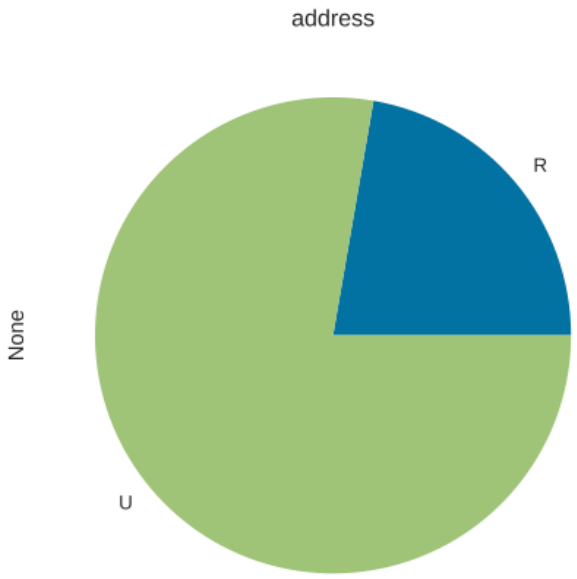
Les élèves de portugais

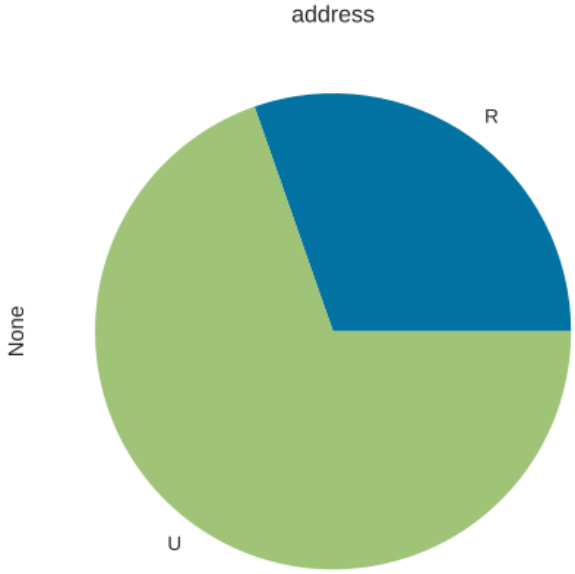


L'attribut "school" nous informe l'école des élèves. Nous pouvons constater que la grande majorité des élèves viennent de l'école Gabriel Pereira. Les analyse seront donc bien plus influencées par les élèves de cette école et ne pourrait donc pas correspondre aux élèves de l'autre école.

0.2.2 L'adresse

Les élèves de mathématique



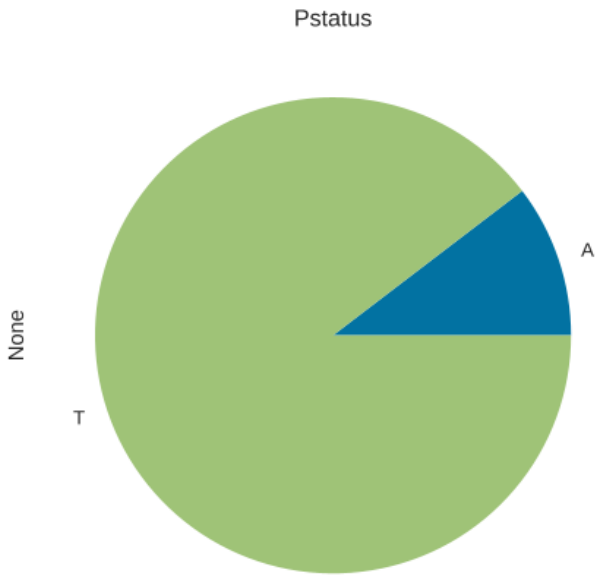


Les élèves de portugais

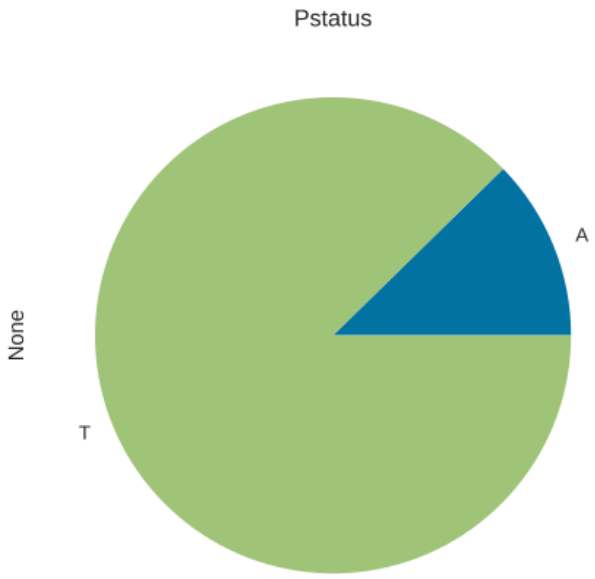
L'attribut "address" nous renseigne sur le milieu dans lequel les élèves vivent. Ici, nous pouvons constatez que la majorité des élèves vivent en milieu Urbain, que ce soit pour les cous de portugais ou les cours de mathématique.

0.2.3 La situation des parents

Les élèves de mathématique



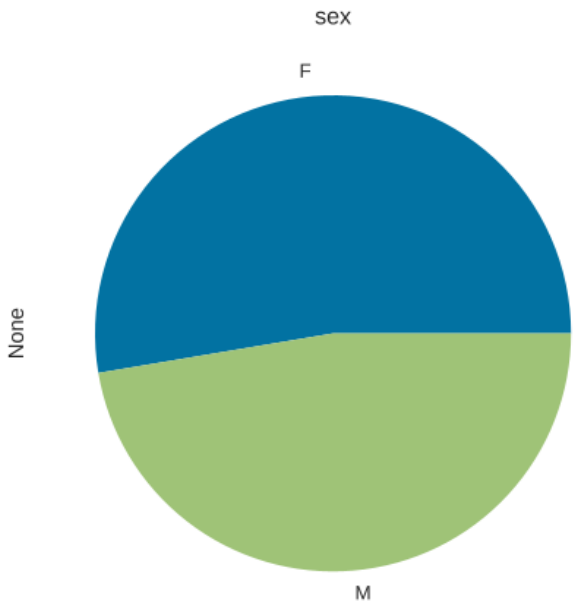
Les élèves de portugais



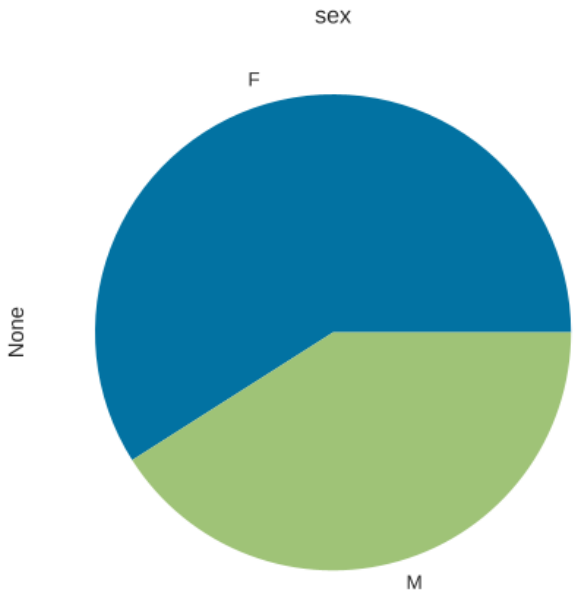
L'attribut "Pstatus" prend pour valeur T si les parents de l'élève vivent ensemble et A dans les autres cas. La majorité des parents des élèves vivent ensemble.

0.2.4 Le sexe des élèves

Les élèves de mathématique



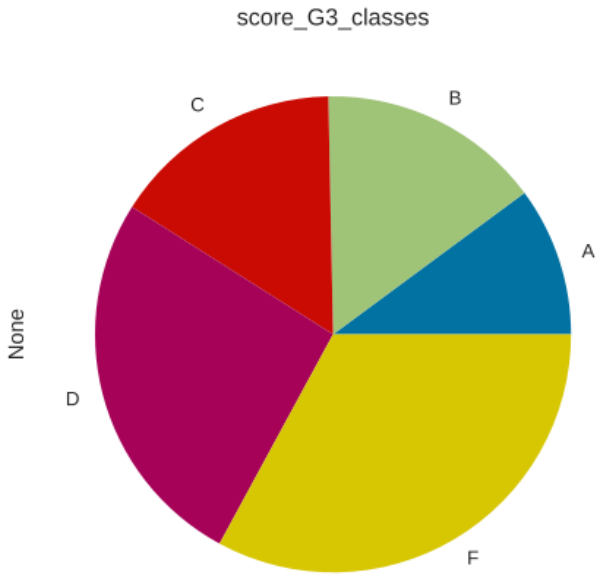
Les élèves de portugais



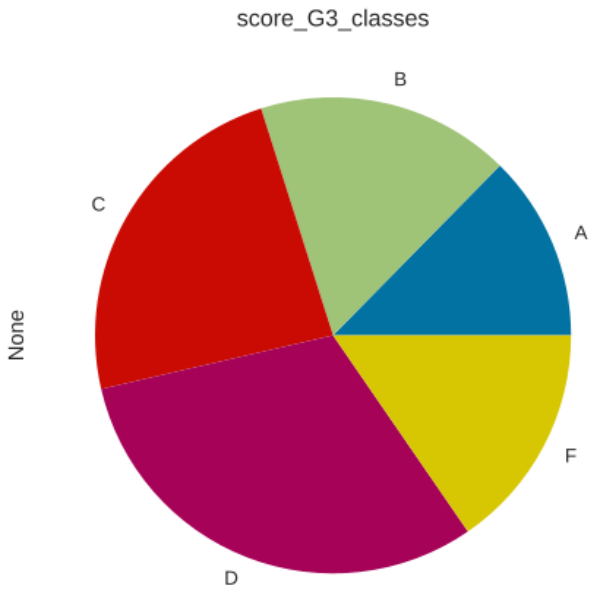
L'attribut "sex" nous renseigne sur le sexe de l'élève. Ici, nous avons une majorité de fille parmi les élève ce qui est conforme à la répartition en France cette année d'après cette publication de l'INSEE (https://www.insee.fr/en/statistiques/2382597?sommaire=2382613).

0.2.5 La note G3 (avec les classes)

Les élèves de mathématique



Les élèves de portugais



Les notes sont globalement faibles dans les deux matières même si les notes des élèves de potugais sont généralement plus élevées que celles des élèves de mathématique. (moins de F mais plus de D et de C).

0.3 Partie 2 - Segmentation des étudiants

0.3.1 Préparation du jeu de données

On retire les colonnes ciblées car nous souhaitons par la suite caractériser nos clusters par rapport à ces attributs. Les attributs G1 et G2 sont également supprimés car l'attribut ratio_G1+G2 est obtenu par ces colonnes, ce qui fausse donc le résultat.

[2]:
```
excluded_columns = [
    "G1",
    "G2",
    "G3",
    "ratio_G1+G2",
    "score_G3_classes_A",
    "score_G3_classes_B",
    "score_G3_classes_C",
    "score_G3_classes_D",
    "score_G3_classes_F",
]
```

0.3.2 Pour les élèves de mathématique

[3]:
```
dataset_for_predictions = mat_formatted.copy()
dataset_for_predictions = csv_parser.drop_columns(dataset_for_predictions,␣
 ↪excluded_columns)

# Nous souhaitons travailler avec 2 clusters à chaque fois.
cluster_count = 3
```

Algorithme du Kmean En utilisant l'algorithme du knn sur notre jeu de donnée, nous optenons 3 clusters C0, C1 et C2 dont voici les centres :

[4]:
```
from sklearn.cluster import KMeans


knn_predictions = KMeans(n_clusters=cluster_count, random_state=1)
knn_predictions.fit(dataset_for_predictions)

pd.DataFrame({
    "field": knn_predictions.feature_names_in_,
    "Centre de C0": knn_predictions.cluster_centers_[0],
    "Centre de C1": knn_predictions.cluster_centers_[1],
    "Centre de C2": knn_predictions.cluster_centers_[2],
})
```

[4]:

	field	Centre de C0	Centre de C1	Centre de C2
0	age	16.5566	17.1605	17.8000
1	Medu	2.6731	3.0247	3.0000
2	Fedu	2.5081	2.5432	3.0000
3	failures	0.2977	0.4691	0.4000
4	schoolsup	0.1327	0.1111	0.2000
5	famsup	0.6052	0.6420	0.6000
6	paid	0.4563	0.4691	0.4000
7	activities	0.5178	0.4815	0.4000
8	nursery	0.7864	0.8272	0.8000
9	higher	0.9482	0.9630	0.8000
10	internet	0.8123	0.9012	1.0000
11	romantic	0.2977	0.4444	0.8000
12	famrel	3.9806	3.7901	4.2000
13	freetime	3.2201	3.3580	2.2000
14	goout	3.0680	3.2963	2.6000
15	Dalc	1.4207	1.7160	1.4000
16	Walc	2.1877	2.7037	2.0000
17	health	3.5728	3.4815	3.6000
18	absences	2.6052	14.6543	52.6000
19	school_GP	0.8706	0.9259	1.0000
20	school_MS	0.1294	0.0741	-0.0000

21	address_R	0.2330	0.1728	0.4000
22	address_U	0.7670	0.8272	0.6000
23	famsize_GT3	0.7282	0.6420	0.8000
24	famsize_LE3	0.2718	0.3580	0.2000
25	Pstatus_A	0.0841	0.1728	0.2000
26	Pstatus_T	0.9159	0.8272	0.8000
27	Mjob_at_home	0.1586	0.1235	0.0000
28	Mjob_health	0.0906	0.0741	0.0000
29	Mjob_other	0.3592	0.3210	0.8000
30	Mjob_services	0.2460	0.3333	0.0000
31	Mjob_teacher	0.1456	0.1481	0.2000
32	Fjob_at_home	0.0518	0.0494	-0.0000
33	Fjob_health	0.0421	0.0617	0.0000
34	Fjob_other	0.5340	0.6173	0.4000
35	Fjob_services	0.2945	0.2099	0.6000
36	Fjob_teacher	0.0777	0.0617	0.0000
37	reason_course	0.3981	0.2716	0.0000
38	reason_home	0.2621	0.3210	0.4000
39	reason_other	0.0841	0.1235	0.0000
40	reason_reputation	0.2557	0.2840	0.6000
41	guardian_father	0.2460	0.1728	0.0000
42	guardian_mother	0.6926	0.6914	0.6000
43	guardian_other	0.0615	0.1358	0.4000
44	traveltime_1	0.6343	0.7160	0.6000
45	traveltime_2	0.2783	0.2346	0.4000
46	traveltime_3	0.0647	0.0370	-0.0000
47	traveltime_4	0.0227	0.0123	-0.0000
48	studytime_1	0.2718	0.2469	0.2000
49	studytime_2	0.4693	0.6173	0.6000
50	studytime_3	0.1812	0.0988	0.2000
51	studytime_4	0.0777	0.0370	-0.0000
52	sex_F	0.5178	0.5432	0.8000
53	sex_M	0.4822	0.4568	0.2000

LE tableau ci dessus nous présente les différents attributs utilisés ainsi que leurs valeurs pour les centres de nos 3 clusters.

Interessons nous maintenant à la répartition des données au sein de ces clusters.

[5]:
```
mat_original['knn_cluster'] = knn_predictions.labels_

knn_c0 = dataset_for_predictions[mat_original['knn_cluster'] == 0]
knn_c1 = dataset_for_predictions[mat_original['knn_cluster'] == 1]
knn_c2 = dataset_for_predictions[mat_original['knn_cluster'] == 2]


pd.DataFrame({
    "Cluster": ["C0", "C1", "C2"],
```

```
    "Nombre d'éléments": [len(knn_c0), len(knn_c1), len(knn_c2)],
})
```

[5]:

	Cluster	Nombre d'éléments
0	C0	309
1	C1	81
2	C2	5

Le tableau ci dessus nous présente la quantité de donnée correspondant à chaque cluster. Nous pouvons constater, que le cluster C0 est nettement plus imposant que les deux autres (il contient 98% de données). Les prédictions résultantes en seront impacté.

Clustering hierarchique Une autre méthode pour clusteriser nos données est le clustering hierarchique.

[6]:
```
from sklearn import cluster

aglo_clusters = cluster.AgglomerativeClustering(cluster_count).
 ↪fit(dataset_for_predictions)

mat_original['aglo_cluster'] = aglo_clusters.labels_

aglo_c0 = dataset_for_predictions[mat_original['aglo_cluster'] == 0]
aglo_c1 = dataset_for_predictions[mat_original['aglo_cluster'] == 1]
aglo_c2 = dataset_for_predictions[mat_original['aglo_cluster'] == 2]

aglo_c0_center = aglo_c0.mean(axis=0)
aglo_c1_center = aglo_c1.mean(axis=0)
aglo_c2_center = aglo_c2.mean(axis=0)

pd.DataFrame({
    "Centre de C0": aglo_c0_center,
    "Centre de C1": aglo_c1_center,
    "Centre de C2": aglo_c2_center,
})
```

[6]:

	Centre de C0	Centre de C1	Centre de C2
age	16.5596	17.3175	17.8000
Medu	2.7095	2.9365	3.0000
Fedu	2.5138	2.5238	3.0000
failures	0.2844	0.5873	0.4000
schoolsup	0.1315	0.1111	0.2000
famsup	0.6055	0.6508	0.6000
paid	0.4587	0.4603	0.4000
activities	0.5168	0.4762	0.4000
nursery	0.7890	0.8254	0.8000
higher	0.9511	0.9524	0.8000

internet	0.8165	0.9048	1.0000
romantic	0.3058	0.4444	0.8000
famrel	3.9755	3.7619	4.2000
freetime	3.2232	3.3810	2.2000
goout	3.0642	3.3810	2.6000
Dalc	1.4251	1.7778	1.4000
Walc	2.1804	2.8889	2.0000
health	3.5688	3.4762	3.6000
absences	3.0153	15.9683	52.6000
school_GP	0.8746	0.9206	1.0000
school_MS	0.1254	0.0794	0.0000
address_R	0.2294	0.1746	0.4000
address_U	0.7706	0.8254	0.6000
famsize_GT3	0.7248	0.6349	0.8000
famsize_LE3	0.2752	0.3651	0.2000
Pstatus_A	0.0887	0.1746	0.2000
Pstatus_T	0.9113	0.8254	0.8000
Mjob_at_home	0.1560	0.1270	0.0000
Mjob_health	0.0856	0.0952	0.0000
Mjob_other	0.3639	0.2857	0.8000
Mjob_services	0.2416	0.3810	0.0000
Mjob_teacher	0.1529	0.1111	0.2000
Fjob_at_home	0.0489	0.0635	0.0000
Fjob_health	0.0428	0.0635	0.0000
Fjob_other	0.5443	0.5873	0.4000
Fjob_services	0.2844	0.2381	0.6000
Fjob_teacher	0.0795	0.0476	0.0000
reason_course	0.4006	0.2222	0.0000
reason_home	0.2508	0.3968	0.4000
reason_other	0.0887	0.1111	0.0000
reason_reputation	0.2599	0.2698	0.6000
guardian_father	0.2385	0.1905	0.0000
guardian_mother	0.7003	0.6508	0.6000
guardian_other	0.0612	0.1587	0.4000
traveltime_1	0.6422	0.6984	0.6000
traveltime_2	0.2752	0.2381	0.4000
traveltime_3	0.0612	0.0476	0.0000
traveltime_4	0.0214	0.0159	0.0000
studytime_1	0.2661	0.2698	0.2000
studytime_2	0.4709	0.6508	0.6000
studytime_3	0.1835	0.0635	0.2000
studytime_4	0.0795	0.0159	0.0000
sex_F	0.5229	0.5238	0.8000
sex_M	0.4771	0.4762	0.2000

De la même manière, interessons nous à la répartition des données parmi nos clusters.

[7]:
```
pd.DataFrame({
    "Cluster": ["C0", "C1", "C2"],
    "Nombre d'éléments": [len(aglo_c0), len(aglo_c1), len(aglo_c2)],
})
```

[7]:

	Cluster	Nombre d'éléments
0	C0	327
1	C1	63
2	C2	5

Là encore, le cluster C0 est aussi très imposant. Une mauvaise répartition des données peut lourdement affecter

Evaluation de nos clusters Nous allons maintenant analyser la qualité de nos clusters.

Il est peu pertinant de comparer les données des clusters car leur nombre d'éléments diffère trop.

0.3.3 Pour les élèves de portugais

[8]:
```
dataset_for_predictions = por_formatted.copy()
dataset_for_predictions = csv_parser.drop_columns(dataset_for_predictions,␣
 ↪excluded_columns)

# Nous souhaitons travailler avec 2 clusters à chaque fois.
cluster_count = 3
```

Algorithme du Kmean En utilisant l'algorithme du knn sur notre jeu de donnée, nous optenons 3 clusters C0, C1 et C2 dont voici les centres :

[9]:
```
from sklearn.cluster import KMeans


knn_predictions = KMeans(n_clusters=cluster_count, random_state=1)
knn_predictions.fit(dataset_for_predictions)

pd.DataFrame({
    "field": knn_predictions.feature_names_in_,
    "Centre de C0": knn_predictions.cluster_centers_[0],
    "Centre de C1": knn_predictions.cluster_centers_[1],
    "Centre de C2": knn_predictions.cluster_centers_[2],
})
```

[9]:

	field	Centre de C0	Centre de C1	Centre de C2
0	age	16.8414	17.1837	16.6273
1	Medu	2.5991	2.3061	2.4906
2	Fedu	2.3304	2.4082	2.2788
3	failures	0.2291	0.4694	0.1850

4	schoolsup	0.0881	0.0816	0.1180
5	famsup	0.6167	0.6327	0.6086
6	paid	0.0705	0.0204	0.0590
7	activities	0.5066	0.3878	0.4853
8	nursery	0.7665	0.7959	0.8257
9	higher	0.8987	0.7143	0.9142
10	internet	0.7577	0.8776	0.7587
11	romantic	0.3833	0.4490	0.3485
12	famrel	3.7974	3.8367	4.0241
13	freetime	3.1674	3.1429	3.1930
14	goout	3.2467	3.4082	3.1180
15	Dalc	1.4934	2.1224	1.4263
16	Walc	2.3789	2.8163	2.1501
17	health	3.4273	3.6939	3.5818
18	absences	5.9736	15.6327	0.6783
19	school_GP	0.6740	0.8571	0.6113
20	school_MS	0.3260	0.1429	0.3887
21	address_R	0.2863	0.2245	0.3244
22	address_U	0.7137	0.7755	0.6756
23	famsize_GT3	0.7269	0.6939	0.6917
24	famsize_LE3	0.2731	0.3061	0.3083
25	Pstatus_A	0.1322	0.2449	0.1019
26	Pstatus_T	0.8678	0.7551	0.8981
27	Mjob_at_home	0.2115	0.2041	0.2064
28	Mjob_health	0.0573	0.0000	0.0938
29	Mjob_other	0.3833	0.4694	0.3968
30	Mjob_services	0.2291	0.2653	0.1903
31	Mjob_teacher	0.1189	0.0612	0.1126
32	Fjob_at_home	0.0881	0.0612	0.0509
33	Fjob_health	0.0352	0.0000	0.0402
34	Fjob_other	0.5903	0.5714	0.5496
35	Fjob_services	0.2291	0.3265	0.3029
36	Fjob_teacher	0.0573	0.0408	0.0563
37	reason_course	0.4141	0.4082	0.4584
38	reason_home	0.2467	0.3265	0.2064
39	reason_other	0.0925	0.0816	0.1260
40	reason_reputation	0.2467	0.1837	0.2091
41	guardian_father	0.1762	0.2041	0.2761
42	guardian_mother	0.7489	0.6122	0.6836
43	guardian_other	0.0749	0.1837	0.0402
44	traveltime_1	0.5595	0.5714	0.5657
45	traveltime_2	0.3348	0.3265	0.3244
46	traveltime_3	0.0881	0.0816	0.0804
47	traveltime_4	0.0176	0.0204	0.0295
48	studytime_1	0.3789	0.4490	0.2788
49	studytime_2	0.4273	0.4898	0.4933
50	studytime_3	0.1322	0.0612	0.1716

```
51        studytime_4        0.0617       -0.0000        0.0563
52              sex_F        0.5595        0.5714        0.6113
53              sex_M        0.4405        0.4286        0.3887
```

Le tableau ci dessus nous montre les centres de nos clusters pour les élèves de portugais avec l'algorithme du Knn. Nous pouvons constater qu'il y a peu de différences marquantes entre les différents centres ce qui peut être du à des données trop proches entre elles.

[10]:
```
por_original['knn_cluster'] = knn_predictions.labels_

knn_c0 = dataset_for_predictions[por_original['knn_cluster'] == 0]
knn_c1 = dataset_for_predictions[por_original['knn_cluster'] == 1]
knn_c2 = dataset_for_predictions[por_original['knn_cluster'] == 2]


pd.DataFrame({
    "Cluster": ["C0", "C1", "C2"],
    "Nombre d'éléments": [len(knn_c0), len(knn_c1), len(knn_c2)],
})
```

[10]:

	Cluster	Nombre d'éléments
0	C0	227
1	C1	49
2	C2	373

Les données sont bien mieux réparties dans les clusters par rapport à la méthode du Knn. Cependant, il y a toujours un grand écart entre le plus petit cluster (C1) et le plus grand (C2).

Clustering hierarchique Une autre méthode pour clusteriser nos données est le clustering hierarchique.

[11]:
```
from sklearn import cluster

aglo_clusters = cluster.AgglomerativeClustering(cluster_count).
 ↪fit(dataset_for_predictions)

por_original['aglo_cluster'] = aglo_clusters.labels_

aglo_c0 = dataset_for_predictions[por_original['aglo_cluster'] == 0]
aglo_c1 = dataset_for_predictions[por_original['aglo_cluster'] == 1]
aglo_c2 = dataset_for_predictions[por_original['aglo_cluster'] == 2]

aglo_c0_center = aglo_c0.mean(axis=0)
aglo_c1_center = aglo_c1.mean(axis=0)
aglo_c2_center = aglo_c2.mean(axis=0)

pd.DataFrame({
    "Centre de C0": aglo_c0_center,
```

```
    "Centre de C1": aglo_c1_center,
    "Centre de C2": aglo_c2_center,
})
```

[11]:

	Centre de C0	Centre de C1	Centre de C2
age	16.6401	17.2414	17.1132
Medu	2.5331	2.6552	2.3868
Fedu	2.3035	2.6207	2.2358
failures	0.1926	0.4138	0.3113
schoolsup	0.1148	0.0345	0.0755
famsup	0.6051	0.6207	0.6509
paid	0.0661	0.0345	0.0377
activities	0.4883	0.2759	0.5283
nursery	0.8016	0.7931	0.8113
higher	0.9125	0.7586	0.8396
internet	0.7490	0.9310	0.8113
romantic	0.3502	0.4828	0.4245
famrel	4.0000	3.7586	3.6415
freetime	3.2140	3.0345	3.0566
goout	3.1284	3.1034	3.4811
Dalc	1.4163	2.1724	1.7358
Walc	2.1537	2.8276	2.7453
health	3.5389	3.4828	3.5377
absences	1.7607	18.1034	8.9151
school_GP	0.6226	1.0000	0.6981
school_MS	0.3774	0.0000	0.3019
address_R	0.3268	0.2069	0.2170
address_U	0.6732	0.7931	0.7830
famsize_GT3	0.7043	0.6207	0.7264
famsize_LE3	0.2957	0.3793	0.2736
Pstatus_A	0.1051	0.2759	0.1698
Pstatus_T	0.8949	0.7241	0.8302
Mjob_at_home	0.2140	0.2414	0.1698
Mjob_health	0.0837	0.0000	0.0472
Mjob_other	0.3852	0.4138	0.4528
Mjob_services	0.2004	0.2414	0.2453
Mjob_teacher	0.1167	0.1034	0.0849
Fjob_at_home	0.0623	0.0690	0.0755
Fjob_health	0.0370	0.0000	0.0377
Fjob_other	0.5584	0.5172	0.6132
Fjob_services	0.2802	0.3448	0.2547
Fjob_teacher	0.0623	0.0690	0.0189
reason_course	0.4553	0.2759	0.4057
reason_home	0.2140	0.4828	0.2358
reason_other	0.1148	0.0345	0.1132
reason_reputation	0.2160	0.2069	0.2453
guardian_father	0.2529	0.1379	0.1792

guardian_mother	0.7004	0.6897	0.7075
guardian_other	0.0467	0.1724	0.1132
traveltime_1	0.5642	0.6207	0.5472
traveltime_2	0.3288	0.3448	0.3208
traveltime_3	0.0817	0.0345	0.1038
traveltime_4	0.0253	0.0000	0.0283
studytime_1	0.3132	0.4138	0.3679
studytime_2	0.4611	0.5517	0.4906
studytime_3	0.1654	0.0345	0.1038
studytime_4	0.0603	0.0000	0.0377
sex_F	0.5992	0.5862	0.5472
sex_M	0.4008	0.4138	0.4528

De la même manière, interessons nous à la répartition des données parmi nos clusters.

[12]:
```
pd.DataFrame({
    "Cluster": ["C0", "C1", "C2"],
    "Nombre d'éléments": [len(aglo_c0), len(aglo_c1), len(aglo_c2)],
})
```

[12]:

	Cluster	Nombre d'éléments
0	C0	514
1	C1	29
2	C2	106

Les données sont ici très mal réparties entre les clusters, le cluster C0 contenant 79% des données. Les analyses résultantes seront lourdement impactés et une donnée aura bien plus de chance d'être associée à ce cluster plutôt qu'aux autres.

0.4 Evaluation de nos clusters

Nous allons maintenant analyser la qualité de nos clusters.

[13]:
```
from sklearn import metrics

pd.DataFrame({
    "Metric": ["G3 : Homogeneity Coefficient", "G3 : Silhouette Coefficient",␣
 ↪"G1+G2 : Homogeneity Coefficient",
               "G1+G2 : Silhouette Coefficient"],
    "Valeur avec méthode Knn": [
        metrics.homogeneity_score(por_original['knn_cluster'],␣
 ↪por_original['G3']),
        metrics.silhouette_score(dataset_for_predictions, por_original['G3']),
        metrics.homogeneity_score(por_original['knn_cluster'],␣
 ↪por_original['ratio_G1+G2']),
        metrics.silhouette_score(dataset_for_predictions,␣
 ↪por_original['ratio_G1+G2'])
    ],
```

```
    "Valeur avec méthode hierarchique": [
        metrics.homogeneity_score(por_original['aglo_cluster'],␣
↪por_original['G3']),
        metrics.silhouette_score(dataset_for_predictions, por_original['G3']),
        metrics.homogeneity_score(por_original['aglo_cluster'],␣
↪por_original['ratio_G1+G2']),
        metrics.silhouette_score(dataset_for_predictions,␣
↪por_original['ratio_G1+G2'])
    ],
})
```

[13]:
```
                            Metric  Valeur avec méthode Knn  \
0     G3 : Homogeneity Coefficient                   0.0654
1      G3 : Silhouette Coefficient                  -0.1470
2  G1+G2 : Homogeneity Coefficient                   0.0667
3   G1+G2 : Silhouette Coefficient                  -0.2302

   Valeur avec méthode hierarchique
0                            0.0652
1                           -0.1470
2                            0.0680
3                           -0.2302
```

Les résultats sont très similaires avec nos deux méthodes.

Le score d'homogénéité est très mauvais (trop proche de zéro) avec nos deux méthodes de clustering. Nous pouvons donc en déduire que : - Il est possible qu'il y ai eu une erreur dans la manipulation des données - Les élèves sont trop différents dans les jeux de données et que les clusters ne reflètent donc pas la réalité

La silhouette est très mauvaise aussi (inférieur à zéro) ce qui indique que les éléves ne sont pas bien placés dans les différents clusters.

0.5 Partie 3 - Prédiction

[14]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

0.5.1 Pour G3

Pour les élèves de math

[15]:
```
train_data, test_data = train_test_split(mat_formatted, test_size=0.3)

train_data_g3= train_data["G3"]
train_data_for_g3 = train_data.drop(labels=["G3"], axis=1)
test_data_g3 = test_data["G3"]
test_data_for_g3 = test_data.drop(labels=["G3"], axis=1)
```

Méthode Kmean

[16]:
```
knn = KNeighborsClassifier()
knn.fit(train_data_for_g3, train_data_g3)
predictions = knn.predict(test_data_for_g3)

knn_absolute_errors = np.abs(np.subtract(predictions, test_data_g3))
knn_square_errors = np.square(knn_absolute_errors)

knn_mse = np.average(knn_square_errors)
knn_rmse = np.sqrt(knn_mse)
knn_mae = np.average(knn_absolute_errors)
```

Linear regression

[17]:
```
from sklearn.linear_model import LinearRegression

lr_reg = LinearRegression().fit(train_data_for_g3, train_data_g3)
lr_results = lr_reg.predict(test_data_for_g3)

lr_absolute_errors = np.abs(np.subtract(lr_results, test_data_g3))
lr_square_errors = np.square(lr_absolute_errors)

lr_mse = np.average(lr_square_errors)
lr_rmse = np.sqrt(lr_mse)
lr_mae = np.average(lr_absolute_errors)
```

Resultats

[18]:
```
pd.DataFrame({
    "Metriques": ["MAE", "MSE", "RMSE"],
    "Valeurs pour Knn": [knn_mae, knn_mse, knn_rmse],
    "Valeurs pour Regression linéaire": [lr_mae, lr_mse, lr_rmse],
})
```

[18]:

	Metriques	Valeurs pour Knn	Valeurs pour Regression linéaire
0	MAE	1.1345	1.2297
1	MSE	3.9748	3.2915
2	RMSE	1.9937	1.8143

Le tableau ci dessous présentent les valuers des metriques MSE, RMSE et MAE pour les prédictions des algorithmes de regression linéaire. Ces différentes métriques se basent sur les erreurs des différents modes de prédiction. Plus une erreur est proche de zéro, plus la prédictionest juste. A l'inverse, plus l'erreur est grande, plus les erreurs de prédiction sont grandes.

Nous pouvons donc constater que l'algorithme de regression linéaire est bien meilleur que celui du kmean pour calculer les notes des élèves sur notre jeu de données.

Pour les élèves de portugais

[19]:
```
train_data, test_data = train_test_split(por_formatted, test_size=0.3)

train_data_g3 = train_data["G3"]
train_data_for_g3 = train_data.drop(labels=["G3"], axis=1)
test_data_g3 = test_data["G3"]
test_data_for_g3 = test_data.drop(labels=["G3"], axis=1)
```

Méthode Kmean

[20]:
```
knn = KNeighborsClassifier()
knn.fit(train_data_for_g3, train_data_g3)
predictions = knn.predict(test_data_for_g3)

knn_absolute_errors = np.abs(np.subtract(predictions, test_data_g3))
knn_square_errors = np.square(knn_absolute_errors)

knn_mse = np.average(knn_square_errors)
knn_rmse = np.sqrt(knn_mse)
knn_mae = np.average(knn_absolute_errors)
```

Linear regression

[21]:
```
from sklearn.linear_model import LinearRegression

lr_reg = LinearRegression().fit(train_data_for_g3, train_data_g3)
lr_results = lr_reg.predict(test_data_for_g3)

lr_absolute_errors = np.abs(np.subtract(lr_results, test_data_g3))
lr_square_errors = np.square(lr_absolute_errors)

lr_mse = np.average(lr_square_errors)
lr_rmse = np.sqrt(lr_mse)
lr_mae = np.average(lr_absolute_errors)
```

Resultats

[22]:
```
pd.DataFrame({
    "Metriques": ["MAE", "MSE", "RMSE"],
    "Valeurs pour Knn": [knn_mae, knn_mse, knn_rmse],
    "Valeurs pour Regression linéaire": [lr_mae, lr_mse, lr_rmse],
})
```

[22]:

	Metriques	Valeurs pour Knn	Valeurs pour Regression linéaire
0	MAE	0.8462	0.5823
1	MSE	1.8821	0.8581
2	RMSE	1.3719	0.9263

Nous pouvons constater une nouvelle fois que l'algorithme de regression linéaire est bien meilleur que celui du kmean pour calculer les notes des élèves sur notre jeu de données.

0.5.2 Pour ratio_G1+G2

Pour les élèves de math

[23]:
```
train_data, test_data = train_test_split(mat_formatted, test_size=0.3)

train_data_g3 = train_data["ratio_G1+G2"]
train_data_for_g3 = train_data.drop(labels=["ratio_G1+G2", "G1", "G2"], axis=1)
test_data_g3 = test_data["ratio_G1+G2"]
test_data_for_g3 = test_data.drop(labels=["ratio_G1+G2", "G1", "G2"], axis=1)
```

Méthode Kmean

[24]:
```
knn = KNeighborsClassifier()
knn.fit(train_data_for_g3, train_data_g3)
predictions = knn.predict(test_data_for_g3)

knn_absolute_errors = np.abs(np.subtract(predictions, test_data_g3))
knn_square_errors = np.square(knn_absolute_errors)

knn_mse = np.average(knn_square_errors)
knn_rmse = np.sqrt(knn_mse)
knn_mae = np.average(knn_absolute_errors)
```

Linear regression

[25]:
```
from sklearn.linear_model import LinearRegression

lr_reg = LinearRegression().fit(train_data_for_g3, train_data_g3)
lr_results = lr_reg.predict(test_data_for_g3)

lr_absolute_errors = np.abs(np.subtract(lr_results, test_data_g3))
lr_square_errors = np.square(lr_absolute_errors)

lr_mse = np.average(lr_square_errors)
lr_rmse = np.sqrt(lr_mse)
lr_mae = np.average(lr_absolute_errors)
```

Resultats

[26]:
```
pd.DataFrame({
    "Metriques": ["MAE", "MSE", "RMSE"],
    "Valeurs pour Knn": [knn_mae, knn_mse, knn_rmse],
    "Valeurs pour Regression linéaire": [lr_mae, lr_mse, lr_rmse],
})
```

[26]:

	Metriques	Valeurs pour Knn	Valeurs pour Regression linéaire
0	MAE	3.9832	1.8363
1	MSE	26.4202	5.1763
2	RMSE	5.1401	2.2751

Le tableau ci dessous présentent les valuers des metriques MSE, RMSE et MAE pour les prédictions des algorithmes de regression linéaire. Ces différentes métriques se basent sur les erreurs des différents modes de prédiction. Plus une erreur est proche de zéro, plus la prédictionest juste. A l'inverse, plus l'erreur est grande, plus les erreurs de prédiction sont grandes.

Nous pouvons donc constater que l'algorithme de regression linéaire est bien meilleur que celui du kmean pour calculer les notes des élèves sur notre jeu de données.

Pour les élèves de portugais

[27]:
```
train_data, test_data = train_test_split(por_formatted, test_size=0.3)

train_data_g3 = train_data["ratio_G1+G2"]
train_data_for_g3 = train_data.drop(labels=["ratio_G1+G2", "G1", "G2"], axis=1)
test_data_g3 = test_data["ratio_G1+G2"]
test_data_for_g3 = test_data.drop(labels=["ratio_G1+G2", "G1", "G2"], axis=1)
```

Méthode Kmean

[28]:
```
knn = KNeighborsClassifier()
knn.fit(train_data_for_g3, train_data_g3)
predictions = knn.predict(test_data_for_g3)

knn_absolute_errors = np.abs(np.subtract(predictions, test_data_g3))
knn_square_errors = np.square(knn_absolute_errors)

knn_mse = np.average(knn_square_errors)
knn_rmse = np.sqrt(knn_mse)
knn_mae = np.average(knn_absolute_errors)
```

Linear regression

[29]:
```
from sklearn.linear_model import LinearRegression

lr_reg = LinearRegression().fit(train_data_for_g3, train_data_g3)
lr_results = lr_reg.predict(test_data_for_g3)

lr_absolute_errors = np.abs(np.subtract(lr_results, test_data_g3))
lr_square_errors = np.square(lr_absolute_errors)

lr_mse = np.average(lr_square_errors)
lr_rmse = np.sqrt(lr_mse)
lr_mae = np.average(lr_absolute_errors)
```

Resultats

[30]:
```
pd.DataFrame({
    "Metriques": ["MAE", "MSE", "RMSE"],
    "Valeurs pour Knn": [knn_mae, knn_mse, knn_rmse],
```

```
    "Valeurs pour Regression linéaire": [lr_mae, lr_mse, lr_rmse],
})
```

[30]:

	Metriques	Valeurs pour Knn	Valeurs pour Regression linéaire
0	MAE	2.6821	1.7138
1	MSE	13.4000	5.7285
2	RMSE	3.6606	2.3934

Nous pouvons constater une nouvelle fois que l'algorithme de regression linéaire est bien meilleur que celui du kmean pour calculer les notes des élèves sur notre jeu de données.

0.6 Conclusion

Pour chacune de nos situations, la méthode de regression linéaire donne des erreurs plus faibles que l'alagithme du Knn. Cette méthode est donc à privilégier.