etic2-ressources

January 21, 2023

1 Clustering Type de ressources

Nous allons procéder ici aussi à un clustering des établissements sur les colonnes suivantes : - TpRess_ManuelNum : *est-ce que le manuel scolaire est disponible sous forme numérique* - TpRess_AnimScienLogiSimu : *Les animations scientifiques et/ou logiciels de simulation sont l'un des types de ressources mises à la disposition des élèves* - TpRess_Bdd : *Les banques de documents multimédias (vidéos, podcasts, textes, cartes, animations, etc.) sont l'un des types de ressources mises à la disposition des élèves.* - TpRess_LogiOutils : *Les logiciels outils (géométrie, indexage, lexicographie, cartographie, etc.) sont l'un des types de ressources mises à la disposition des élèves* - TpRess_OuvRef : *Les ouvrages de référence interactifs (atlas, dictionnaire, encyclopédie...) sont l'un des types de ressources mises à la disposition des élèves* - TpRess_ResEntrainement : *Les ressources pour s'entraîner sont l'un des types de ressources mises à la disposition des élèves* - TpRess_Autres : *D'autres types de ressources sont mis à la disposition des élèves*

Le but va être de réduire ces 7 colonnes en une seule avec plusieurs modalités, pour faciliter l'analyse globale. Nous allons procèder comme nous avons pu le faire précedemment. Nous allons utiliser l'algorithme de `KMeans` pour procéder à une classification non supervisée pour voir si des groupes se dégagent

1.0.1 *Import des bibliothèques*

[1]:
```
import pandas as pd
import plotly.graph_objects as go
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import plotly.express as px
import prince as pc

# import image module
from IPython.display import Image
import kaleido
import io
from PIL import Image as ImagePIL
```

1.0.2 *Import du fichier*

[2]:
```
df = pd.read_csv('../data/lycee-college/fr-en-etic_2d.csv', sep=';')

#on garde les données les plus récentes, donc 2019
df = df.loc[df["Millesime"] == 2019]

#On consulte les colonnes existantes
df.columns
```

[2]:
```
Index(['Code_UAI', 'Millesime', 'Code_nature', 'nature_uai', 'typ_etab',
       'Academie', 'Departement', 'NbEleve', 'NbEnseignant',
       'SiEtabCentreRessource', 'SiProjetNumEcole', 'SiEntDisponible',
       'SiProjEtabIntegreENT', 'Maint_PersCollect', 'Maint_PrestaExt',
       'Maint_PersonnelEducNatHsEtab', 'Maint_PersonnelEtab',
       'Maint_AutreNeSaitPas', 'Maint_Personne', 'NbRessourceEdit',
       'TpRess_ManuelNum', 'TpRess_AnimScienLogiSimu', 'TpRess_Bdd',
       'TpRess_LogiOutils', 'TpRess_OuvRef', 'TpRess_ResEntrainement',
       'TpRess_Autres', 'TpRess_aucune', 'ServInt_NoteElev', 'ServInt_AbsElev',
       'ServInt_EdtElevCls', 'ServInt_CahierTxt', 'ServInt_DocRessPeda',
       'ServInt_AgdActuEtab', 'ServInt_PlatApp', 'ServInt_Autres',
       'ServInt_aucun', 'NbTerminaux', 'NbTablette', 'NbTablettePC',
       'NbMicroMoins5', 'NbMicroPortable', 'NbPortAffEl', 'NbPortAffEns',
       'NbEleveEqASH', 'NbPosteEqASH', 'NbTBI', 'propClassesAvecTBI',
       'NbVideoProj', 'NbClassMobile', 'NbLecteurMpx', 'NbImpr3D',
       'AccWeb_RTC', 'AccWeb_CableFibreOptique', 'AccWeb_ADSL',
       'AccWeb_AutresHautDebit', 'AccWeb_NeSaitPas', 'AccWeb_Aucun',
       'DebitWeb', 'SiWifi', 'SalleInternet', 'PostesInfoElvHorsCours',
       'SiPareFeuEtab', 'SiOuifiltrageWeb', 'ControlePosteriori',
       'SiCharteUsageWeb', 'Diffch_AnnexeeRI',
       'Diffch_DossierRentreeEnseignants', 'Diffch_CRConseilAdmin',
       'Diffch_DiffusionParents', 'Diffch_Autres', 'AccesParentCharte',
       'ElvAuthentif', 'SiVisioConferenc', 'SiEntUtilise',
       'TypeMatHandi_Tablette', 'TypeMatHandi_OrdiPort', 'TypeMatHandi_LogApp',
       'TypeMatHandi_Autre', 'Code_region', 'Libelle_region'],
      dtype='object')
```

1.0.3 *Traitement du fichier*

On va fixer l'index et ensuite supprimer toutes les colonnes qui seront inutiles pour cette analyse. On supprime également les établissement pour lesquels il y a des `NaN` car cela va poser problème lors de la classification.

[3]:
```
df_ressources = df.set_index('Code_UAI')

#On supprime les colonnes inutiles
df_ressources.drop(columns=['Millesime', 'Code_nature', 'nature_uai',␣
 ↪'typ_etab',
```

```
       'Academie', 'Departement', 'NbEleve', 'NbEnseignant',
       'SiEtabCentreRessource', 'SiProjetNumEcole', 'SiEntDisponible',
       'SiProjEtabIntegreENT', 'Maint_PersCollect', 'Maint_PrestaExt',
       'Maint_PersonnelEducNatHsEtab', 'Maint_PersonnelEtab',
       'Maint_AutreNeSaitPas', 'Maint_Personne', 'NbRessourceEdit',
       'ServInt_NoteElev', 'ServInt_AbsElev',
       'ServInt_EdtElevCls', 'ServInt_CahierTxt', 'ServInt_DocRessPeda',
       'ServInt_AgdActuEtab', 'ServInt_PlatApp', 'ServInt_Autres',
       'ServInt_aucun', 'NbTerminaux', 'NbTablette', 'NbTablettePC',
       'NbMicroMoins5', 'NbMicroPortable', 'NbPortAffEl', 'NbPortAffEns',
       'NbEleveEqASH', 'NbPosteEqASH', 'NbTBI', 'propClassesAvecTBI',
       'NbVideoProj', 'NbClassMobile', 'NbLecteurMpx', 'NbImpr3D',
       'AccWeb_RTC', 'AccWeb_CableFibreOptique', 'AccWeb_ADSL',
       'AccWeb_AutresHautDebit', 'AccWeb_NeSaitPas', 'AccWeb_Aucun',
       'DebitWeb', 'SiWifi', 'SalleInternet', 'PostesInfoElvHorsCours',
       'SiPareFeuEtab', 'SiOuifiltrageWeb', 'ControlePosteriori',
       'SiCharteUsageWeb', 'Diffch_AnnexeeRI',
       'Diffch_DossierRentreeEnseignants', 'Diffch_CRConseilAdmin',
       'Diffch_DiffusionParents', 'Diffch_Autres', 'AccesParentCharte',
       'ElvAuthentif', 'SiVisioConferenc', 'SiEntUtilise',
       'TypeMatHandi_Tablette', 'TypeMatHandi_OrdiPort', 'TypeMatHandi_LogApp',
       'TypeMatHandi_Autre', 'Code_region', 'Libelle_region'], inplace=True)


#On supprime les lignes avec des valeurs manquantes
df_ressources.dropna(inplace=True)
```

On vérifie que tout s'est bien supprimé

[4]:
```
df_ressources.head()
```

[4]:

Code_UAI	TpRess_ManuelNum	TpRess_AnimScienLogiSimu	TpRess_Bdd	TpRess_LogiOutils	TpRess_OuvRef	TpRess_ResEntrainement	TpRess_Autres
0810016C	oui	oui	oui	oui	oui	non	oui
0810026N	non	oui	oui	oui	oui	non	oui
0810041E	oui	non	oui	oui	oui	oui	oui
0810124V	oui	oui	oui	oui	oui	non	oui
0810125W	oui	non	non	oui	non	oui	non

```
          TpRess_aucune
Code_UAI
0810016C           non
0810026N           non
0810041E           non
0810124V           non
0810125W           non
```

On va procéder ensuite au renommage des colonnes

[5]:
```
old_columns = df_ressources.columns

for column in old_columns:
    df_ressources.rename(columns={column: column.replace('TpRess_', '').
 ↪lower()}, inplace=True)

df_ressources.columns
```

[5]:
```
Index(['manuelnum', 'animscienlogisimu', 'bdd', 'logioutils', 'ouvref',
       'resentrainement', 'autres', 'aucune'],
      dtype='object')
```

On va procèder maintenant à la transformation des modalités en valeurs numériques (1 pour “oui” et 0 pour “non”). Cette transformation est nécessaire pour pouvoir faire du clustering et faire une ACP à la fin pour avoir une représentation.

[6]:
```
for column in df_ressources.columns:
    df_ressources[column] = df_ressources[column].replace({'oui': 1, 'non': 0})
    df_ressources[column] = df_ressources[column].astype('float64')

df_ressources.head()
```

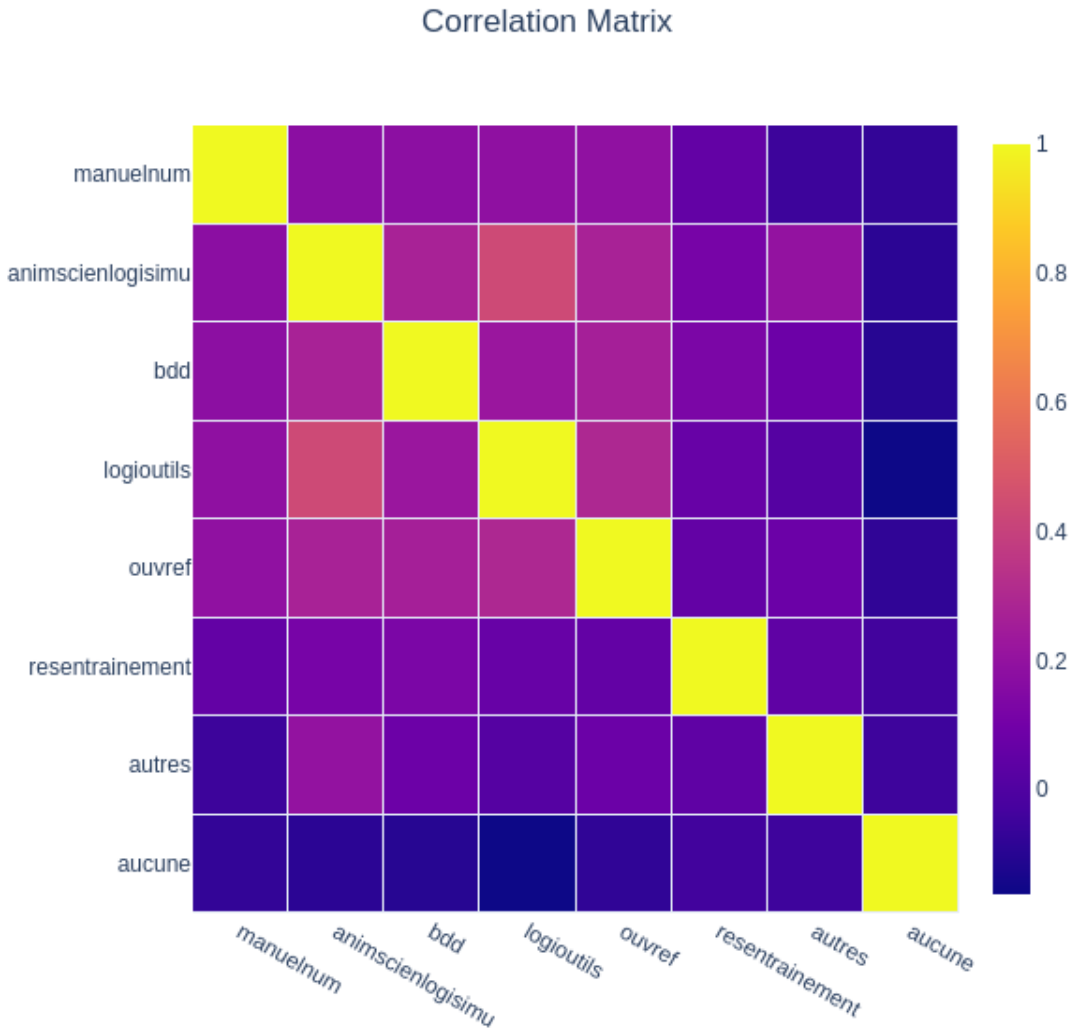
[6]:
```
          manuelnum  animscienlogisimu  bdd  logioutils  ouvref  \
Code_UAI
0810016C        1.0                1.0  1.0         1.0     1.0
0810026N        0.0                1.0  1.0         1.0     1.0
0810041E        1.0                0.0  1.0         1.0     1.0
0810124V        1.0                1.0  1.0         1.0     1.0
0810125W        1.0                0.0  0.0         1.0     0.0

          resentrainement  autres  aucune
Code_UAI
0810016C              0.0     1.0     0.0
0810026N              0.0     1.0     0.0
0810041E              1.0     1.0     0.0
0810124V              0.0     1.0     0.0
0810125W              1.0     0.0     0.0
```

1.0.4 Corrélation

Avant de faire de la classification, nous allons voir si il existe une corrélation entre les différentes variables qu'on a là

[7]:
```
columns_names = df_ressources.columns

heat = go.Heatmap(
    z = df_ressources.corr(),
    x = columns_names,
    y = columns_names,
    xgap = 1, ygap = 1,
    colorbar_thickness=20,
    colorbar_ticklen=3,
    hovertext = df_ressources.corr(),
    hoverinfo='text'
)

layout = go.Layout(
    title_text= "Correlation Matrix",
    title_x = 0.5,
    width=600, height=600,
    xaxis_showgrid=False,
    yaxis_showgrid=False,
    yaxis_autorange='reversed'
)

fig = go.Figure(data=[heat], layout=layout)
Image(fig.to_image(format="png", engine="kaleido", width=600, height=600))
#fig.show()
```

[7]:



On voit qu'il n'y a pas tellement de corrélation entre les variables qu'on a prises ici.

1.0.5 Clustering

On peut maintenant passer au Clustering avec l'algorithme de `KMeans`.

[8]:
```
km_ressources = KMeans(
    n_clusters=4,
    random_state=42,
    init='k-means++',
    max_iter=3600,
    n_init=20
)
```

```
y_km_ressources = km_ressources.fit_predict(df_ressources)

resultat = pd.DataFrame(km_ressources.cluster_centers_, columns=km_ressources.
 ↪feature_names_in_)

resultat
```

[8]:

	manuelnum	animscienlogisimu	bdd	logioutils	ouvref	resentrainement	autres	aucune
0	0.690909	0.927273	0.900000	0.954545	0.800000	1.110223e-16	1.000000e+00	-1.734723e-18
1	0.829114	0.892405	0.867089	0.993671	0.829114	4.556962e-01	6.661338e-16	0.000000e+00
2	0.773585	0.943396	0.896226	0.981132	0.849057	1.000000e+00	1.000000e+00	-1.734723e-18
3	0.365079	0.047619	0.476190	0.571429	0.269841	2.857143e-01	2.698413e-01	1.587302e-02

On va convertir ces valeurs numériques en “oui” et “non” pour pouvoir comprendre le comportement moyen qui se dégage d’un cluster. Cela sera plus facile à comprendre et interprèter

[9]:
```
def change_value(row):
    if(row > 0.5):
        return 1
    else:
        return 0


for column in resultat.columns:
    resultat[column] = resultat[column].apply(change_value)
    resultat[column] = resultat[column].replace({1: 'oui', 0: 'non'})
    resultat[column] = resultat[column].astype(str)


resultat
```

[9]:

	manuelnum	animscienlogisimu	bdd	logioutils	ouvref	resentrainement	autres	aucune
0	oui	oui	oui	oui	oui	non	oui	non
1	oui	oui	oui	oui	oui	non	non	non
2	oui	oui	oui	oui	oui	oui	oui	non
3	non	non	non	oui	non	non	non	non

```
3    non
```

Après plusieurs essais, avec plusieurs valeurs pour le nombre de clusters à trouver, nous avons pu voir que les meilleurs paramètres mis au dessus dans la fonction du KMeans est celui mis actuellement. Il permet de discrétiser en 4 modalités ces 8 colonnes. Si on augmente le nombre de clusters, on a des classes qui se ressemblent, et si on le diminue les classes ne sont pas précises. Nous avons donc ici 4 modalités pour la nouvelle classe "ressources mises en place" : - "*Très bien*" : la classe 1 - "*Bien*" : la classe 3 - "*Assez bien*" : la classe 0 - "*Mauvais*" : la classe 2

On va ensuite procéder à une ACP pour voir si on a une bonne découpe des clusters et avoir une représentation physique qui va plus nous parler.

On ajoute à chaque individu sa classe.

[10]:
```
df_ressources['cluster'] = y_km_ressources
df_ressources['cluster'] = df_ressources['cluster'].astype(str)
df_ressources.head()
```

[10]:

Code_UAI	manuelnum	animscienlogisimu	bdd	logioutils	ouvref	resentrainement	autres	aucune	cluster
0810016C	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	0
0810026N	0.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	0
0810041E	1.0	0.0	1.0	1.0	1.0	1.0	1.0	0.0	2
0810124V	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	0
0810125W	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	3

Après avoir bien mis en forme nos données, on peut passer à une ACP. Nous allons d'abord faire une ACP à l'aide de la classe `PCA` fournie par `sklearn`

[11]:
```
pca_ressources = PCA(n_components=2)

components = pca_ressources.fit_transform(df_ressources.
 ↪drop(columns=['cluster']))

fig = px.scatter(components, x=0, y=1, color=df_ressources["cluster"],␣
 ↪title="Coordonnées des individus", labels={"0": "PC1", "1": "PC2"})
Image(fig.to_image(format="png", engine="kaleido", width=1000, height=600))
#fig.show()
```

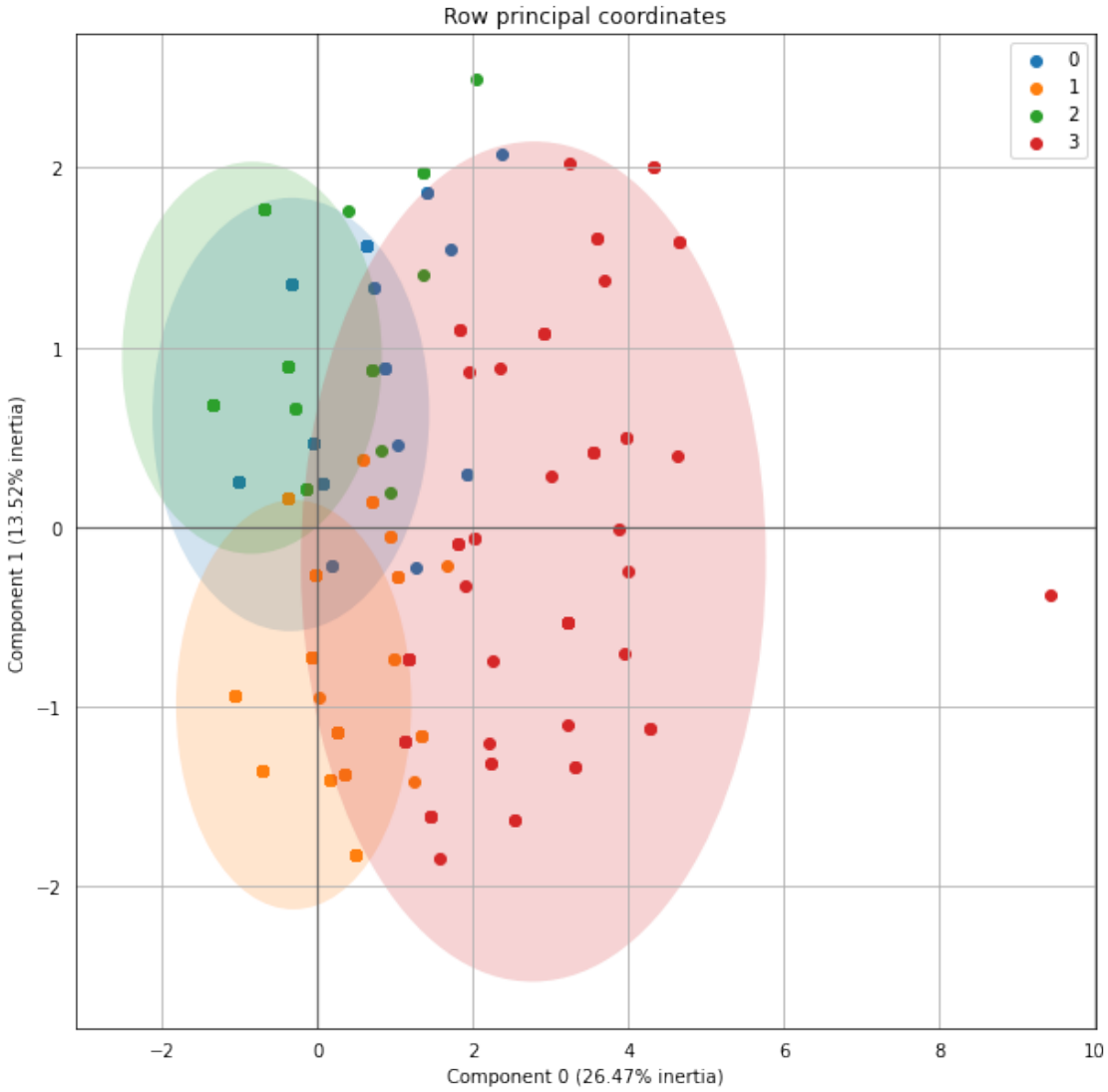
```
print("La variance expliquée par le premier axe est de : ", pca_ressources.
 ↪explained_variance_ratio_[0]*100, "%"," et celle par le second axe est de :␣
 ↪", pca_ressources.explained_variance_ratio_[1]*100, "% ")
print("La variance totale est de : ", pca_ressources.explained_variance_ratio_.
 ↪sum()*100, "%")
```

```
La variance expliquée par le premier axe est de :  26.511371190331083 %  et
celle par le second axe est de :  19.693530326518708 %
La variance totale est de :  46.204901516849795 %
```

On voit que les clusters ici sont mélangés. Ceci peut être expliqué par la variance totale expliquée par ces deux dimensions. On a seulement 46.6 % de l'information. Il faut donc manipuler avec des pincettes ce graphique pour juger le découpage en clusters.

[12]:
```
pca_ressources = pc.PCA(
    n_components=2,
    random_state=0,
    n_iter=100,
    rescale_with_mean=True,
    rescale_with_std=True,
    copy=True,
    engine='auto',
    check_input=True
)

pca_ressources = pca_ressources.fit(df_ressources.drop(columns=['cluster']))
```

[13]:
```
ax_ressources = pca_ressources.plot_row_coordinates(
    df_ressources.drop(columns=['cluster']),
    ax=None,
    figsize=(10, 10),
    x_component=0,
    y_component=1,
    labels=None,
    color_labels=df_ressources['cluster']
)
```

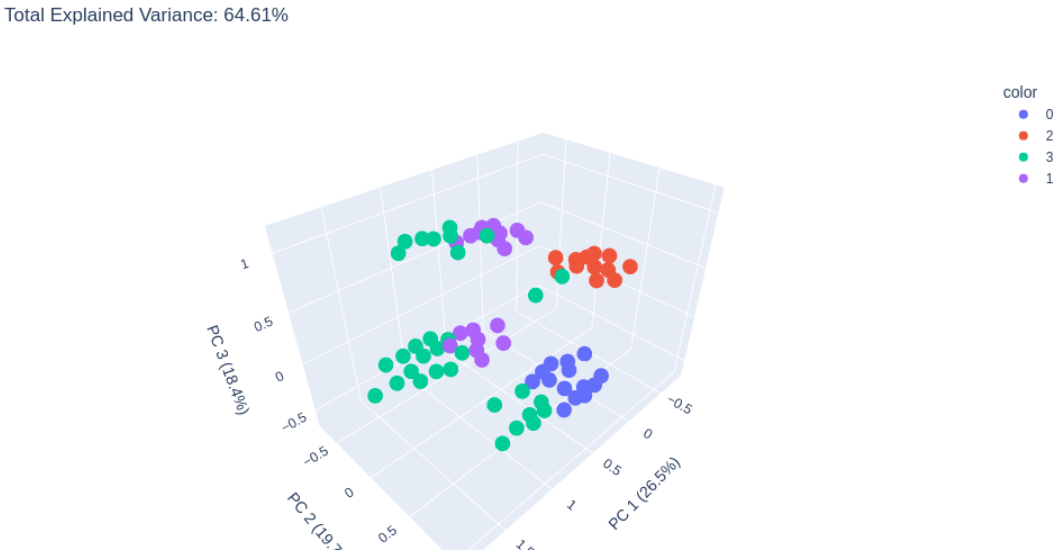


---

[14]:
```
pca = PCA(n_components=3)

components = pca.fit_transform(df_ressources.drop(columns=['cluster']))

total_var = pca.explained_variance_ratio_.sum() * 100

labels = {
    str(i): f"PC {i+1} ({var:.1f}%)"
    for i, var in enumerate(pca.explained_variance_ratio_ * 100)
}
```

```
fig = px.scatter_3d(components, x=0, y=1, z=2, color=df_ressources["cluster"],␣
 ↪title=f'Total Explained Variance: {total_var:.2f}%', labels=labels)
Image(fig.to_image(format="png", engine="kaleido", width=1000, height=600))

#fig.show()
```

[14]:



En passant à une visualisation 3D qui est souvent dure à interpréter, nous passons à une variance totale expliquée de 64.52%. On voit que la séparation des clusters est meilleure qu'en 2 dimensions. On a recouru à un graphique en 3 dimensions pour nous assurer que les clusters ont bien été faits et avoir une représentation graphique

1.0.6 Création de la colonne `Accés aux ressources`

Nous allons maintenant procéder à la création de cette colonne pour ensuite l'enregistrer dans un csv qui pourra être utilisé dans l'analyse générale

[15]:
```
df_ressources_final = pd.DataFrame(df_ressources.cluster.astype(str).
 ↪replace({"0":"Assez bien", "1": "Très bien", "3": "Bien", "2": "Mauvais"}))
df_ressources_final.rename(columns={"cluster": "acces_ressources"},␣
 ↪inplace=True)

df_ressources_final.head()
```

[15]:

Code_UAI	acces_ressources
0810016C	Assez bien

```
0810026N       Assez bien
0810041E          Mauvais
0810124V       Assez bien
0810125W             Bien
```

On va stocker cela dans un fichier csv

[16]:
```
df_ressources_final.to_csv("../data/analyses/acces_ressources.csv", index=True,␣
↪sep=';')
```