Projet Python pour le Data Scientist

November 6, 2020

1 En France, le député assidu présente-t-il un profil type ?

Alors que la société française se caractérise par une grande défiance de la population vis-à-vis de ses élus et responsables politiques, nous nous pencherons dans ce projet sur la question de l'assiduité des députés au cours de leur mandat. Nous verrons notamment grâce à différents outils (boîtes de Tukey, matrice de nuages de points, analyse en composantes principales, clustering, régression linéaire…) si l'on peut dresser un portrait type pour caractériser le député assidu ou absentéiste.

[1]:
```
%matplotlib inline

import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d as plt3d
import seaborn as sns
```

1.1 I. Récupération et préparation des données

1.1.1 1. Importation des données

1.1 Récupération de la base de données principale On récupère sur le site citoyen nosdeputes.fr une base de données synthétisant l'activité parlementaire sur les 12 derniers mois. Pour chaque député, cette table, que nous avons convertie au format .tsv, contient des informations relatives à son état civil, à la circonscription dont il est le représentant, à son éventuel parti politique et surtout à son travail parlementaire (semaines de présence à l'Assemblée nationale, rédaction de rapports, participation à des commissions…).

[2]:
```
# Conversion du fichier tsv en DataFrame.
df = pd.read_csv("../Données/nosdeputes.fr_synthese_2020-10-24.tsv", sep='\t')
df
```

[2]:

	id	nom	nom_de_famille	prenom	sexe	date_naissance	\
0	32	Damien Abad	Abad	Damien	H	1980-04-05	
1	43	Caroline Abadie	Abadie	Caroline	F	1976-09-07	
2	493	Jean-Félix Acquaviva	Acquaviva	Jean-Félix	H	1973-03-19	
3	152	Lénaïck Adam	Adam	Lénaïck	H	1992-02-19	
4	234	Damien Adam	Adam	Damien	H	1989-06-28	

```
..   …                    …             …          …  …             …
534   26        Martine Wonner         Wonner     Martine    F     1964-03-27
535  215       Hubert Wulfranc       Wulfranc      Hubert    H     1956-12-17
536  130        Hélène Zannier        Zannier      Hélène    F     1972-09-19
537   31      Jean-Marc Zulesi         Zulesi   Jean-Marc    H     1988-06-06
538  329      Michel Zumkeller      Zumkeller      Michel    H     1966-01-21

                      lieu_naissance num_deptmt             nom_circo  \
0                        Nîmes (Gard)         01                   Ain
1        Saint-Martin-d'Hères (Isère)         38                 Isère
2                Bastia (Haute-Corse)         2B           Haute-Corse
3    Saint Laurent du Maroni (Guyane)        973                Guyane
4                    Orléans (Loiret)         76        Seine-Maritime
..                                …        …                   …
534                Hayange (Moselle)         67              Bas-Rhin
535            Rouen (Seine-Maritime)         76        Seine-Maritime
536             Saint-Avold (Moselle)         57               Moselle
537      Marseille (Bouches-du-Rhône)         13      Bouches-du-Rhône
538   Belfort (Territoire de Belfort)         90  Territoire de Belfort

     num_circo  … hemicycle_interventions  hemicycle_interventions_courtes  \
0            5  …                    253                              307
1            8  …                     17                                8
2            2  …                    109                                9
3            2  …                      3                                0
4            1  …                     24                                7
..         …  …                      …                              …
534          4  …                     61                                6
535          3  …                    211                              143
536          7  …                      9                                3
537          8  …                     40                               37
538          2  …                     39                               11

     amendements_proposes amendements_signes amendements_adoptes rapports  \
0                      19               1736                  63        0
1                      14                362                 196        0
2                     186               2309                  89        0
3                      15                581                 175        0
4                      74                687                 225        0
..                    …                 …                 …      …
534                   249               1581                 157        0
535                   895               2776                  70        0
536                    25                475                 214        0
537                   105               1111                 279        0
538                    18               1603                  61        0

     propositions_ecrites  propositions_signees questions_ecrites  \
```

```
0                      0                     25                 38
1                      0                      3                  3
2                      0                      4                 22
3                      0                      4                  1
4                      0                      4                 10
..                   …                      …                  …
534                    0                      3                 14
535                    0                      8                 35
536                    0                      4                 12
537                    0                      4                 52
538                    0                      3                 15

     questions_orales
0                   9
1                   0
2                   3
3                   0
4                   2
..                …
534                 3
535                16
536                 0
537                 4
538                 4

[539 rows x 40 columns]
```

1.2 Complétion de la base de données à l'aide de webscraping Ce jeu de données est plutôt complet, mais il lui manque une variable qui pourrait nous intéresser pour notre étude : le statut du député (sortant, élu pour la première fois, ancien député ou arrivé en cours de mandat).

Nous allons donc compléter la table avec la variable "statut" du tableau disponible sur la page http://www2.assemblee-nationale.fr/elections/liste/2017/resultats/RESULTAT.

[3]:
```
# On récupère le code source de la page afin d'en extraire le tableau.
from urllib import request
import bs4

request_text = request.urlopen("http://www2.assemblee-nationale.fr/elections/
↪liste/2017/resultats/RESULTAT").read()
page = bs4.BeautifulSoup(request_text, "html")
tableau_html = page.find("table") # On extrait le tableau d'intérêt de la page␣
↪HTML.
```

Nous avons récupéré le code HTML du tableau, récupérons maintenant ses entêtes.

[4]:
```
entetes = tableau_html.find('thead')
entetes = entetes.find('tr')
entetes = entetes.find_all('th')
entetes = [entete.text.strip() for entete in entetes]
print(entetes)
```

```
['Civ.', 'Nom', 'Prénom', 'Département', 'Circ.', 'Statut', 'Tour', 'Nuance']
```

Puis complétons un dictionnaire avec ses lignes et transformons le en DataFrame.

[5]:
```
dict_tableau = {}
for entete in entetes:
    dict_tableau[entete] = []

corps_tableau = tableau_html.find('tbody')
lignes_tableau = corps_tableau.find_all('tr')
for ligne in lignes_tableau:
    colonnes = ligne.find_all('td')
    for i, element in enumerate(colonnes):
        dict_tableau[entetes[i]].append(element.text.strip())

df1 = pd.DataFrame.from_dict(dict_tableau)
```

On affiche les premières lignes pour s'assurer qu'on obtient bien le tableau souhaité.

[6]:
```
df1.head()
```

[6]:

	Civ.	Nom	Prénom	Département	Circ.	Statut	Tour	Nuance
0	M.	ABAD	DAMIEN	AIN	5	SORTANT	2	LR
1	Mme	ABADIE	CAROLINE	ISERE	8	ELUE POUR LA 1ERE FOIS	2	REM
2	Mme	ABBA	BÉRANGÈRE	HAUTE-MARNE	1	ELUE POUR LA 1ERE FOIS	2	REM
3	M.	ACQUAVIVA	JEAN-FÉLIX	HAUTE-CORSE	2	ELU POUR LA 1ERE FOIS	2	REG
4	M.	ADAM	LÉNAÏCK	GUYANE	2	ELU POUR LA 1ERE FOIS	2	REM

1.1.2 2. Nettoyage des données

Maintenant que nous disposons de nos deux bases, nous allons travailler sur celles-ci de sorte à les rendre plus maniables : nous procédons donc au nettoyage des données.

Dans le tableau webscrapé, conservons uniquement les variables "Nom", "Prénom" et "Statut" et convertissons la casse des modalités et des variables, dans l'optique de les comparer avec celles de la table principale.

[7]:
```
df1 = df1[['Nom','Prénom','Statut']] # On ne conserve que 3 variables.
# On modifie la casse des modalités.
df1["Nom"] = df1["Nom"].str.lower()
df1["Prénom"] = df1["Prénom"].str.lower()
df1["Statut"] = df1["Statut"].str.lower()
df1
```

[7]:

	Nom	Prénom	Statut
0	abad	damien	sortant
1	abadie	caroline	elue pour la 1ere fois
2	abba	bérangère	elue pour la 1ere fois
3	acquaviva	jean-félix	elu pour la 1ere fois
4	adam	lénaïck	elu pour la 1ere fois
..	…	…	…
572	wonner	martine	elue pour la 1ere fois
573	wulfranc	hubert	elu pour la 1ere fois
574	zannier	hélène	elue pour la 1ere fois
575	zulesi	jean-marc	elu pour la 1ere fois
576	zumkeller	michel	sortant

[577 rows x 3 columns]

On concatène le nom et le prénom pour ne conserver que le patronyme complet et le comparer avec celui de l'autre base.

[8]:
```
df1['Nom'] = df1['Prénom'] + ' ' + df1['Nom']
df1 = df1.drop(['Prénom'], axis=1) # On retire donc la variable 'Prénom',␣
↪désormais inutile.
```

Afin d'uniformiser les styles d'écriture, on enlève désormais les mots vides (ici les déterminants "de" fréquemment présents dans les noms de famille), car ils ne sont pas écrits de la même manière dans les deux bases.

[9]:
```
replace_values_ean = {' de ':' ', ' (de) ':' '}

# On crée la fonction de nettoyage qui retire le déterminant "de".
def clean_dataset(data):
    data.replace({'nom': replace_values_ean, 'Nom' :␣
↪replace_values_ean},regex=True,inplace=True)
    return data
```

[10]:
```
# On crée une copie de la table principale pour ne pas modifier la base␣
↪initiale.
df_new = df.copy()
df_new['nom'] = df_new['nom'].str.lower()
clean_dataset(df_new) # On procède au nettoyage sur la base initiale.
```

[10]:

	id	nom	nom_de_famille	prenom	sexe	date_naissance	\
0	32	damien abad	Abad	Damien	H	1980-04-05	
1	43	caroline abadie	Abadie	Caroline	F	1976-09-07	
2	493	jean-félix acquaviva	Acquaviva	Jean-Félix	H	1973-03-19	
3	152	lénaïck adam	Adam	Lénaïck	H	1992-02-19	
4	234	damien adam	Adam	Damien	H	1989-06-28	
..	…	…	…	…	…	…	
534	26	martine wonner	Wonner	Martine	F	1964-03-27	
535	215	hubert wulfranc	Wulfranc	Hubert	H	1956-12-17	
536	130	hélène zannier	Zannier	Hélène	F	1972-09-19	
537	31	jean-marc zulesi	Zulesi	Jean-Marc	H	1988-06-06	
538	329	michel zumkeller	Zumkeller	Michel	H	1966-01-21	

	lieu_naissance	num_deptmt	nom_circo	\
0	Nîmes (Gard)	01	Ain	
1	Saint-Martin-d'Hères (Isère)	38	Isère	
2	Bastia (Haute-Corse)	2B	Haute-Corse	
3	Saint Laurent du Maroni (Guyane)	973	Guyane	
4	Orléans (Loiret)	76	Seine-Maritime	
..	…	…	…	
534	Hayange (Moselle)	67	Bas-Rhin	
535	Rouen (Seine-Maritime)	76	Seine-Maritime	
536	Saint-Avold (Moselle)	57	Moselle	
537	Marseille (Bouches-du-Rhône)	13	Bouches-du-Rhône	
538	Belfort (Territoire de Belfort)	90	Territoire de Belfort	

	num_circo	…	hemicycle_interventions	hemicycle_interventions_courtes	\
0	5	…	253	307	
1	8	…	17	8	
2	2	…	109	9	
3	2	…	3	0	
4	1	…	24	7	
..	…	…	…	…	
534	4	…	61	6	
535	3	…	211	143	
536	7	…	9	3	
537	8	…	40	37	
538	2	…	39	11	

	amendements_proposes	amendements_signes	amendements_adoptes	rapports	\
0	19	1736	63	0	
1	14	362	196	0	
2	186	2309	89	0	
3	15	581	175	0	
4	74	687	225	0	
..	…	…	…	…	
534	249	1581	157	0	

```
535                  895               2776                  70        0
536                   25                475                 214        0
537                  105               1111                 279        0
538                   18               1603                  61        0

     propositions_ecrites  propositions_signees questions_ecrites  \
0                       0                    25                38
1                       0                     3                 3
2                       0                     4                22
3                       0                     4                 1
4                       0                     4                10
..                    …                     …                 …
534                     0                     3                14
535                     0                     8                35
536                     0                     4                12
537                     0                     4                52
538                     0                     3                15

     questions_orales
0                   9
1                   0
2                   3
3                   0
4                   2
..                …
534                 3
535                16
536                 0
537                 4
538                 4

[539 rows x 40 columns]
```

[11]:
```
clean_dataset(df1) # On procède au nettoyage sur la table webscrapée.
```

[11]:
```
                     Nom                 Statut
0             damien abad                sortant
1         caroline abadie  elue pour la 1ere fois
2          bérangère abba  elue pour la 1ere fois
3    jean-félix acquaviva   elu pour la 1ere fois
4            lénaïck adam   elu pour la 1ere fois
..                     …                     …
572       martine wonner  elue pour la 1ere fois
573      hubert wulfranc   elu pour la 1ere fois
574       hélène zannier  elue pour la 1ere fois
575     jean-marc zulesi   elu pour la 1ere fois
576     michel zumkeller                sortant
```

```
[577 rows x 2 columns]
```

On élabore maintenant une fonction qui retire les accents.

[12]:
```
import unicodedata,string
import unidecode

def remove_accent(s) :
    return ''.join((c for c in unicodedata.normalize('NFD', s) if unicodedata.
 ↪category(c) != 'Mn'))
```

[13]:
```
# Retrait des accents sur tous les noms complets des députés dans les deux␣
 ↪bases.
df_new["nom"]=df_new["nom"].map(lambda x: remove_accent(x))
df1["Nom"]=df1["Nom"].map(lambda x: remove_accent(x))
```

On peut désormais joindre les deux tables pour ajouter à la table initiale la variable portant sur le statut du député.

[14]:
```
df_work = pd.merge(df_new, df1, how='left', left_on='nom', right_on='Nom')
```

[15]:
```
df_work = df_work.drop('Nom', axis=1) # On retire la variable "Nom" qui fait␣
 ↪désormais doublon.
```

On va remplacer les valeurs manquantes pour la variable “statut” par la modalité “arrivé en cours de mandat”. En effet, cela concerne des députés présents dans la base mis à jour mais non présents dans la base webscrapée qui correspond aux députés élus en 2017 ; ils sont donc arrivés au cours des trois dernières années.

[16]:
```
for i in range(len(df_work)) :
    if df_work['Statut'].isnull()[i] == True :
        df_work['Statut'][i] = 'arrive en cours de mandat'
```

On crée désormais une variable âge à partir de la variable date_naissance. Cela va nous permettre de répartir les députés en tranches d’âge de façon à analyser l’influence de l’âge sur l’assiduité.

[17]:
```
import datetime

df_work['age'] = 0 # On initialise à 0 la valeur de la variable age pour chaque␣
 ↪individu.
adj = datetime.date.today()
for i in range(len(df_work)) :
    date = datetime.datetime.strptime(df_work["date_naissance"][i], '%Y-%m-%d')
    df_work['age'][i] = adj.year - date.year - ((adj.month, adj.day) < (date.
 ↪month, date.day))
```

```
    # On calcule la différence entre l'année actuelle et l'année de naissance␣
 ↪de chaque député, en corrigeant d'une unité si l'anniversaire n'est pas␣
 ↪encore arrivé.

df_work = df_work.sort_values("age") # On réordonne la base dans l'ordre␣
 ↪croissant de l'âge des députés.

# On répartit maintenant les députés dans différentes tranches d'âge.
df_work['tranche_age'] = 0
for i in range(len(df_work)) :
    if 20 <= df_work['age'][i] < 30 :
        df_work['tranche_age'][i] = '20-30 ans'
    if 30 <= df_work['age'][i] < 40 :
        df_work['tranche_age'][i] = '30-40 ans'
    if 40 <= df_work['age'][i] < 50 :
        df_work['tranche_age'][i] = '40-50 ans'
    if 50 <= df_work['age'][i] < 60 :
        df_work['tranche_age'][i] = '50-60 ans'
    if 60 <= df_work['age'][i] < 70 :
        df_work['tranche_age'][i] = '60-70 ans'
    if df_work['age'][i] >= 70 :
        df_work['tranche_age'][i] = '+ de 70 ans'
```

On poursuit le nettoyage en remplacant la modalité “0” de la variable profession par la modalité “Aucune”, pour des questions de lisibilité.

[18]:
```
for i in range(len(df_work)) :
    if df_work['profession'][i] == '0' :
        df_work['profession'][i] = 'Aucune'
```

On peut maintenant analyser les variables présentes dans la base définitive df_work afin de sélectionner celles que nous pouvons éliminer dans le cadre de notre étude.

[19]:
```
df_work.columns # On affiche la liste des variables.
```

[19]:
```
Index(['id', 'nom', 'nom_de_famille', 'prenom', 'sexe', 'date_naissance',
       'lieu_naissance', 'num_deptmt', 'nom_circo', 'num_circo',
       'mandat_debut', 'mandat_fin', 'ancien_depute', 'groupe_sigle',
       'parti_ratt_financier', 'sites_web', 'emails', 'anciens_mandats',
       'profession', 'place_en_hemicycle', 'url_an', 'id_an', 'slug',
       'url_nosdeputes', 'url_nosdeputes_api', 'nb_mandats', 'twitter',
       'semaines_presence', 'commission_presences', 'commission_interventions',
       'hemicycle_interventions', 'hemicycle_interventions_courtes',
       'amendements_proposes', 'amendements_signes', 'amendements_adoptes',
       'rapports', 'propositions_ecrites', 'propositions_signees',
       'questions_ecrites', 'questions_orales', 'Statut', 'age',
       'tranche_age'],
```

```
      dtype='object')
```

[20]:
```
# On retire les variables que l'on juge inutiles pour notre étude.
df_work = df_work.drop(['id', 'nom_de_famille', 'prenom', 'date_naissance',␣
↪'lieu_naissance',
            'mandat_debut', 'mandat_fin', 'ancien_depute',␣
↪'parti_ratt_financier', 'sites_web',
            'emails', 'anciens_mandats', 'place_en_hemicycle', 'url_an',
            'id_an', 'slug', 'url_nosdeputes', 'url_nosdeputes_api',␣
↪'twitter'], axis=1)
```

[21]:
```
df_work.head() # On regarde les premières lignes de la base de travail␣
↪définitive.
```

[21]:

	nom	sexe	num_deptmt	nom_circo	num_circo	groupe_sigle	\
383	ludovic pajot	H	62	Pas-de-Calais	10	NI	
139	typhanie degois	F	73	Savoie	1	LREM	
235	pierre henriet	H	85	Vendée	5	LREM	
3	lenaick adam	H	973	Guyane	2	LREM	
375	mickael nogal	H	31	Haute-Garonne	4	LREM	

	profession	nb_mandats	semaines_presence	\
383	Aucune	1	23	
139	Juriste	1	21	
235	Professeur du secondaire et technique	1	31	
3	Cadre supérieur (entreprises privée)	1	8	
375	Autre profession libérale	1	29	

	commission_presences	…	amendements_signes	amendements_adoptes	\
383	15	…	327	1	
139	23	…	839	207	
235	26	…	382	188	
3	5	…	581	175	
375	30	…	362	176	

	rapports	propositions_ecrites	propositions_signees	questions_ecrites	\
383	0	0	2	54	
139	0	0	3	36	
235	0	0	3	9	
3	0	0	4	1	
375	0	0	4	0	

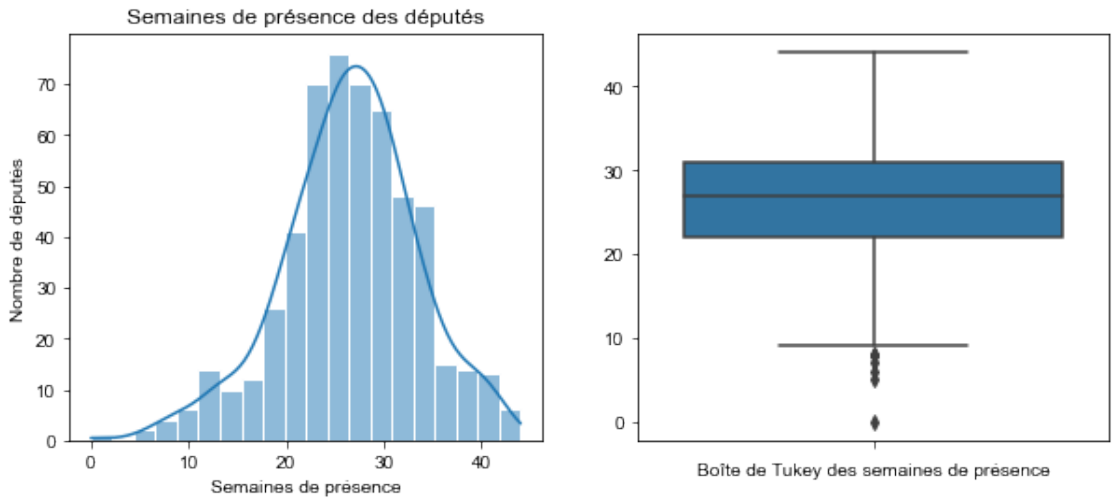
	questions_orales	Statut	age	tranche_age
383	5	elu pour la 1ere fois	26	20-30 ans
139	1	elue pour la 1ere fois	27	20-30 ans
235	0	elu pour la 1ere fois	28	20-30 ans
3	0	elu pour la 1ere fois	28	20-30 ans

```
375                 2    elu pour la 1ere fois    29    20-30 ans

[5 rows x 24 columns]
```

1.2 II. Analyse descriptive

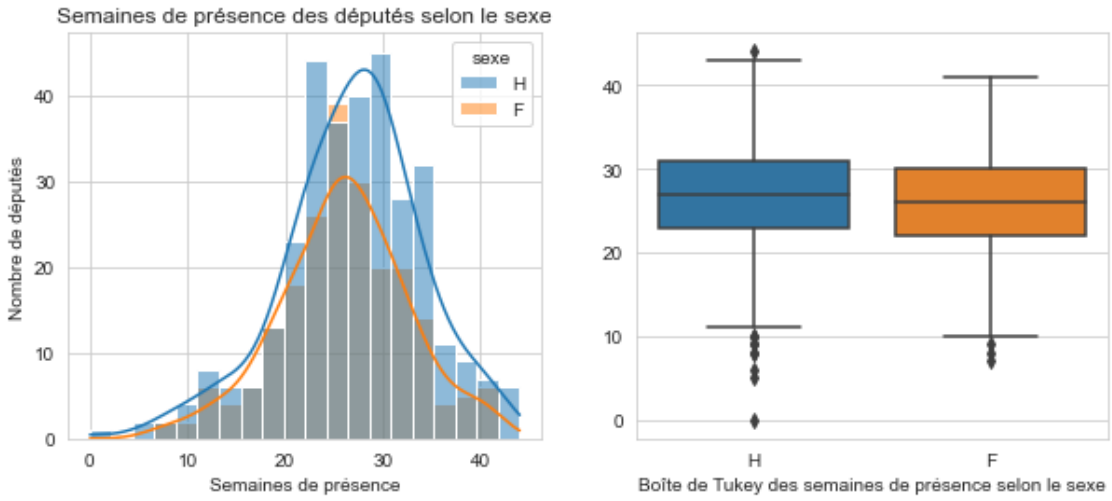
On procède à une analyse descriptive des données afin d'identifier les premières grandes tendances qui guideront notre travail. Les boîtes à moustaches (ou boîtes de Tukey, dites boxplots en anglais) permettent par exemple d'étudier les statistiques liées aux semaines de présence des députés à l'Assemblée nationale, en distinguant selon le sexe, le parti politique ou encore l'âge. On utilise ici la librairie seaborn.

[22]:
```
fig, ax = plt.subplots(1, 2, figsize=(10,4))
sns.set_style("whitegrid")
sns.histplot(data=df_work, x="semaines_presence", kde=True, ax=ax[0])
ax[0].set_xlabel('Semaines de présence')
ax[0].set_ylabel('Nombre de députés')
ax[0].set_title('Semaines de présence des députés')
sns.boxplot(y="semaines_presence",data=df_work, ax=ax[1])
ax[1].set_xlabel("Boîte de Tukey des semaines de présence")
ax[1].set_ylabel('')
plt.show()
```

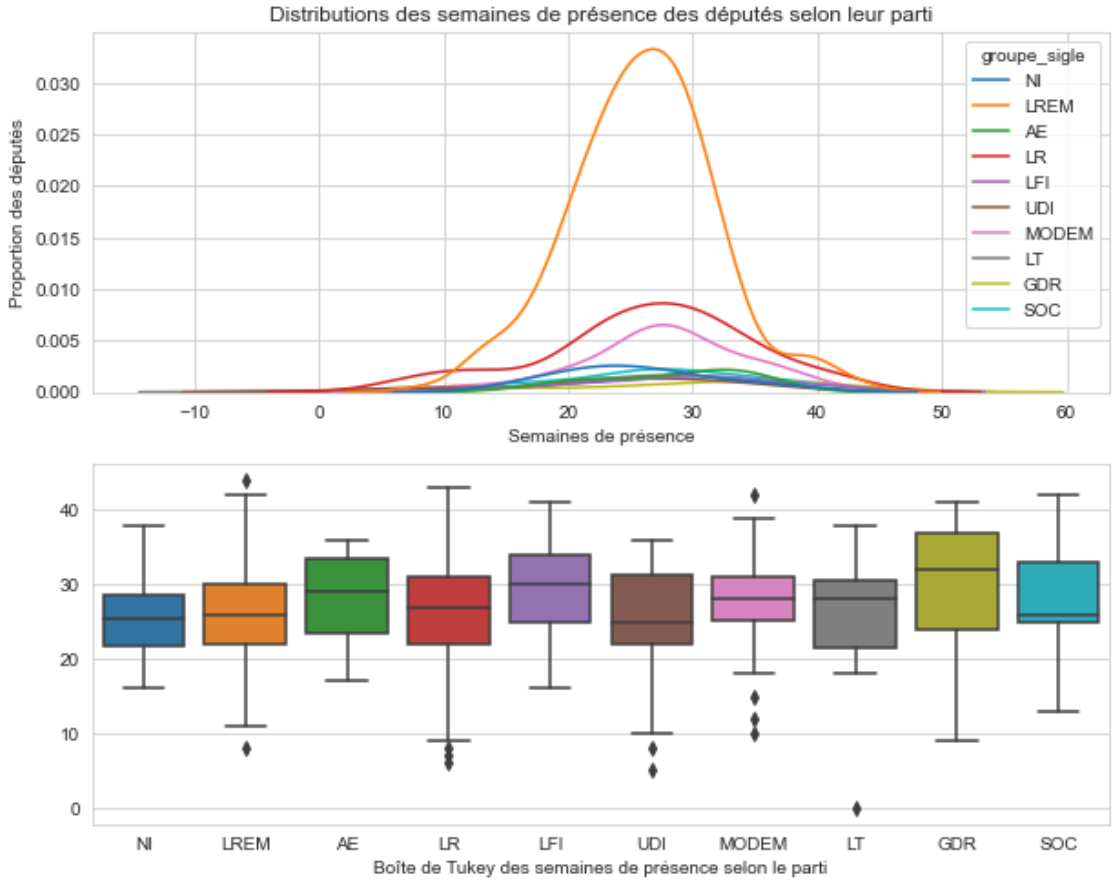


[23]:
```
fig, ax = plt.subplots(1, 2, figsize=(10,4))
sns.set_style("whitegrid")
sns.histplot(data=df_work, x="semaines_presence", hue="sexe", kde=True,␣
↪ax=ax[0])
ax[0].set_xlabel('Semaines de présence')
ax[0].set_ylabel('Nombre de députés')
ax[0].set_title('Semaines de présence des députés selon le sexe')
```

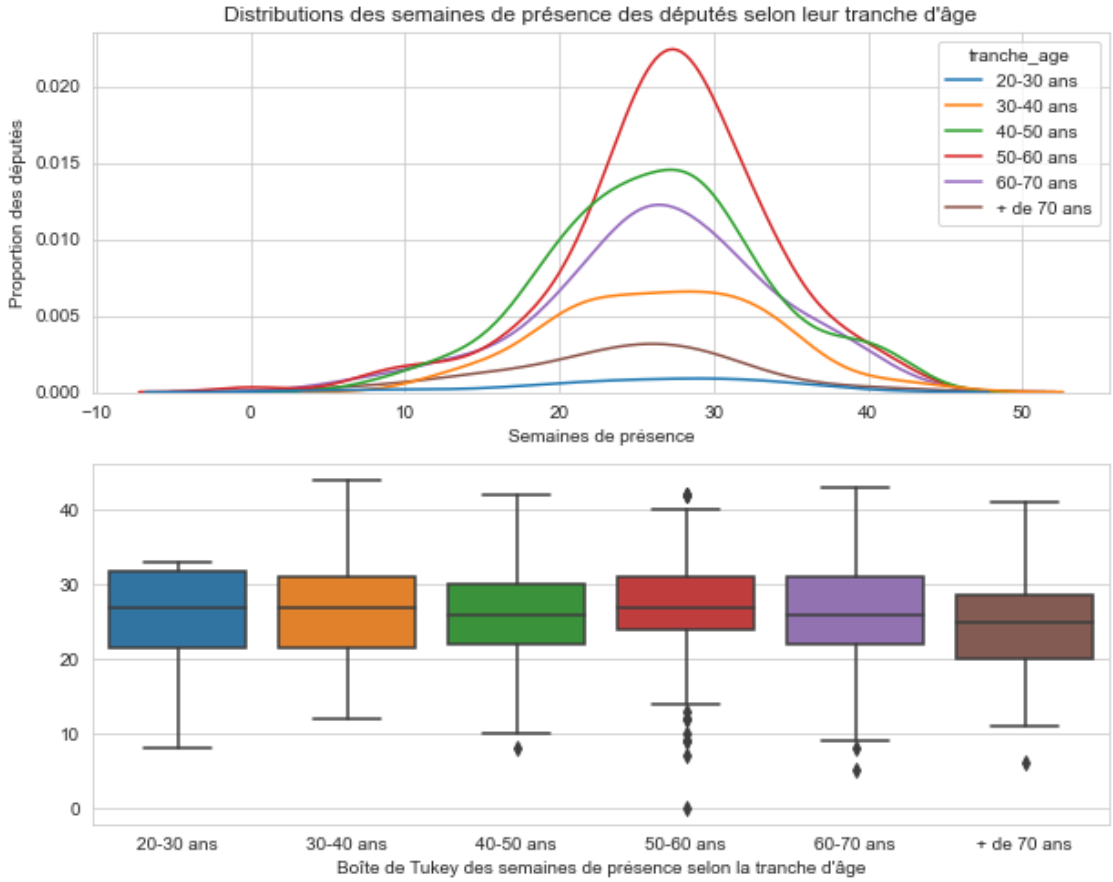
```
sns.boxplot(x="sexe", y="semaines_presence", data=df_work, ax=ax[1])
ax[1].set_xlabel("Boîte de Tukey des semaines de présence selon le sexe")
ax[1].set_ylabel('')
plt.show()
```



[24]:
```
fig, ax = plt.subplots(2, 1, figsize=(10,8))
sns.set_style("whitegrid")
sns.kdeplot(data=df_work, x="semaines_presence", hue="groupe_sigle", ax=ax[0])
ax[0].set_title('Distributions des semaines de présence des députés selon leur␣
↪parti')
ax[0].set_xlabel('Semaines de présence')
ax[0].set_ylabel('Proportion des députés')
sns.boxplot(x="groupe_sigle", y="semaines_presence", data=df_work, ax=ax[1])
ax[1].set_xlabel("Boîte de Tukey des semaines de présence selon le parti")
ax[1].set_ylabel("")
plt.show()
```



[25]:
```
fig, ax = plt.subplots(2, 1, figsize=(10,8))
sns.set_style("whitegrid")
sns.kdeplot(data=df_work, x="semaines_presence", hue="tranche_age", ax=ax[0])
ax[0].set_title("Distributions des semaines de présence des députés selon leur␣
 ↪tranche d'âge")
ax[0].set_xlabel('Semaines de présence')
ax[0].set_ylabel('Proportion des députés')
sns.boxplot(x="tranche_age", y="semaines_presence", data=df_work, ax=ax[1])
ax[1].set_xlabel("Boîte de Tukey des semaines de présence selon la tranche␣
 ↪d'âge")
ax[1].set_ylabel("")
plt.show()
```



L'observation de ces boîtes à moustaches donne à voir quelques premières tendances : - certains députés se démarquent de l'immense majorité des autres députés par un très fort absentéisme, comme le montrent les valeurs extrêmes du premier graphique ; - globalement, les hommes sont légèrement plus présents à l'Assemblée nationale que les femmes ; - il y a d'assez grandes disparités entre les différents partis ; - les députés les plus âgés semblent être les moins assidus, tandis que les 50-60 ans sont visiblement la génération la plus assidue.

1.3 III. Modélisation

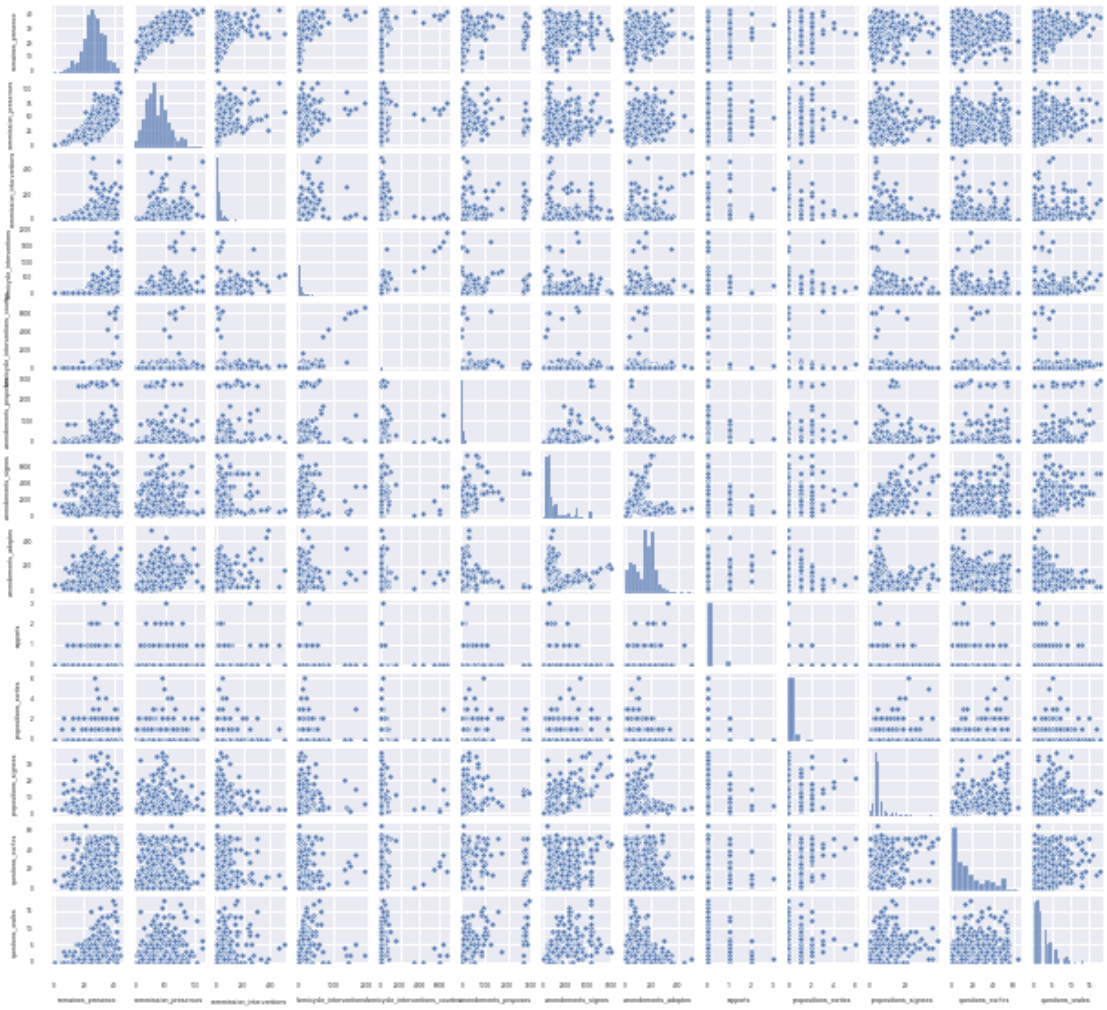
1.3.1 3.1 Liens entre les différentes variables d'assiduité

Nous avons jusqu'à présent étudié uniquement le nombre de semaines de présence des députés sur les bancs de l'Assemblée, sans nous préoccuper des autres indicateurs d'assiduité. Nous allons nous demander dans quelle mesure la présence est suffisante pour révéler l'assiduité, notamment en étudiant les corrélations entre les différents indicateurs. Nous effectuerons surtout une Analyse en Composantes Principales (ACP) sur ces indicateurs, en espérant que le premier axe explique une grande part de la variance et puisse s'interpréter comme une variable d'assiduité.

Les indicateurs d'assiduité que nous avons à disposition sont 'semaines_presence', 'commission_presences', 'commission_interventions', 'hemicycle_interventions','hemicycle_interventions_courtes', 'amendements_proposes', 'amende-

ments_signes', 'amendements_adoptes', 'rapports', 'propositions_ecrites', 'propositions_signees','questions_ecrites' et 'questions_orales'.

[26]:
```
# On trace la matrice des nuages de points afin de percevoir les premiers liens␣
↪entre les variables d'assiduité.
sns.set(font_scale=0.3)
sns.pairplot(df_work[['semaines_presence', 'commission_presences',␣
↪'commission_interventions',␣
↪'hemicycle_interventions','hemicycle_interventions_courtes',␣
↪'amendements_proposes', 'amendements_signes', 'amendements_adoptes',␣
↪'rapports', 'propositions_ecrites',␣
↪'propositions_signees','questions_ecrites', 'questions_orales']], height=0.
↪6, markers=".", aspect=1.1)
plt.show()
```



1.3.2 3.2 Normalisation des données

Afin d'éviter que des variables l'emportent sur d'autres par des effets d'échelle, nous allons normaliser (centrer et réduire) les variables sur lesquelles nous feront l'ACP.

[27]:
```
# On crée une nouvelle table avec le numéro du député en index et les variables␣
↪d'assiduité.
df_acp = df_work[['semaines_presence', 'commission_presences',␣
↪'commission_interventions',␣
↪'hemicycle_interventions','hemicycle_interventions_courtes',␣
↪'amendements_proposes', 'amendements_signes', 'amendements_adoptes',␣
↪'rapports', 'propositions_ecrites',␣
↪'propositions_signees','questions_ecrites', 'questions_orales']]
```

[28]:
```
import sklearn # On importe scikit-learn.
# On centre et réduit les variables de manière à effectuer une ACP normée.
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
Z = sc.fit_transform(df_acp)
Z
```

[28]:
```
array([[-4.99136041e-01, -1.17818822e+00, -4.25038063e-01, …,
        -6.82286754e-01,  2.36772151e+00,  6.48408303e-01],
       [-7.86259121e-01, -7.67223361e-01, -4.76110854e-01, …,
        -5.05906629e-01,  1.23583513e+00, -6.12158245e-01],
       [ 6.49356279e-01, -6.13111538e-01, -4.42062327e-01, …,
        -5.05906629e-01, -4.61994440e-01, -9.27299882e-01],
       …,
       [-4.99136041e-01, -4.07629108e-01,  5.68528285e-04, …,
         7.28754245e-01,  1.42448286e+00,  3.33266666e-01],
       [ 3.62233199e-01,  1.57447577e-01,  1.36762638e-01, …,
         2.13979524e+00, -7.13524747e-01,  1.27869158e+00],
       [-2.93968222e+00, -1.69189430e+00, -5.78256436e-01, …,
         2.13979524e+00,  9.84304822e-01, -9.27299882e-01]])
```

On vérifie que les moyennes sont nulles et les écarts-types unitaires ; c'est bien le cas : la normalisation a bien été effectuée.

[29]:
```
print(np.mean(Z,axis=0))
print(np.std(Z,axis=0,ddof=0))
```

```
[-5.43782706e-17 -6.26174025e-17  1.31826111e-17 -4.44913123e-17
  4.94347914e-18 -6.59130553e-18 -3.95478332e-17  9.22782774e-17
 -2.14217430e-17  5.76739234e-17  9.22782774e-17  4.28434859e-17
 -9.88695829e-18]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

1.3.3 3.3 Analyse en Composantes Principales

Les données étant centrées et réduites, nous pouvons désormais débuter l'ACP.

[30]:
```
from sklearn.decomposition import PCA
# Instanciation de l'objet PCA
acp = PCA(svd_solver='full')
print(acp)
```

```
PCA(svd_solver='full')
```

On remarque que le nombre de composantes n'est pas spécifié, il correspond donc par défaut au nombre de variables, à savoir 13 (ce qu'on vérifie dans la cellule suivante). Nous stockons maintenant les coordonnées factorielles dans la variable coord grâce à la fonction fit_transform().

[31]:
```
# Calcul des coordonnées factorielles
coord = acp.fit_transform(Z)
# Vérification du nombre de composantes principales
print(acp.n_components_)
```

```
13
```

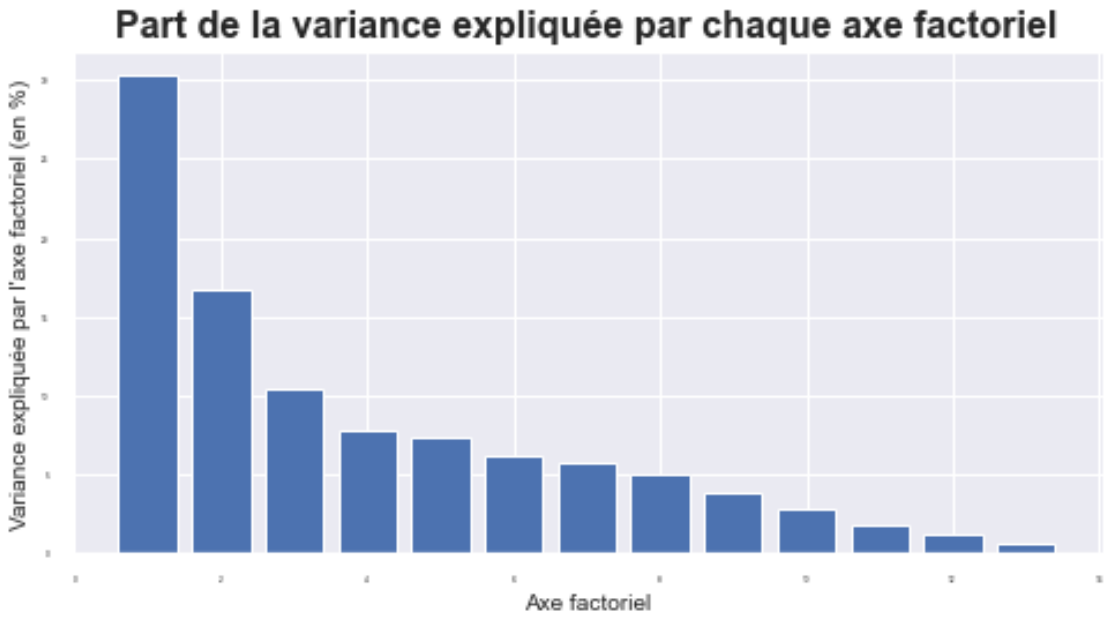
Désormais, nous allons afficher le pourcentage de la variance expliqué par chaque axe factoriel.

[32]:
```
print(acp.explained_variance_ratio_)
```

```
[0.30292818 0.16774724 0.10467894 0.07784485 0.07354728 0.06130974
 0.0578616  0.04991936 0.03846265 0.02819897 0.01826355 0.01285006
 0.00638759]
```

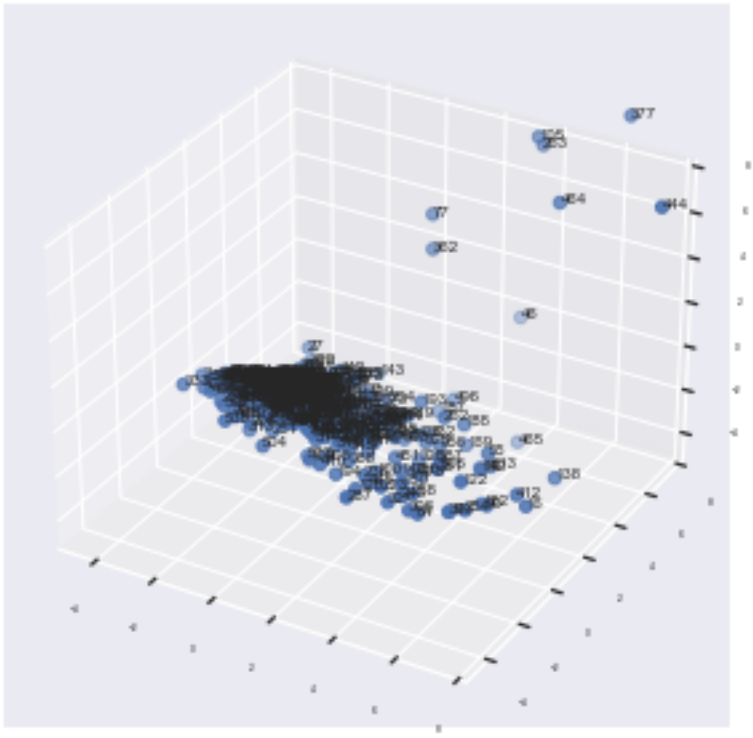
On remarque que le premier axe explique environ 30% de l'information disponible, et les trois premiers près de 57 % : les autres axes ne semblent à première vue pas complètement anecdotiques. Nous allons donc tracer l'éboulis des valeurs propres (qui correspondent aussi à la variance expliquée par chaque axe) afin d'appliquer la méthode dite du coude.

[33]:
```
plt.figure(figsize=(8,4))
plt.bar(np.arange(1,acp.n_components_+1),acp.explained_variance_ratio_*100)
plt.title("Part de la variance expliquée par chaque axe factoriel", size=16,␣
 ↪fontweight='bold')
plt.ylabel("Variance expliquée par l'axe factoriel (en %)", size=10)
plt.xlabel("Axe factoriel", size=10)
plt.show()
```



Au vu du diagramme ci-dessus, nous allons choisir de garder les trois premières composantes factorielles des individus. Nous les représentons dans la figure en trois dimensions ci-dessous.

[34]:
```
fig = plt.figure(figsize=(8,5))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlim(-5,8)
ax.set_ylim(-5,8)
ax.set_zlim(-5,8)

for i in range(len(coord[:,0])):
    ax.text(coord[i,0], coord[i,1], coord[i,2], str(i), size=6)

ax.scatter(coord[:,0], coord[:,1], coord[:,2])
plt.show()
```



Nous pouvons constater que le nuage de points se compose de deux groupes : l'un quasiment aplati dans le plan (Oxy), qui est le premier plan factoriel, l'autre ayant des cotes elevées.

Afin d'interpréter les différents axes, calculons la matrice des corrélations des variables.

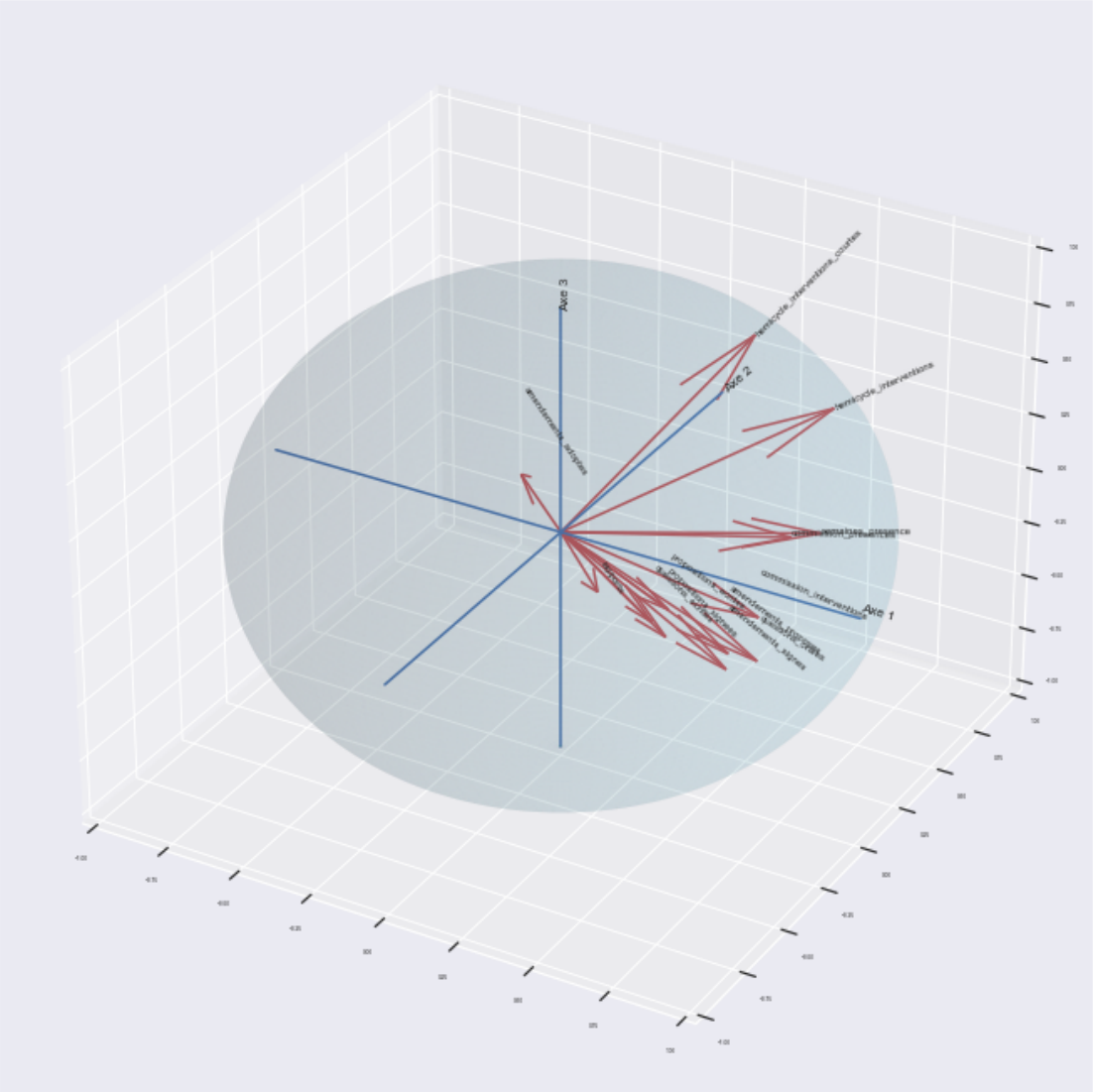
[35]:
```
n = len(df_acp)
p = len(df_acp.columns)

eigval = (n-1)/n*acp.explained_variance_
sqrt_eigval = np.sqrt(eigval)
corvar = np.zeros((p,p))
for k in range(p):
    corvar[:,k] = acp.components_[k,:] * sqrt_eigval[k]

print(pd.DataFrame({'Variable':df_acp.columns,'Cor_axe_1':corvar[:
 ↪,0],'Cor_axe_2':corvar[:,1], 'Cor_axe_3':corvar[:,2]}))
```

	Variable	Cor_axe_1	Cor_axe_2	Cor_axe_3
0	semaines_presence	0.557067	0.600929	-0.191771
1	commission_presences	0.425776	0.662578	-0.302692
2	commission_interventions	0.496364	0.325125	-0.420058
3	hemicycle_interventions	0.648652	0.488061	0.485997
4	hemicycle_interventions_courtes	0.399350	0.453602	0.745994
5	amendements_proposes	0.695054	-0.235665	-0.117199

```
6                amendements_signes    0.763213  -0.375204  -0.062249
7              amendements_adoptes   -0.428367   0.497853  -0.223568
8                          rapports    0.027278   0.174714  -0.383634
9            propositions_ecrites    0.495685  -0.237651   0.004301
10           propositions_signees    0.587427  -0.415955   0.040091
11              questions_ecrites    0.497084  -0.334478   0.013451
12               questions_orales    0.729486  -0.121882  -0.216111
```

Puis représentons ces corrélations sur la sphère unité de $\mathbb{R}^3$.

[36]:
```
r = 1
pi = np.pi
cos = np.cos
sin = np.sin
phi, theta = np.mgrid[0.0:pi:100j, 0.0:2.0*pi:100j]
x = r*sin(phi)*cos(theta)
y = r*sin(phi)*sin(theta)
z = r*cos(phi)

fig = plt.figure(figsize=(12,12))
ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(x, y, z,  rstride=1, cstride=1, color='c', alpha=0.1,␣
 ↪linewidth=0)

ax.add_line(plt3d.art3d.Line3D([-1,1], [0,0], [0,0]))
ax.add_artist(plt3d.art3d.Text3D(x=1, y=0, z=0, text='Axe 1', zdir='x', size=8))

ax.add_line(plt3d.art3d.Line3D([0,0], [-1,1], [0,0]))
ax.add_artist(plt3d.art3d.Text3D(x=0, y=1, z=0, text='Axe 2', zdir='y', size=8))

ax.add_line(plt3d.art3d.Line3D([0,0], [0,0], [-1,1]))
ax.add_artist(plt3d.art3d.Text3D(x=0, y=0, z=1, text='Axe 3', zdir='z', size=8))

ax.quiver(np.zeros(p), np.zeros(p), np.zeros(p), corvar[:,0], corvar[:,1],␣
 ↪corvar[:,2], color='r')
for i in range(p):
    ax.add_artist(plt3d.art3d.Text3D(x=corvar[i,0], y=corvar[i,1],␣
 ↪z=corvar[i,2], text=df_acp.columns[i], zdir=(corvar[i,0], corvar[i,1],␣
 ↪corvar[i,2]), size=6))


ax.set_xlim([-1,1])
ax.set_ylim([-1,1])
ax.set_zlim([-1,1])

plt.show()
```



On constate que le troisième axe différencie essentiellement les députés effectuant de nombreuses interventions (courtes ou non) dans l'hémicycle. Le groupe de député se situant dans le groupe "en haut" sur le nuage de point est donc celui des députés les plus actifs en séance.

1.3.4 3.4 Clustering

Afin de déterminer des groupes de députés en fonction de leur assiduité, on va désormais procéder à un clustering, en s'appuyant sur la méthode des k-moyennes (k-means), très répandue en partitionnement des données. Cette étape va nous permettre de constituer des groupes de députés d'assiduité comparable ; nous dégagerons ensuite les caractéristiques partagées par les députés appartenant au même cluster.

[37]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score # On importe la métrique␣
 ↪"silhouette" qui nous servira à déterminer le nombre de clusters optimal.
```

[38]:
```
# On applique la méthode des k-means sur les variables d'assiduité centrées et␣
 ↪réduites, en supposant pour l'instant par défaut qu'il existe 3 clusters.
kmeans = KMeans(n_clusters=3)
kmeans.fit(df_acp)

# On trie les index en fonction des différents groupes de députés (ie des␣
 ↪clusters).
idk = np.argsort(kmeans.labels_)

# Affichage des observations et leurs groupes
print(pd.DataFrame(df_acp.index[idk],kmeans.labels_[idk]))

# Calcul des distances aux centroïdes des députés : le député est rattaché au␣
 ↪cluster du centroïde dont il est le plus proche.
print(kmeans.transform(df_acp))
```

```
      0
0   383
0    12
0   324
0   396
0    69
..  …
2   230
2   463
2   299
2   440
2   532

[539 rows x 1 columns]
[[ 479.11596367 3648.13177215 5724.52434349]
 [ 100.55790219 3152.26809738 5654.18953033]
 [ 398.81073101 3604.42983382 5700.84624939]
 …
 [ 721.15362157 2506.1698067  5598.77361207]
 [1067.11019678 2191.0571746  5541.45603241]
 [ 286.04607026 3446.22404067 5702.54387738]]
```

On s'intéresse désormais à la répartition des députés au sein des différents clusters.

[39]:
```
nb_clusters = kmeans.labels_.tolist()
print(nb_clusters.count(0)) # Comptage du nombre de députés dans le cluster 1.
print(nb_clusters.count(1)) # Comptage du nombre de députés dans le cluster 2.
print(nb_clusters.count(2)) # Comptage du nombre de députés dans le cluster 3.
```

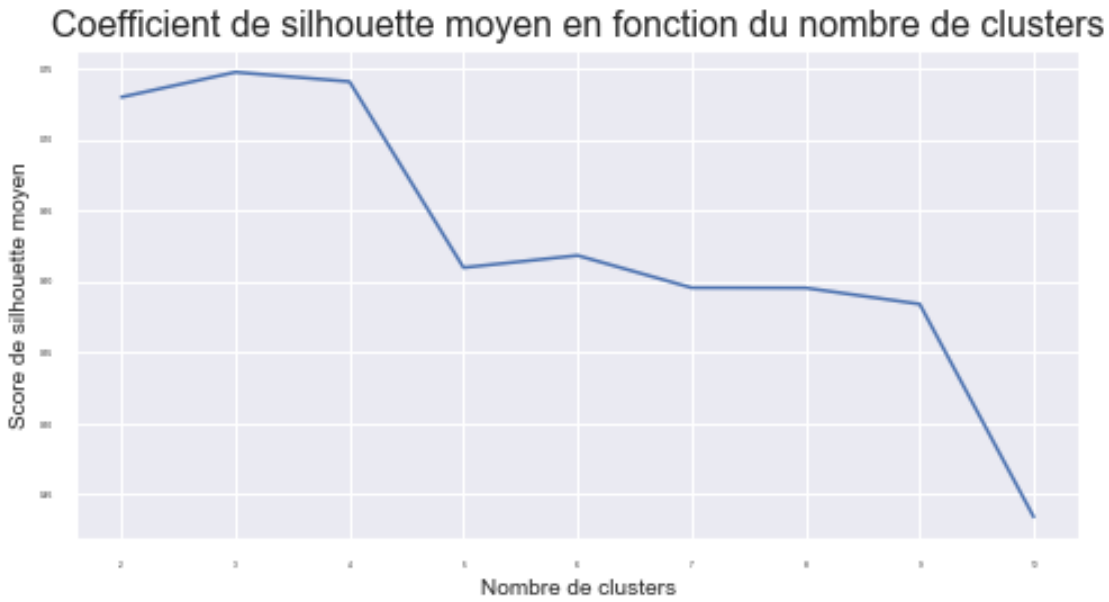
```
428
104
7
```

On observe que l'un des clusters, restreint à 7 observations, semble contenir des députés très marginalisés, dont nous analyserons plus tard les caractéristiques. Vérifions tout d'abord le nombre optimal de clusters en s'appuyant sur la métrique "silhouette" : celle-ci évalue grâce à un coefficient (compris entre -1 et 1) propre à chaque point la différence entre la distance moyenne de ce point avec les points du même cluster et la distance moyenne de ce même point avec les points des autres clusters. Si le point est bien placé, il doit donc être plus proche en moyenne des points de son cluster ; ceci se matérialise par un coefficient de silhouette proche de 1. A l'inverse, un coefficient proche de -1 témoigne d'un mauvais partitionnement. On va donc tracer le score moyen de silhouette en fonction du nombre de clusters, et retenir le nombre de clusters correspondant au score maximal.

[40]:
```
# On va faire varier le nombre de clusters dans un intervalle réaliste, disons␣
↪de 2 à 10.
res = np.arange(9,dtype="double")
for k in np.arange(9):
    km = KMeans(n_clusters=k+2)
    km.fit(df_acp)
    res[k] = silhouette_score(df_acp,km.labels_)
print(res)
```

```
[0.72962241 0.74721136 0.74056135 0.60944656 0.61799867 0.59526497
 0.59513225 0.58376584 0.43371426]
```

[41]:
```
plt.figure(figsize=(8,4))
plt.plot(np.arange(2,11,1),res)
plt.title("Coefficient de silhouette moyen en fonction du nombre de clusters",␣
↪size=16)
plt.xlabel("Nombre de clusters", size=10)
plt.ylabel("Score de silhouette moyen", size=10)
plt.show()
```



A la vue de ce graphique, le choix optimal semble se porter sur 3 clusters.

1.4 Calcul du score d'assiduité

Néanmoins, on peut se demander si la seule présence est suffisante pour établir l'assiduité d'un député. C'est pourquoi on cherche désormais à établir un score pour déterminer le degré d'activité d'un député ; celui s'appuie sur une combinaison linéaire des variables d'activité proposés et pas seulement sur le nombre de semaines de présence. En effet, il semble évident que la rédaction d'un rapport, les interventions dans l'hémicycle ou encore les questions posées sont des éléments manifestant une implication importante du député dans l'exercice de sa tâche. La difficulté réside ici dans le fait qu'il faut habilement pondérer le poids des variables, en majorant le poids des activités individuelles au détriment des actions collectives. On crée des variables intermédiaires portant sur les scores de présence, de participation et de proposition.

[42]:
```
df_work['score_presence'] = 3*df_work['semaines_presence'] +␣
↪2*df_work['commission_presences']
df_work['score_presence'].describe()
```

[42]:
```
count    539.000000
mean     155.300557
std       56.276853
min        0.000000
25%      115.500000
50%      153.000000
75%      190.500000
max      351.000000
Name: score_presence, dtype: float64
```

[43]:
```
df_work['score_participation'] = 2*df_work['commission_interventions'] +␣
↪df_work['hemicycle_interventions'] +␣
↪3*df_work['hemicycle_interventions_courtes'] +␣
↪3*df_work['questions_ecrites'] + 10*df_work['questions_orales']
df_work['score_participation'].describe()
```

[43]:
```
count      539.000000
mean       631.903525
std       2101.486936
min          6.000000
25%         87.000000
50%        183.000000
75%        442.500000
max      22213.000000
Name: score_participation, dtype: float64
```

[44]:
```
df_work['score_proposition'] = 3*(100*df_work['rapports'] +␣
↪100*df_work['propositions_ecrites'] + df_work['amendements_proposes'])
df_work['score_proposition'].describe()
```

[44]:
```
count     539.000000
mean      694.703154
std      1598.578194
min         0.000000
25%        51.000000
50%       162.000000
75%       528.000000
max      9108.000000
Name: score_proposition, dtype: float64
```

[45]:
```
df_work['score_assiduite'] = df_work['score_presence'] +␣
↪df_work['score_participation'] + df_work['score_proposition']
df_work['score_assiduite'].describe()
```

[45]:
```
count      539.000000
mean      1481.907236
std       2847.519860
min         24.000000
25%        311.000000
50%        568.000000
75%       1187.500000
max      25687.000000
Name: score_assiduite, dtype: float64
```

[46]:
```
plt.figure(figsize=(15,10))
sns.set_style("whitegrid")
ax = sns.boxplot(x=df_work["score_assiduite"])
```

```
plt.xlabel("Score d'assiduité du député",size=15)
plt.title("Boxplot représentant le score d'assiduité des␣
 ↪députés",size=20,fontweight='bold')
# Bcp de valeurs extrêmes : quelques députés se démarquent fortement par une␣
 ↪activité très élevée, notamment en matière d'interventions dans l'hémicycle
```

[46]: Text(0.5, 1.0, "Boxplot représentant le score d'assiduité des députés")