Notebook - UE Réduction de dimensionalité et clustering

Quentin Fouché

6 octobre 2022

1 Projet 1 : Réduction de dimensionalité

1.1 Exercice 1

[1]:
```
# (1.1) Import des packages
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
import matplotlib.transforms as transforms
import matplotlib.cm as cm
import seaborn as sns
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.cluster import AgglomerativeClustering, DBSCAN
import scipy.cluster.hierarchy as sch
import sys
!{sys.executable} -m pip install bioinfokit
from bioinfokit.visuz import cluster
!{sys.executable} -m pip install prince
import prince
```

[2]:
```
# (1.2) Import du jeu de données "Iris.csv"
iris_orig = pd.read_csv(r"C:\Users\quent\Documents\DU Data Analyst 2022\UE n°5 - Réduction de dimensionalité et algorithmes de clustering\Jeux de données\Iris.csv", low_memory = False, encoding = "latin-1")
iris = iris_orig.copy(deep = True)
iris.head()
```

[2]:

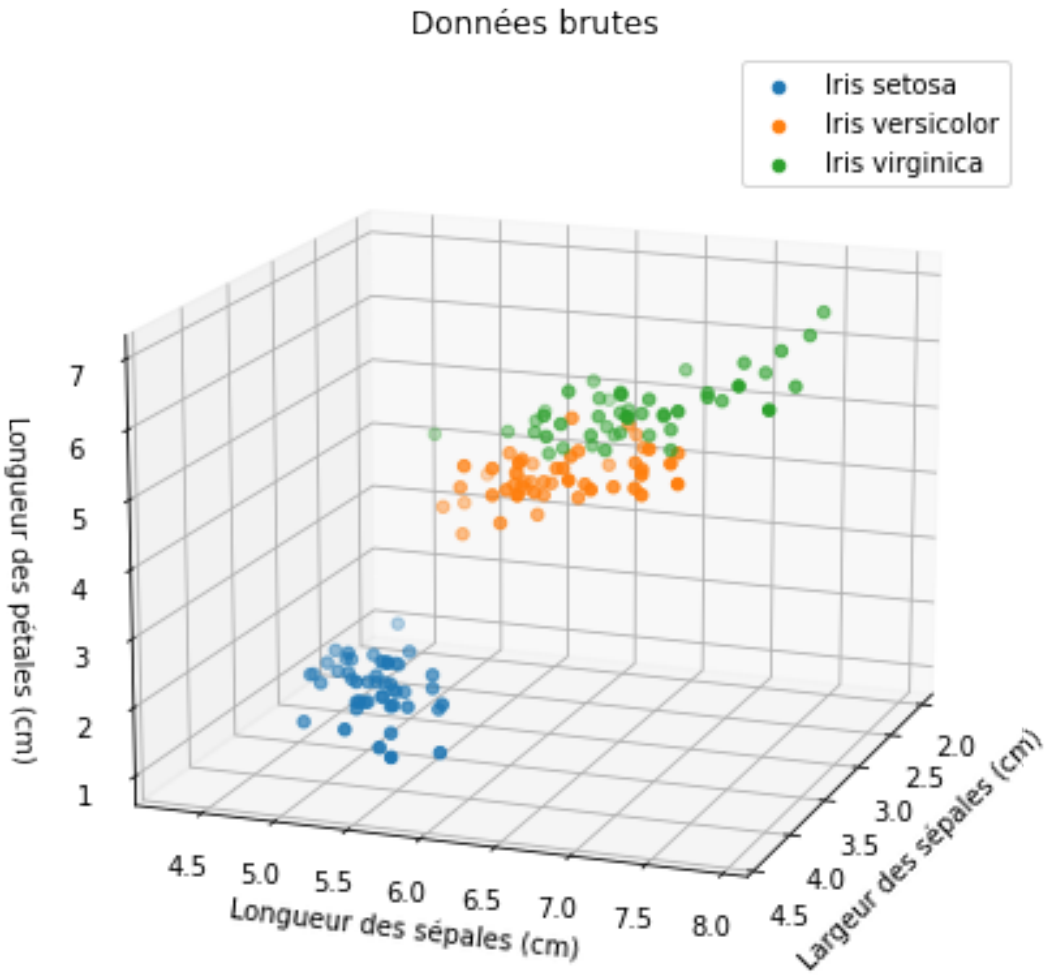
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa

```
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
```

[3]:
```
# (1.3) Inspection des variables
print(iris.info())
print()
print("Nom des espèces :")
print(iris["Species"].unique())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
None

Nom des espèces :
['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

[4]:
```
# (1.4) Représentation des individus en trois dimensions en fonction de la␣
↪longueur des sépales, de la largeur des sépales et de la longueur des␣
↪pétales
iris_setosa = iris[iris["Species"] == "Iris-setosa"]
iris_versicolor = iris[iris["Species"] == "Iris-versicolor"]
iris_virginica = iris[iris["Species"] == "Iris-virginica"]

fig = plt.figure(figsize = (17.5/2.54, 17.5/2.54))
ax = fig.add_subplot(projection = "3d")
ax.scatter(iris_setosa["SepalWidthCm"], iris_setosa["SepalLengthCm"],␣
↪iris_setosa["PetalLengthCm"], marker = "o", label = "Iris setosa")
ax.scatter(iris_versicolor["SepalWidthCm"], iris_versicolor["SepalLengthCm"],␣
↪iris_versicolor["PetalLengthCm"], marker = "o", label = "Iris versicolor")
ax.scatter(iris_virginica["SepalWidthCm"], iris_virginica["SepalLengthCm"],␣
↪iris_virginica["PetalLengthCm"], marker = "o", label = "Iris virginica")
ax.set_title("Données brutes")
ax.set_xlabel("Largeur des sépales (cm)")
ax.set_ylabel("Longueur des sépales (cm)")
ax.set_zlabel("Longueur des pétales (cm)")
ax.legend()
ax.view_init(15,20)
plt.show()
```



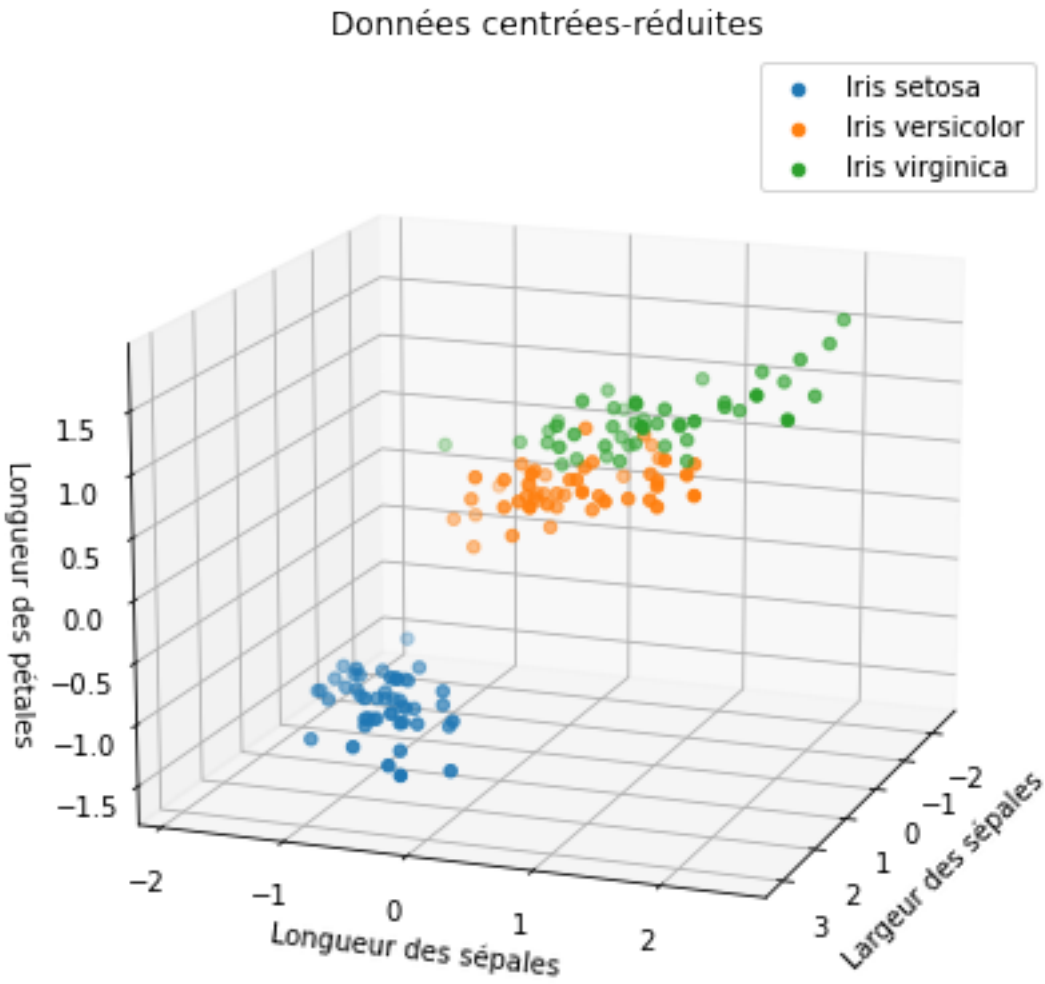
1.2 Exercice 2

[5]:
```
# (2.1) Centrage et réduction des données
iris["SepalLengthCm_stdz"] = (np.array(iris["SepalLengthCm"]) - np.
 ↪array(iris["SepalLengthCm"]).mean()) / np.array(iris["SepalLengthCm"]).std()
iris["SepalWidthCm_stdz"] = (np.array(iris["SepalWidthCm"]) - np.
 ↪array(iris["SepalWidthCm"]).mean()) / np.array(iris["SepalWidthCm"]).std()
iris["PetalLengthCm_stdz"] = (np.array(iris["PetalLengthCm"]) - np.
 ↪array(iris["PetalLengthCm"]).mean()) / np.array(iris["PetalLengthCm"]).std()
iris["PetalWidthCm_stdz"] = (np.array(iris["PetalWidthCm"]) - np.
 ↪array(iris["PetalWidthCm"]).mean()) / np.array(iris["PetalWidthCm"]).std()
iris.head()

# rq : le package StandardScaler permet de réaliser ce calcul automatiquement
#    from sklearn.preprocessing import StandardScaler
#    df_st = StandardScaler().fit_transform(iris[["SepalLengthCm_stdz",␣
 ↪"SepalWidthCm_stdz", "PetalLengthCm_stdz", "PetalWidthCm_stdz"]])
```

```
#    pd.DataFrame(df_st, columns = iris[["SepalLengthCm_stdz",␣
 ↪"SepalWidthCm_stdz", "PetalLengthCm_stdz", "PetalWidthCm_stdz"]].columns)
```

[5]:
```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species ␣
 ↪\
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa

   SepalLengthCm_stdz  SepalWidthCm_stdz  PetalLengthCm_stdz  \
0           -0.900681           1.032057           -1.341272
1           -1.143017          -0.124958           -1.341272
2           -1.385353           0.337848           -1.398138
3           -1.506521           0.106445           -1.284407
4           -1.021849           1.263460           -1.341272

   PetalWidthCm_stdz
0          -1.312977
1          -1.312977
2          -1.312977
3          -1.312977
4          -1.312977
```

[6]:
```
# (2.2) Nouvelle représentation 3D avec les données centrées-réduites
iris_setosa = iris[iris["Species"] == "Iris-setosa"]
iris_versicolor = iris[iris["Species"] == "Iris-versicolor"]
iris_virginica = iris[iris["Species"] == "Iris-virginica"]

fig = plt.figure(figsize = (17.5/2.54, 17.5/2.54))
ax = fig.add_subplot(projection = "3d")
ax.scatter(iris_setosa["SepalWidthCm_stdz"],␣
 ↪iris_setosa["SepalLengthCm_stdz"], iris_setosa["PetalLengthCm_stdz"],␣
 ↪marker = "o", label = "Iris setosa")
ax.scatter(iris_versicolor["SepalWidthCm_stdz"],␣
 ↪iris_versicolor["SepalLengthCm_stdz"],␣
 ↪iris_versicolor["PetalLengthCm_stdz"], marker = "o", label = "Iris␣
 ↪versicolor")
ax.scatter(iris_virginica["SepalWidthCm_stdz"],␣
 ↪iris_virginica["SepalLengthCm_stdz"], iris_virginica["PetalLengthCm_stdz"],␣
 ↪marker = "o", label = "Iris virginica")
ax.set_title("Données centrées-réduites")
ax.set_xlabel("Largeur des sépales")
ax.set_ylabel("Longueur des sépales")
ax.set_zlabel("Longueur des pétales")
ax.legend()
ax.view_init(15,20)
plt.show()
```



1.3 Exercice 3

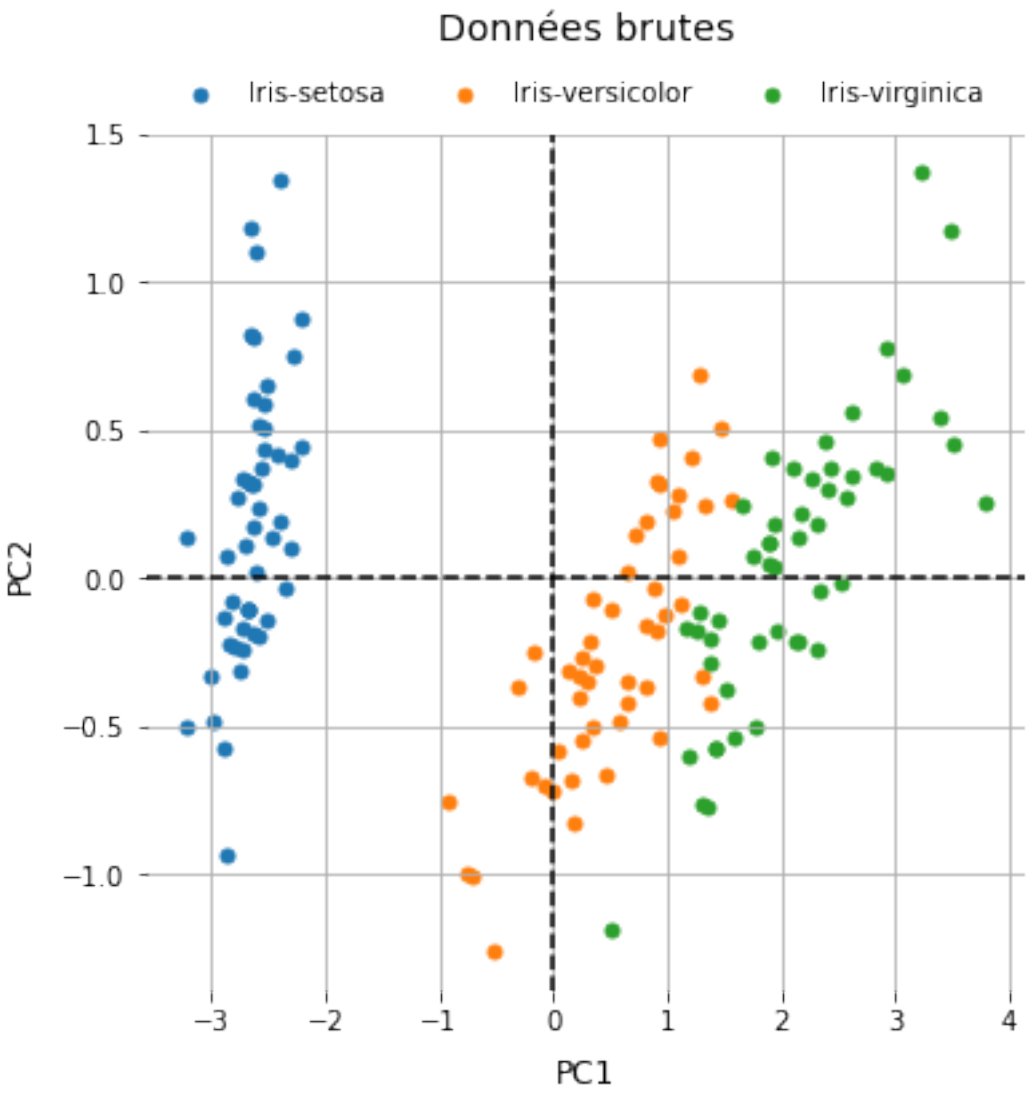
[7]:
```
# (3.1) Réalisation d'une ACP sur les données brutes et affichage des␣
 ↪coordonnées des individus dans les nouvelles composantes
iris_brut = iris[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm",␣
 ↪"PetalWidthCm"]]
pca_brut = PCA()
pca_scores_brut = pca_brut.fit_transform(iris_brut)
pc_list = ["PC" + str(i) for i in list(range(1, len(pca_scores_brut[0])+1))]
df_pca_scores_brut = pd.DataFrame(data = pca_scores_brut, columns = pc_list)
iris_brut_scores_pca = pd.concat([iris[["Species"]], df_pca_scores_brut],␣
 ↪axis = 1)
iris_brut_scores_pca.head()
```

[7]:

	Species	PC1	PC2	PC3	PC4
0	Iris-setosa	-2.684207	0.326607	-0.021512	0.001006
1	Iris-setosa	-2.715391	-0.169557	-0.203521	0.099602

```
2  Iris-setosa -2.889820 -0.137346  0.024709  0.019305
3  Iris-setosa -2.746437 -0.311124  0.037672 -0.075955
4  Iris-setosa -2.728593  0.333925  0.096230 -0.063129
```

[8]:
```
# (3.2) Représentation des individus sur les deux premières composantes␣
↪(données brutes), avec l'espèce en couleur
targets = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]
colors = ["C0", "C1", "C2"]
fig, ax = plt.subplots(figsize = (15/2.54, 15/2.54))
for target, color in zip(targets, colors):
    indicesToKeep = iris_brut_scores_pca["Species"] == target
    ax.scatter(iris_brut_scores_pca.loc[indicesToKeep, "PC1"],␣
↪iris_brut_scores_pca.loc[indicesToKeep, "PC2"], c = color, s = 25)
ax.set_title("Données brutes", fontsize = 14, pad = 35)
ax.set_xlabel("PC1", fontsize = 12, labelpad = 8)
ax.set_ylabel("PC2", fontsize = 12, labelpad = 8)
ax.legend(targets, bbox_to_anchor = (0.98, 1.09), ncol = 3, frameon = False)
ax.spines[:].set_visible(False)
ax.axline(xy1 = (0, 0), xy2 = (0, 1), linewidth = 1.5, c = "black", ls = "--")
ax.axline(xy1 = (0, 0), xy2 = (1, 0), linewidth = 1.5, c = "black", ls = "--")
ax.grid()
```



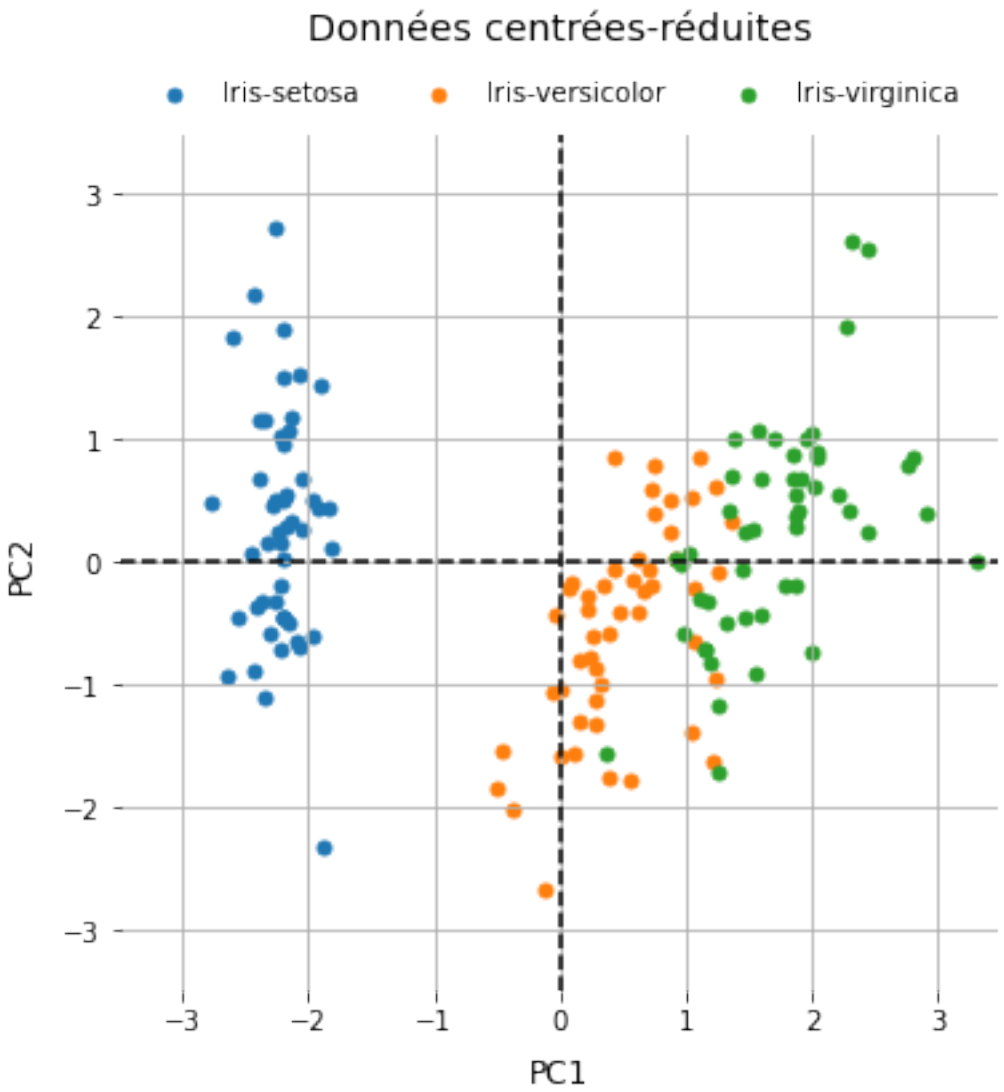
[9]:
```
# (3.3) Réalisation d'une ACP sur les données centrées-réduites et affichage␣
↪des coordonnées des individus dans les nouvelles composantes
iris_stdz = iris[["SepalLengthCm_stdz", "SepalWidthCm_stdz",␣
↪"PetalLengthCm_stdz", "PetalWidthCm_stdz"]]
pca = PCA()
pca_scores = pca.fit_transform(iris_stdz)
pc_list = ["PC" + str(i) for i in list(range(1, len(pca_scores[0])+1))]
df_pca_scores = pd.DataFrame(data = pca_scores, columns = pc_list)
iris_pca_scores = pd.concat([iris[["Species"]], df_pca_scores], axis = 1)
iris_pca_scores.head()
```

[9]:

	Species	PC1	PC2	PC3	PC4
0	Iris-setosa	-2.264542	0.505704	-0.121943	-0.023073
1	Iris-setosa	-2.086426	-0.655405	-0.227251	-0.103208
2	Iris-setosa	-2.367950	-0.318477	0.051480	-0.027825
3	Iris-setosa	-2.304197	-0.575368	0.098860	0.066311

```
4  Iris-setosa -2.388777  0.674767  0.021428  0.037397
```

[10]:
```
# (3.4) Représentation des individus sur les deux premières composantes␣
↪(données centrées-réduites), avec l'espèce en couleur
targets = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]
colors = ["C0", "C1", "C2"]
fig, ax = plt.subplots(figsize = (15/2.54, 15/2.54))
for target, color in zip(targets, colors):
    indicesToKeep = iris_pca_scores["Species"] == target
    ax.scatter(iris_pca_scores.loc[indicesToKeep, "PC1"], iris_pca_scores.
↪loc[indicesToKeep, "PC2"], c = color, s = 25)
ax.set_title("Données centrées-réduites", fontsize = 14, pad = 35)
ax.set_xlabel("PC1", fontsize = 12, labelpad = 8)
ax.set_ylabel("PC2", fontsize = 12, labelpad = 8)
ax.set_xlim(-3.49,3.49)
ax.set_ylim(-3.49,3.49)
ax.legend(targets, bbox_to_anchor = (0.98, 1.09), ncol = 3, frameon = False)
ax.spines[:].set_visible(False)
ax.axline(xy1 = (0, 0), xy2 = (0, 1), linewidth = 1.5, c = "black", ls = "--")
ax.axline(xy1 = (0, 0), xy2 = (1, 0), linewidth = 1.5, c = "black", ls = "--")
ax.grid()
```



Dans les exercices suivants, seules les données standardisées sont utilisées.

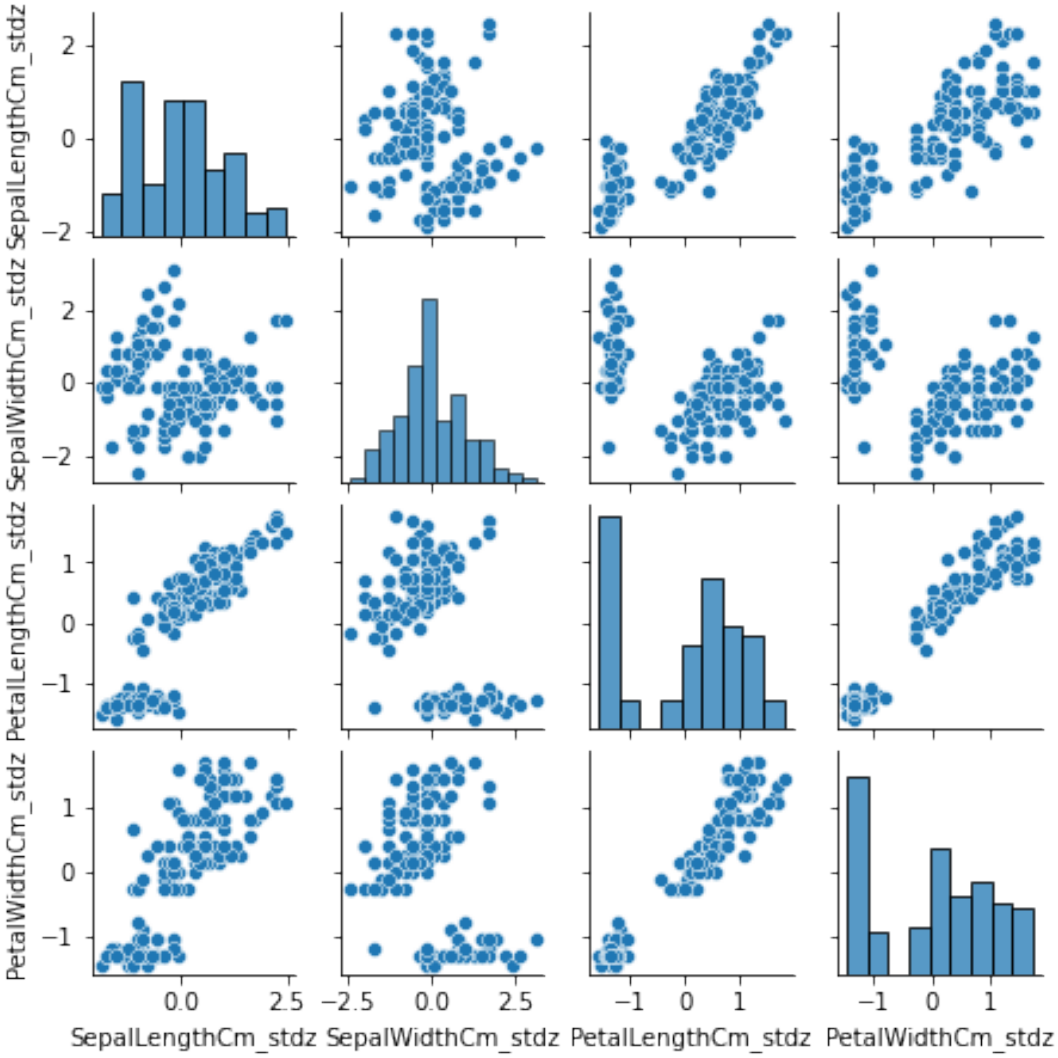
1.4 Exercice 4

[11]:
```
# (4.1) Calcul des coefficients de corrélations entre chaque paire de␣
↪variables
iris_stdz.corr("pearson")
```

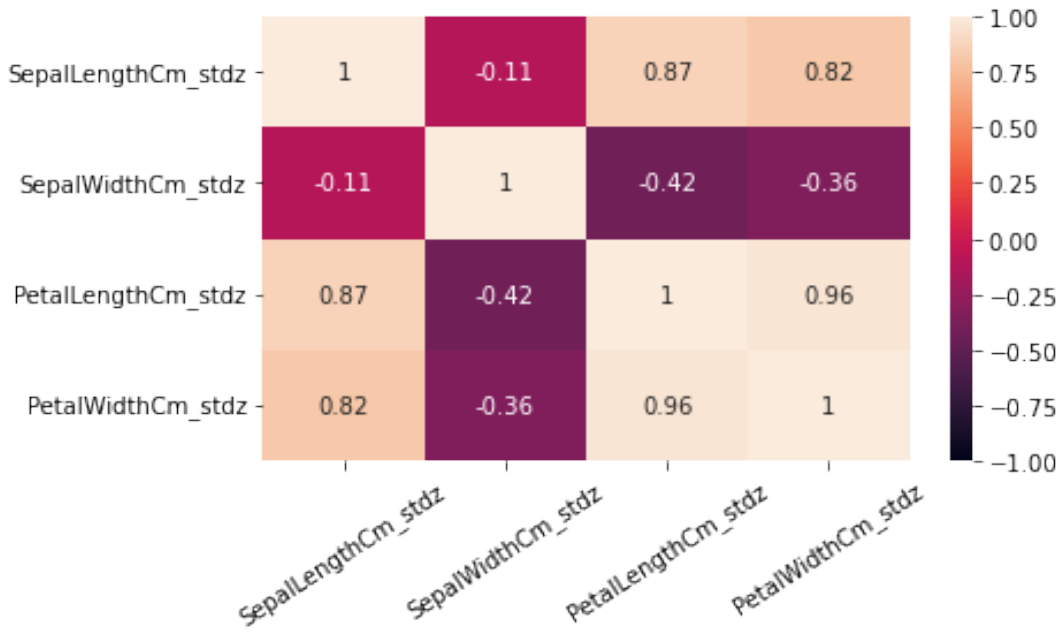
[11]:
```
                    SepalLengthCm_stdz  SepalWidthCm_stdz  PetalLengthCm_stdz␣
↪ \
SepalLengthCm_stdz            1.000000          -0.109369            0.871754
SepalWidthCm_stdz            -0.109369           1.000000           -0.420516
PetalLengthCm_stdz            0.871754          -0.420516            1.000000
PetalWidthCm_stdz             0.817954          -0.356544            0.962757

                    PetalWidthCm_stdz
SepalLengthCm_stdz           0.817954
```

```
SepalWidthCm_stdz            -0.356544
PetalLengthCm_stdz            0.962757
PetalWidthCm_stdz             1.000000
```

[12]:
```
# (4.2) Représentation des corrélations par des nuages de points (et␣
↪histogrammes de distribution)
sns.pairplot(iris_stdz, height = 4/2.54)
plt.show()
```



[13]:
```
# (4.3) Représentation des corrélations par un corrélogramme
plt.figure(figsize = (16/2.54, 9/2.54))
sns.heatmap(iris_stdz.corr(), annot = True, vmin = -1, vmax = 1)
plt.xticks(rotation = 33)
plt.show()
```



1.5 Exercice 5

[14]:
```
# (5.1) Affichage des pourcentages de variance expliquée par composante␣
↪principale
for i in range(len(pc_list)):
    if len(str(round(pca.explained_variance_ratio_[i], 3)*100)) <= 4:
        print(str(pc_list[i]) + " : " + str(round(pca.
↪explained_variance_ratio_[i], 3)*100) + "%")
    else:
        print(str(pc_list[i]) + " : " + str(round(pca.
↪explained_variance_ratio_[i], 3)*100)[0:3] + "%")
```

```
PC1 : 72.8%
PC2 : 23.0%
PC3 : 3.6%
PC4 : 0.5%
```

[15]:
```
# (5.2) Import des coordonnées des variables à partir de R (avec le code␣
↪indiqué ci-dessous)
iris_pca_var = pd.read_csv(r"C:\Users\quent\Documents\DU Data Analyst 2022\UE␣
↪n°5 - Réduction de dimensionalité et algorithmes de clustering\Jeux de␣
↪données\ACP_var.csv", low_memory = False, encoding = "latin-1")
iris_pca_var.head()
```

[15]:

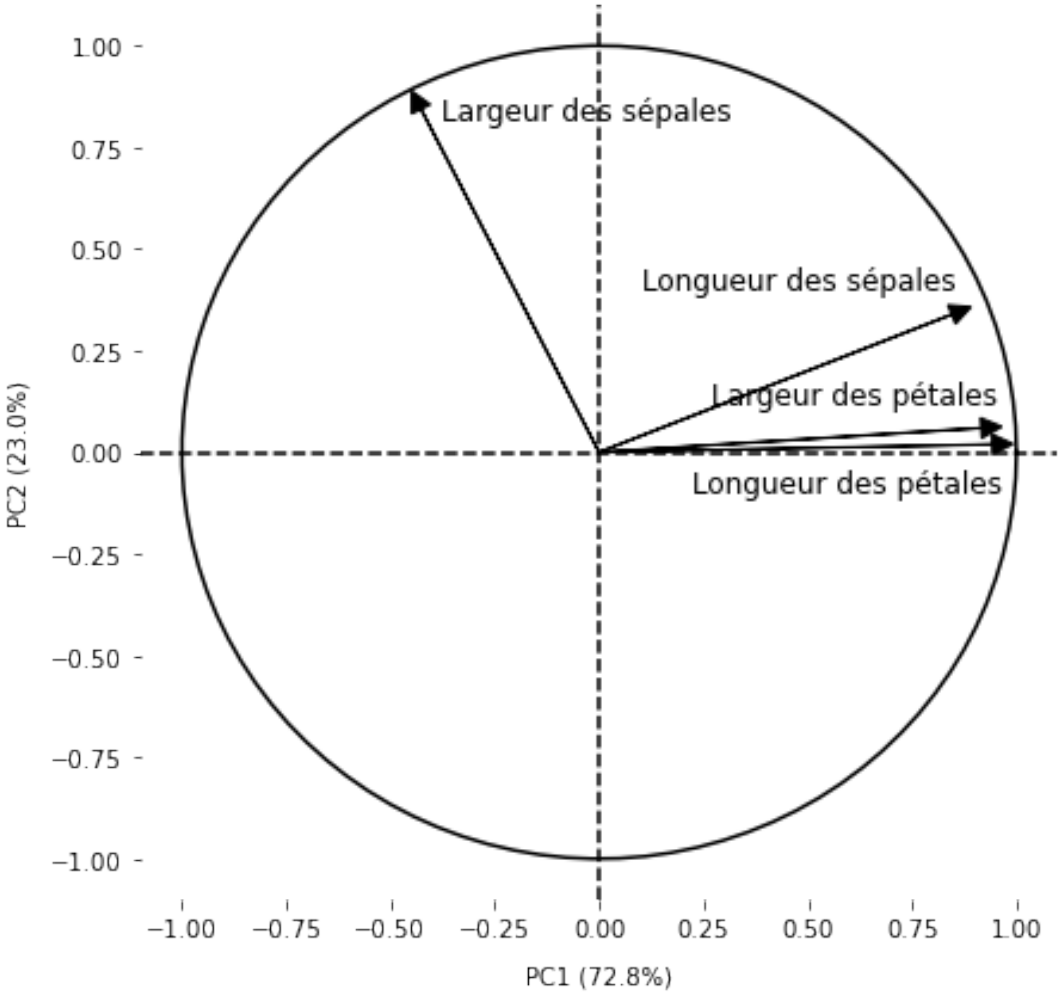
	name	x	y	coord	cos2	contrib
0	SepalLengthCm_sdtz	0.891224	0.357352	0.921982	0.921982	24.05982
1	SepalWidthCm_sdtz	-0.449313	0.888351	0.991050	0.991050	25.86222
2	PetalLengthCm_sdtz	0.991684	0.020247	0.983848	0.983848	25.67427
3	PetalWidthCm_sdtz	0.964996	0.062786	0.935159	0.935159	24.40369

Code utilisé sous R :

```
library("factoextra")
library("FactoMineR")
res.pca <- PCA(iris, graph = FALSE)
fviz_pca_var(res.pca)$data
```

```
[output]
              name          x          y     coord      cos2  contrib
SepalLengthCm_sdtz  0.8912245 0.35735211 0.9219816 0.9219816 24.05982
 SepalWidthCm_sdtz -0.4493130 0.88835148 0.9910505 0.9910505 25.86222
PetalLengthCm_sdtz  0.9916844 0.02024682 0.9838479 0.9838479 25.67427
 PetalWidthCm_sdtz  0.9649958 0.06278622 0.9351590 0.9351590 24.40369
```

[16]:
```
# (5.3) Représentation du cercle des corrélations
fig, ax = plt.subplots(figsize = (17.5/2.54, 17.5/2.54))
for i in range(0, iris_pca_var.shape[0]):
    ax.arrow(0, 0, iris_pca_var["x"][i], iris_pca_var["y"][i], head_width = 0.
 ↪05, head_length = 0.05, length_includes_head = True, color = "black")
circle = np.linspace(0, 2 * np.pi, 100)
ax.plot(np.cos(circle), np.sin(circle), color = "black")
ax.set_xlabel("PC1 (72.8%)", labelpad = 10)
ax.set_ylabel("PC2 (23.0%)", labelpad = 8)
ax.text(0.1, 0.4, "Longueur des sépales", fontsize = 12)
ax.text(-0.38, 0.82, "Largeur des sépales", fontsize = 12)
ax.text(0.22, -0.1, "Longueur des pétales", fontsize = 12)
ax.text(0.27, 0.12, "Largeur des pétales", fontsize = 12)
ax.spines[:].set_visible(False)
ax.axline(xy1 = (0, 0), xy2 = (0, 1), linewidth = 1.5, c = "black", ls = "--")
ax.axline(xy1 = (0, 0), xy2 = (1, 0), linewidth = 1.5, c = "black", ls = "--")
plt.show()
```



D'après le corrélogramme affiché dans l'exercice 4, les variables les plus corrélées sont la longueur des pétales, la largeur des pétales et la longueur des sépales (coefficients de corrélation allant de 0.82 et 0.96). La largeur des sépales est faiblement et négativement corrélée à ces trois variables (coefficients de corrélation allant de -0.11 et -0.42).

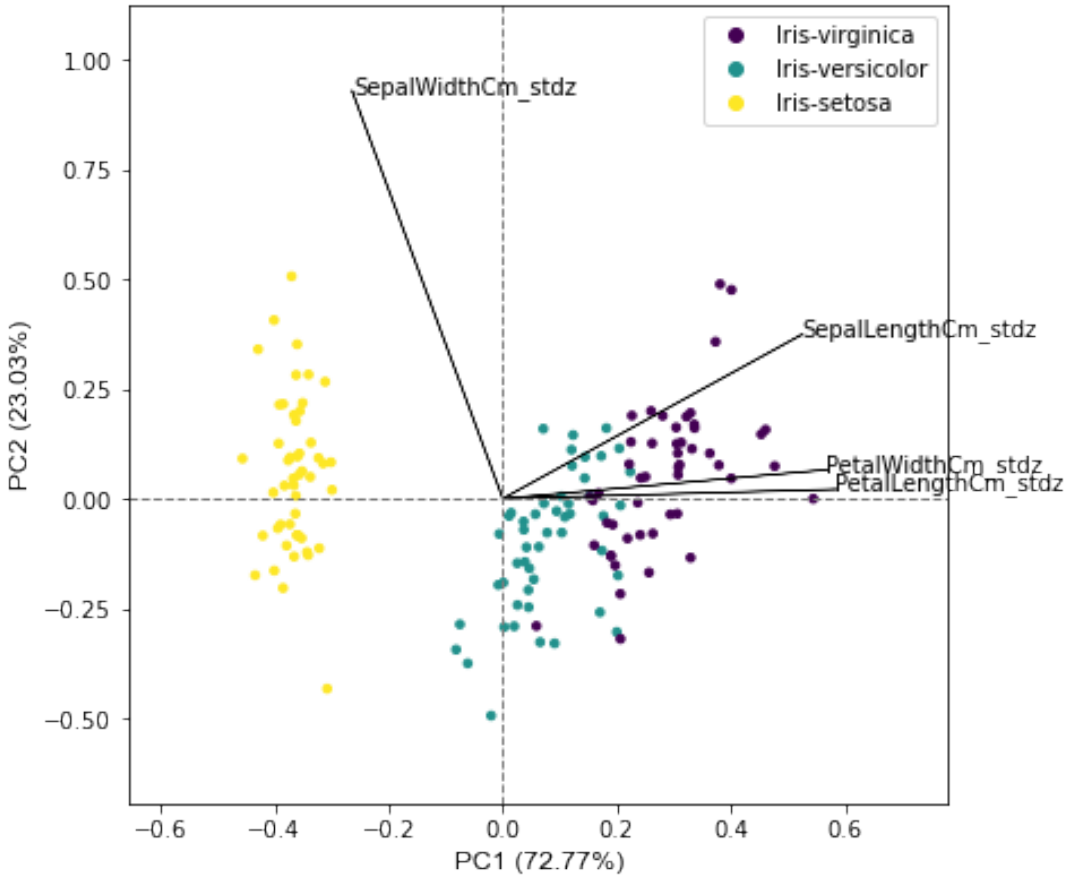
Ces corrélations se matérialisent dans le cercle des corrélations par une proximité forte entre les flèches associées à la longueur des pétales, la largeur des pétales et la longueur des sépales, les trois flèches pointant dans la même direction. La faible corrélation de la largeur des sépales avec ces trois variables est matérialisée par un angle proche de 90° entre la flèche associée à cette variable et les trois autres.

1.6 Exercice 6

[17]:
```
# (6.1) Calcul de la saturation des variables par composante (voir aussi␣
↪l'exercice 10)
pca_out = pca.fit(iris_stdz)
loadings = pca_out.components_
num_pc = pca_out.n_features_
loadings_df = pd.DataFrame.from_dict(dict(zip(pc_list, loadings)))
loadings_df["variable"] = iris_stdz.columns.values
```

```
loadings_df = loadings_df.set_index("variable")
```

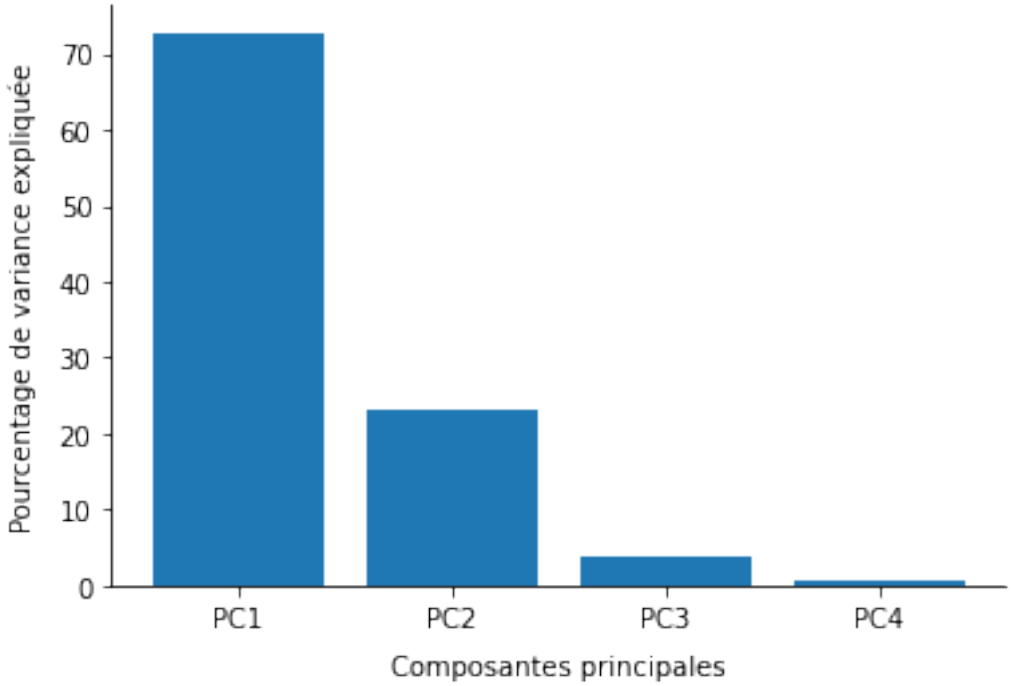
[18]:
```
# (6.2) Représentation des individus et des variables dans un biplot␣
↪(documentation : https://reneshbedre.github.io/blog/howtoinstall.html)
cluster.biplot(cscore = pca_scores, loadings = loadings, labels = iris_stdz.
↪columns.values, var1 = round(pca_out.explained_variance_ratio_[0]*100, 2),␣
↪var2 = round(pca_out.explained_variance_ratio_[1]*100, 2), colorlist = np.
↪array(iris["Species"]), show = True, axlabelfontsize = 12, dim = (16.5/2.
↪54, 16.5/2.54), arrowcolor = "black", dotsize = 12)
```



La part de la variance représentée sur ce biplot est égale à la somme de la variance expliquée par les deux composantes principales : $72.77 + 23.03 = 95.8\%$.

1.7 Exercice 7

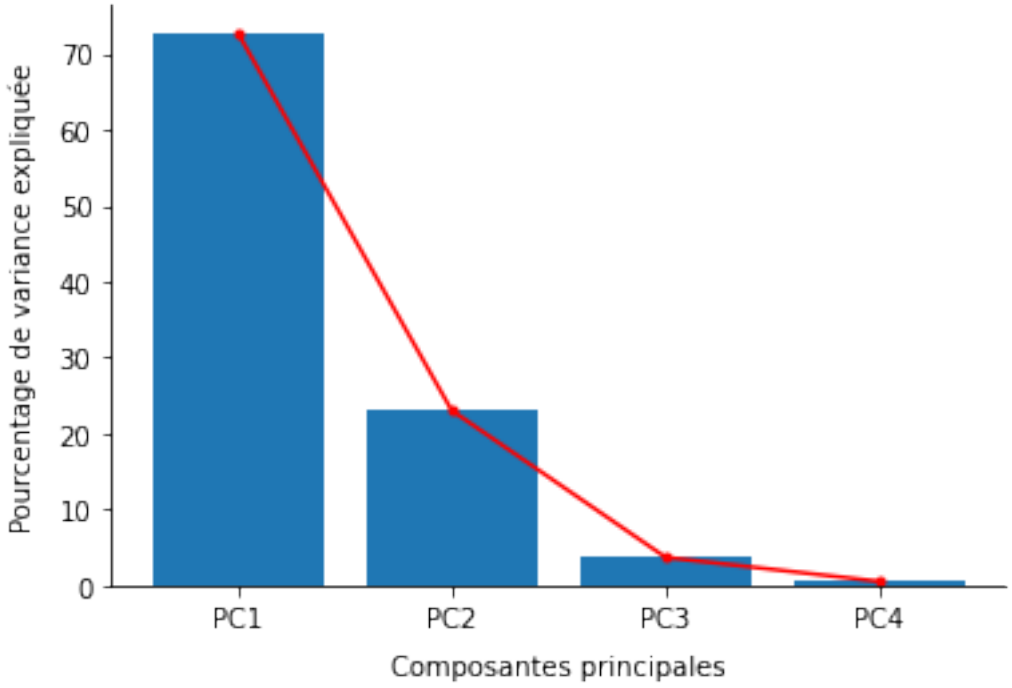
[19]:
```
# (7.1) Représentation du scree plot (pourcentage de variance expliquée en␣
↪fonction des composantes principales)
plt.bar(x = ["PC1", "PC2", "PC3", "PC4"], height = pca.
↪explained_variance_ratio_*100)
plt.xlabel("Composantes principales", labelpad = 8)
plt.ylabel("Pourcentage de variance expliquée", labelpad = 8)
plt.gca().spines[["right", "top"]].set_visible(False)
plt.show()
```



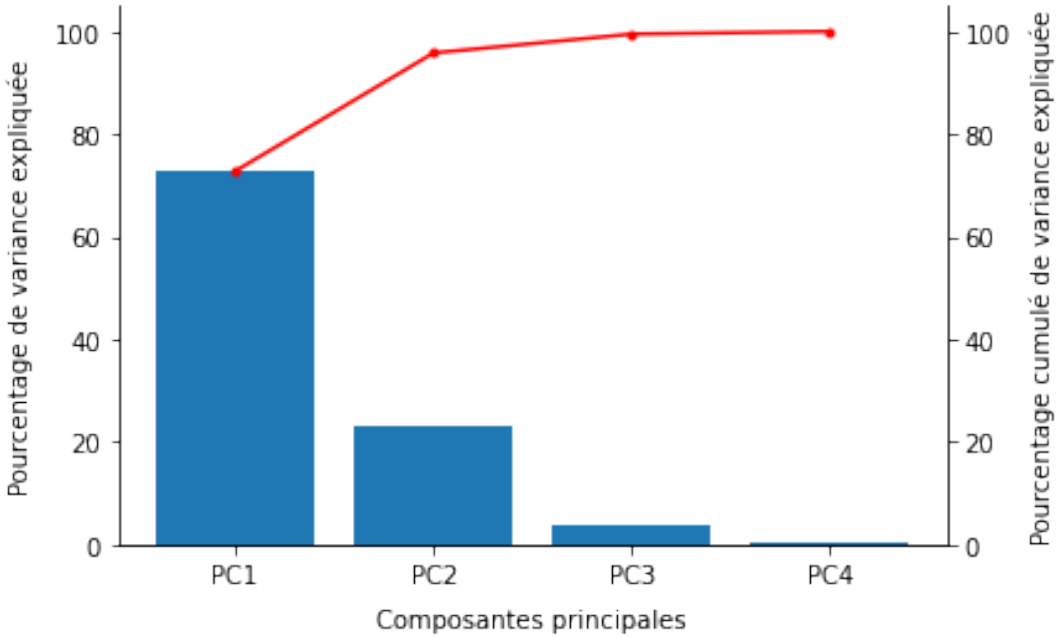
D'après ce scree plot, les deux premières composantes expliquent respectivement 73% et 23% de la variance initiale, soit 95% à elles deux. Il est donc pertinent de retenir uniquement ces deux composantes pour représenter les données.

1.8 Exercice 8

[20]:
```
# (8.1) Représentation du scree plot en affichant une courbe permettant de␣
↪visualiser plus nettement le "coude"
plt.bar(x = ["PC1", "PC2", "PC3", "PC4"], height = pca.
↪explained_variance_ratio_*100)
plt.plot(np.arange(4), pca.explained_variance_ratio_*100, color = "red",␣
↪marker = ".")
plt.xlabel("Composantes principales", labelpad = 8)
plt.ylabel("Pourcentage de variance expliquée", labelpad = 8)
plt.gca().spines[["right", "top"]].set_visible(False)
plt.show()
```



[21]:
```
# (8.2) Représentation du scree plot ; la courbe représente cette fois le␣
 ↪pourcentage cumulé de variance expliquée au fil des composantes
pca_explained_variance_ratio_cum = np.cumsum(pca.
 ↪explained_variance_ratio_*100)
fig, ax = plt.subplots()
ax.bar(x = ["PC1", "PC2", "PC3", "PC4"], height = pca.
 ↪explained_variance_ratio_*100)
ax2 = ax.twinx()
ax2.plot(np.arange(4), pca_explained_variance_ratio_cum, color = "red",␣
 ↪marker = ".")
ax.set_xlabel("Composantes principales", labelpad = 8)
ax.set_ylabel("Pourcentage de variance expliquée", labelpad = 8)
ax2.set_ylabel("Pourcentage cumulé de variance expliquée", labelpad = 8)
ax.set_ylim(0, 105)
ax2.set_ylim(0, 105)
ax.spines[["right", "top"]].set_visible(False)
ax2.spines[["left", "top"]].set_visible(False)
plt.show()
```



1.9 Exercice 9

[22]:
```
# (9.1) Création d'un tableau indiquant pour chaque composante les valeurs␣
↪propres, le pourcentage de variance expliquée et le pourcentage cumulé de␣
↪variance expliquée
df_eigenvalues = pd.DataFrame({"Valeurs propres" : pca.explained_variance_,␣
↪"Variance expliquée (%)" : pca.explained_variance_ratio_*100, "Variance␣
↪expliquée cumulée (%)" : pca_explained_variance_ratio_cum})
df_eigenvalues.index = ["PC1", "PC2", "PC3", "PC4"]
df_eigenvalues
```

[22]:

	Valeurs propres	Variance expliquée (%)	Variance expliquée cumulée (%)
PC1	2.930354	72.7705	72.770452
PC2	0.927404	23.0305	95.800975
PC3	0.148342	3.6838	99.484807
PC4	0.020746	0.5152	100.000000

La part de variance expliquée par une composante est proportionnelle à sa valeur propre (“eigen value”).

1.10 Exercice 10

[23]:
```
# (10.1) Affichage de la saturation des variables par composante (ou␣
↪"vecteurs propres" ; reprise du tableau créé dans l'exercice 6)
loadings_df.index = ["Longueur des sépales", "Largeur des sépales", "Longueur␣
↪des pétales", "Largeur des pétales"]
loadings_df
```

[23]:

	PC1	PC2	PC3	PC4
Longueur des sépales	0.522372	0.372318	-0.721017	-0.261996

```
Largeur des sépales  -0.263355  0.925556  0.242033  0.124135
Longueur des pétales  0.581254  0.021095  0.140892  0.801154
Largeur des pétales   0.565611  0.065416  0.633801 -0.523546
```

Pour la première composante, la variable dont la saturation est la plus forte est la longueur des pétales (0.58). Dans le cercle des corrélations, le vecteur représentant cette variable est le plus proche de l'axe de la première composante.

1.11 Exercice 11

[25]:
```
# (11.1) Affichage de la saturation des variables par composante en triant␣
↪les lignes par ordre décroissant des valeurs sur la PC1
loadings_df.sort_values("PC1", ascending = False)
```

[25]:
```
                            PC1       PC2       PC3       PC4
Longueur des pétales   0.581254  0.021095  0.140892  0.801154
Largeur des pétales    0.565611  0.065416  0.633801 -0.523546
Longueur des sépales   0.522372  0.372318 -0.721017 -0.261996
Largeur des sépales   -0.263355  0.925556  0.242033  0.124135
```

1.12 Exercice 12

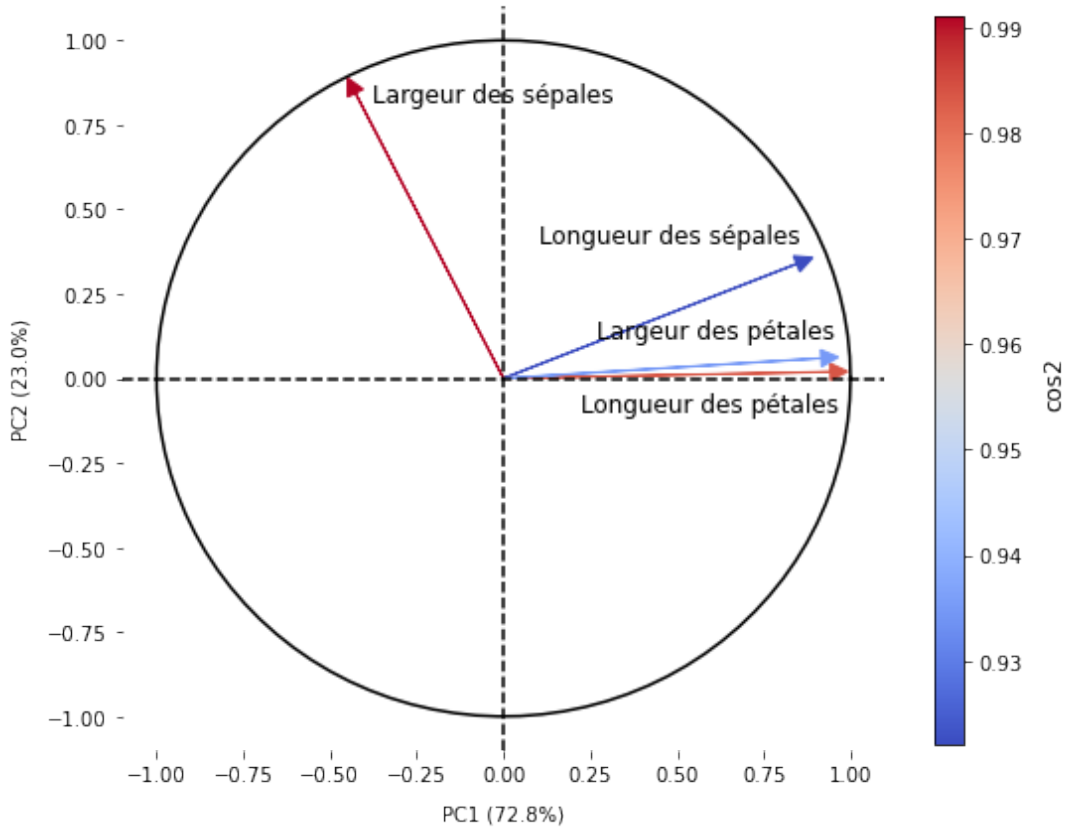
La saturation d'une variable sur une composante représente le coefficient de corrélation entre cette variable et la composante. Dans le cercle des corrélations, les coordonnées de l'extrémité des flèches correspondent à la saturation des variables sur chaque composante.

1.13 Exercice 13

La qualité de la représentation d'une variable par une ACP correspond au pourcentage de variance de cette variable expliquée par les premières composantes de l'ACP. Ainsi, plus la part de variance expliquée par les premières composantes est élevée, mieux la variable est représentée.

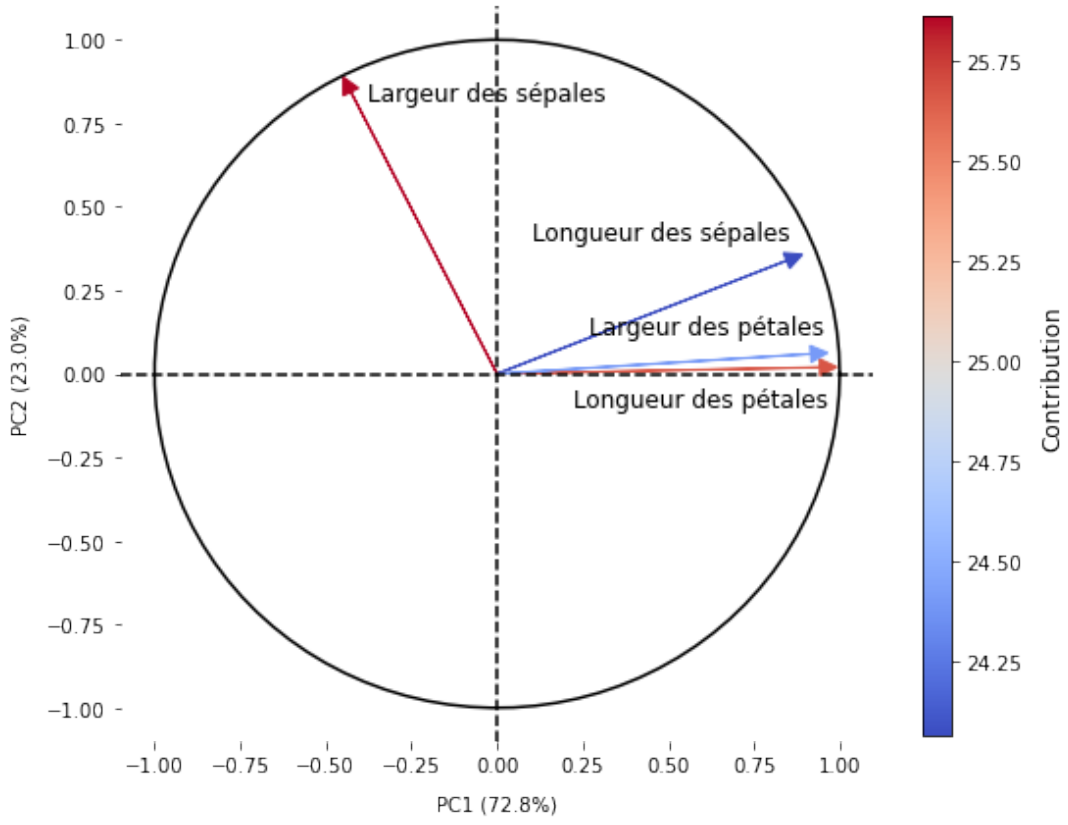
[25]:
```
# (13.1) Représentation du cercle des corrélations en indiquant par un␣
↪gradient de couleur la qualité de la représentation de chaque variable (le␣
↪gradient est basé sur le cosinus carré des variables ; bleu : cosinus␣
↪faible ; rouge : cosinus élevé))
my_color_gradient = mpl.colormaps['coolwarm'](range(256))
cos2 = list(iris_pca_var["cos2"])
cos2_pour_gradient = list(np.array(cos2) / (np.array(cos2).max() - np.
↪array(cos2).min()))
cos2_pour_gradient = list((np.array(cos2_pour_gradient) - np.
↪array(cos2_pour_gradient).min())*255)
fig, ax = plt.subplots(figsize = (17.5/2.54, 17.5/2.54))
for i in range(0, iris_pca_var.shape[0]):
    ax.arrow(0, 0, iris_pca_var["x"][i], iris_pca_var["y"][i], head_width = 0.
↪05, head_length = 0.05, length_includes_head = True, color =␣
↪my_color_gradient[int(cos2_pour_gradient[i])])
circle = np.linspace(0, 2 * np.pi, 100)
ax.plot(np.cos(circle), np.sin(circle), color = "black")
ax.set_xlabel("PC1 (72.8%)", labelpad = 10)
ax.set_ylabel("PC2 (23.0%)", labelpad = 8)
ax.text(0.1, 0.4, "Longueur des sépales", fontsize = 12)
```

```
ax.text(-0.38, 0.82, "Largeur des sépales", fontsize = 12)
ax.text(0.22, -0.1, "Longueur des pétales", fontsize = 12)
ax.text(0.27, 0.12, "Largeur des pétales", fontsize = 12)
ax.spines[:].set_visible(False)
ax.axline(xy1 = (0, 0), xy2 = (0, 1), linewidth = 1.5, c = "black", ls = "--")
ax.axline(xy1 = (0, 0), xy2 = (1, 0), linewidth = 1.5, c = "black", ls = "--")
cb = fig.colorbar(mpl.cm.ScalarMappable(norm = mpl.colors.Normalize(vmin = np.
 ↪array(cos2).min(), vmax = np.array(cos2).max()), cmap = mpl.cm.coolwarm),␣
 ↪cax = plt.axes([0.95, 0.13, 0.03, 0.74]), orientation = "vertical")
cb.set_label("cos2", fontsize = 12, labelpad = 10)
plt.show()
```



[26]:
```
# (13.2) Même représentation mais en basant le gradient de couleur sur la␣
 ↪contribution de la variable aux deux premières composantes (bleu : faible␣
 ↪contribution ; rouge : forte contribution)
my_color_gradient = mpl.colormaps['coolwarm'](range(256))
contrib = list(iris_pca_var["contrib"])
contrib_pour_gradient = list(np.array(contrib) / (np.array(contrib).max() -␣
 ↪np.array(contrib).min()))
contrib_pour_gradient = list((np.array(contrib_pour_gradient) - np.
 ↪array(contrib_pour_gradient).min())*255)
fig, ax = plt.subplots(figsize = (17.5/2.54, 17.5/2.54))
for i in range(0, iris_pca_var.shape[0]):
```

```
    ax.arrow(0, 0, iris_pca_var["x"][i], iris_pca_var["y"][i], head_width = 0.
 ↪05, head_length = 0.05, length_includes_head = True, color =␣
 ↪my_color_gradient[int(contrib_pour_gradient[i])])
circle = np.linspace(0, 2 * np.pi, 100)
ax.plot(np.cos(circle), np.sin(circle), color = "black")
ax.set_xlabel("PC1 (72.8%)", labelpad = 10)
ax.set_ylabel("PC2 (23.0%)", labelpad = 8)
ax.text(0.1, 0.4, "Longueur des sépales", fontsize = 12)
ax.text(-0.38, 0.82, "Largeur des sépales", fontsize = 12)
ax.text(0.22, -0.1, "Longueur des pétales", fontsize = 12)
ax.text(0.27, 0.12, "Largeur des pétales", fontsize = 12)
ax.spines[:].set_visible(False)
ax.axline(xy1 = (0, 0), xy2 = (0, 1), linewidth = 1.5, c = "black", ls = "--")
ax.axline(xy1 = (0, 0), xy2 = (1, 0), linewidth = 1.5, c = "black", ls = "--")
cb = fig.colorbar(mpl.cm.ScalarMappable(norm = mpl.colors.Normalize(vmin = np.
 ↪array(contrib).min(), vmax = np.array(contrib).max()), cmap = mpl.cm.
 ↪coolwarm), cax = plt.axes([0.95, 0.13, 0.03, 0.74]), orientation =␣
 ↪"vertical")
cb.set_label("Contribution", fontsize = 12, labelpad = 10)
plt.show()
```



Remarque : les données de $\cos^2$ et de contribution ont été calculées sur R, selon le code indiqué dans l'exercice 5.

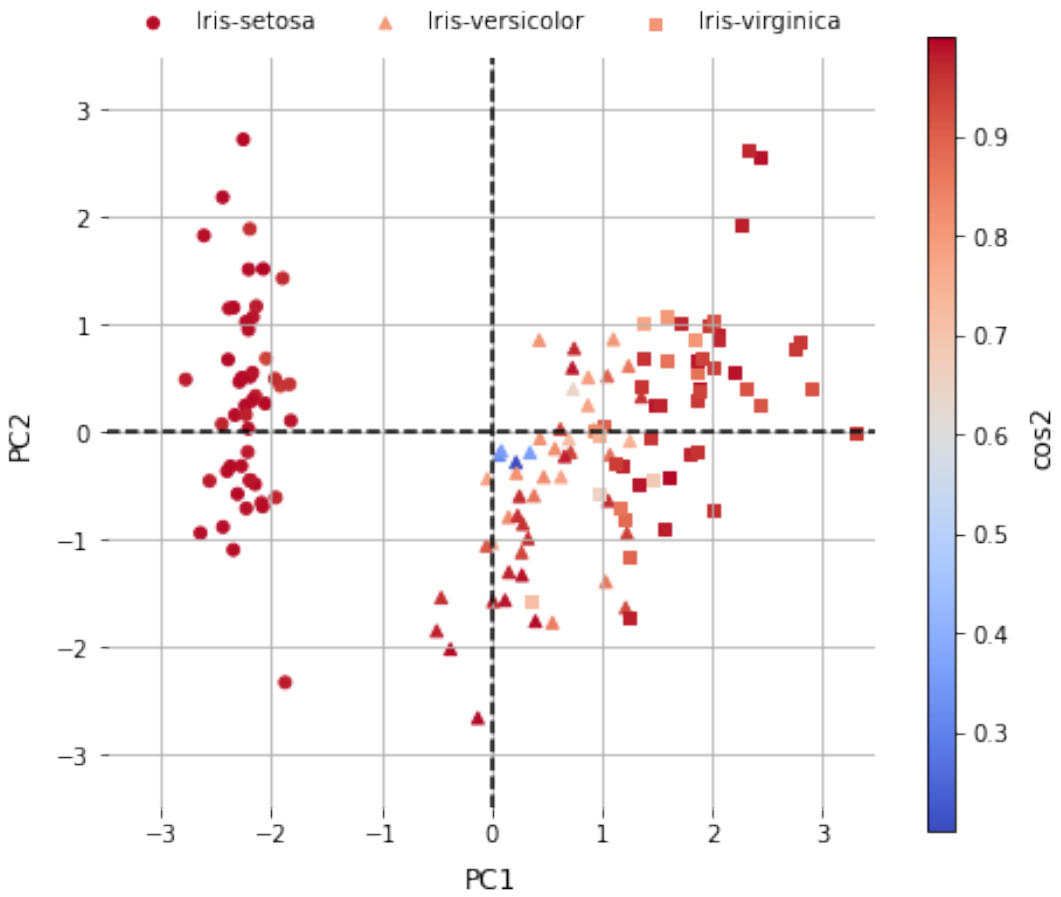
Le cosinus carré représente la qualité de représentation de la variable par l'ACP, tandis que la

contribution représente le “poids” apporté par la variable à la composante. Ces deux métriques donnent des résultats similaires car elles ne diffèrent que d’une constante.

1.14 Exercice 14

[27]:
```
# (14.1) Représentation des individus sur les deux premières composantes, en␣
 ↪indiquant par un gradient de couleur la qualité de la représentation de␣
 ↪chaque individu (i.e., le cos²)
# ___(14.1.1) Création du gradient de couleur
iris_pca_contrib_ind = pd.read_csv(r"C:\Users\quent\Documents\DU Data Analyst␣
 ↪2022\UE n°5 - Réduction de dimensionalité et algorithmes de clustering\Jeux␣
 ↪de données\ACP_contrib_ind.csv", low_memory = False, encoding = "latin-1")
iris_pca_scores_cos2 = iris_pca_scores.copy(deep = True)
iris_pca_scores_cos2["cos2"] = iris_pca_contrib_ind["cos2"]
my_color_gradient = mpl.colormaps['coolwarm'](range(256))
cos2_pour_gradient = list(np.array(list(iris_pca_scores_cos2["cos2"])) / (np.
 ↪array(list(iris_pca_scores_cos2["cos2"])).max() - np.
 ↪array(list(iris_pca_scores_cos2["cos2"])).min()))
cos2_pour_gradient = list((np.array(cos2_pour_gradient) - np.
 ↪array(cos2_pour_gradient).min())*255)
cos2_pour_gradient = [int(cos2_pour_gradient[i]) for i in␣
 ↪range(len(cos2_pour_gradient))]
iris_pca_scores_cos2["color_gradient"] =␣
 ↪[my_color_gradient[cos2_pour_gradient[i]] for i in␣
 ↪range(len(cos2_pour_gradient))]

# ___(14.1.2) Création du graphe (les espèces sont représentées par des␣
 ↪symboles différents)
targets = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]
markers = ["o", "^", "s"]
fig, ax = plt.subplots(figsize = (15/2.54, 15/2.54))
for target, marker in zip(targets, markers):
    indicesToKeep = iris_pca_scores_cos2["Species"] == target
    ax.scatter(iris_pca_scores_cos2.loc[indicesToKeep, "PC1"],␣
 ↪iris_pca_scores_cos2.loc[indicesToKeep, "PC2"], c = iris_pca_scores_cos2.
 ↪loc[indicesToKeep, "color_gradient"], s = 25, marker = marker)
ax.set_xlabel("PC1", fontsize = 12, labelpad = 8)
ax.set_ylabel("PC2", fontsize = 12, labelpad = 8)
ax.set_xlim(-3.49,3.49)
ax.set_ylim(-3.49,3.49)
ax.legend(targets, bbox_to_anchor = (0.98, 1.09), ncol = 3, frameon = False)
ax.spines[:].set_visible(False)
ax.axline(xy1 = (0, 0), xy2 = (0, 1), linewidth = 1.5, c = "black", ls = "--")
ax.axline(xy1 = (0, 0), xy2 = (1, 0), linewidth = 1.5, c = "black", ls = "--")
ax.grid()
cb = fig.colorbar(mpl.cm.ScalarMappable(norm = mpl.colors.Normalize(vmin = np.
 ↪array(iris_pca_scores_cos2["cos2"]).min(), vmax = np.
 ↪array(iris_pca_scores_cos2["cos2"]).max()), cmap = mpl.cm.coolwarm), cax =␣
 ↪plt.axes([0.95, 0.1, 0.03, 0.8]), orientation = "vertical")
cb.set_label("cos2", fontsize = 12, labelpad = 8)
```

```
plt.show()
```



Remarque : comme pour les variables, les données de $\cos^2$ et de contribution pour les individus ont été calculées sur R (puis stockées dans la table "ACP_contrib_ind.csv") selon le code ci-dessous.

```
library("factoextra")
library("FactoMineR")
res.pca <- PCA(iris, graph = FALSE)
fviz_pca_ind(res.pca)$data
```

```
[output]
  name         x          y    coord      cos2   contrib
     1 -2.264542  0.5057039 5.383886 0.9971473 0.9366442
     2 -2.086426 -0.6554047 4.782727 0.9871425 0.8320595
     3 -2.367950 -0.3184773 5.708617 0.9994005 0.9931383
     4 -2.304197 -0.5753677 5.640373 0.9974939 0.9812657
     5 -2.388777  0.6747674 6.161569 0.9996986 1.0719391
     6 -2.070537  1.5185486 6.593112 0.9998543 1.1470155
     [...]
```

1.15 Exercice 15

[28]:
```
# (15.1) Affichage de la contribution des individus aux deux premiers axes de␣
↪l'ACP
iris_pca_contrib_ind_pc = pd.read_csv(r"C:\Users\quent\Documents\DU Data␣
↪Analyst 2022\UE n°5 - Réduction de dimensionalité et algorithmes de␣
↪clustering\Jeux de données\ACP_contrib_ind_pc.csv", low_memory = False,␣
↪encoding = "latin-1")
iris_pca_contrib_ind_pc = iris_pca_contrib_ind_pc.reset_index()
iris_pca_contrib_ind_pc["Individus"] = iris_pca_contrib_ind_pc["index"]
iris_pca_contrib_ind_pc = iris_pca_contrib_ind_pc.drop(columns = "index")
iris_pca_contrib_ind_pc.index = iris_pca_contrib_ind_pc["Individus"]
iris_pca_contrib_ind_pc = iris_pca_contrib_ind_pc.drop(columns = "Individus")
iris_pca_contrib_ind_pc[["PC1", "PC2"]]
```

[28]:

Individus	PC1	PC2
0	1.174504	0.185071
1	0.997010	0.310860
2	1.284218	0.073401
3	1.215998	0.239572
4	1.306908	0.329498
...	...	...
145	0.801345	0.106057
146	0.556292	0.593121
147	0.529741	0.051511
148	0.433888	0.747552
149	0.210766	0.000359

[150 rows x 2 columns]

La contribution d'un individu aux deux premiers axes de l'ACP indique la qualité de la représentation de l'individu par chacun des deux axes.

Remarque : les données de contribution des individus aux deux premiers axes de l'ACP ont été calculées sur R (puis stockées dans la table "ACP_contrib_ind_pc.csv") selon le code ci-dessous.

```
library("factoextra")
library("FactoMineR")
res.pca <- PCA(iris, graph = FALSE)
res.pca$ind$contrib
```

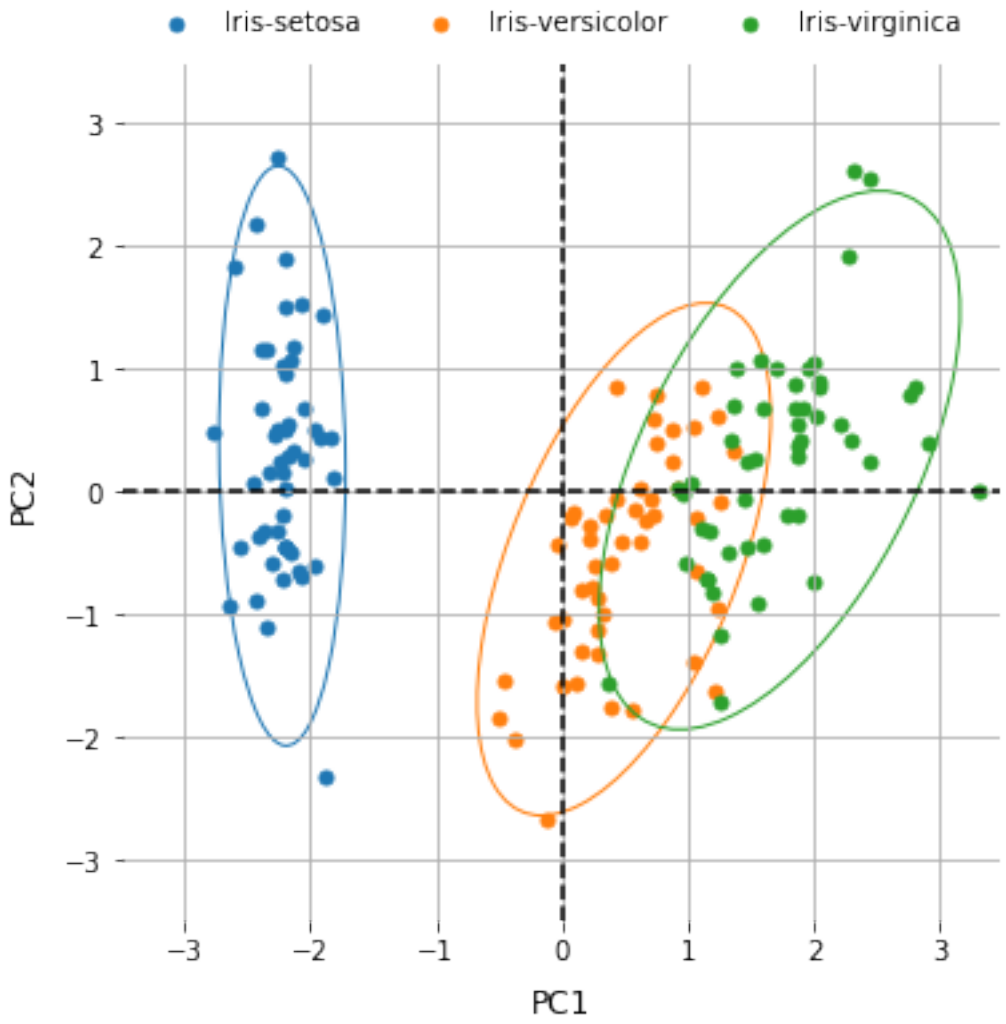
```
    [output]
                  Dim.1        Dim.2        Dim.3        Dim.4
        1   1.174504e+00 1.850707e-01 6.727678e-02 0.0172226261
        2   9.970098e-01 3.108595e-01 2.336468e-01 0.3445941072
        3   1.284218e+00 7.340099e-02 1.199001e-02 0.0250470415
        4   1.215998e+00 2.395719e-01 4.421749e-02 0.1422513099
        5   1.306908e+00 3.294983e-01 2.077333e-03 0.0452438193
        [...]
```

1.16 Exercice 16

[29]:
```
# (16.1) Définition de deux fonctions permettant la création d'ellipses de␣
↪confiance (source : https://matplotlib.org/stable/gallery/statistics/
↪confidence_ellipse.html)
def confidence_ellipse(x, y, ax, n_std=3.0, facecolor='none', **kwargs):
    """
    Create a plot of the covariance confidence ellipse of *x* and *y*.

    Parameters
    ----------
    x, y : array-like, shape (n, )
        Input data.

    ax : matplotlib.axes.Axes
        The axes object to draw the ellipse into.

    n_std : float
        The number of standard deviations to determine the ellipse's radiuses.

    **kwargs
        Forwarded to `~matplotlib.patches.Ellipse`

    Returns
    -------
    matplotlib.patches.Ellipse
    """
    if x.size != y.size:
        raise ValueError("x and y must be the same size")

    cov = np.cov(x, y)
    pearson = cov[0, 1]/np.sqrt(cov[0, 0] * cov[1, 1])
    # Using a special case to obtain the eigenvalues of this
    # two-dimensional dataset.
    ell_radius_x = np.sqrt(1 + pearson)
    ell_radius_y = np.sqrt(1 - pearson)
    ellipse = Ellipse((0, 0), width=ell_radius_x * 2, height=ell_radius_y * 2,
                      facecolor=facecolor, **kwargs)

    # Calculating the standard deviation of x from
    # the squareroot of the variance and multiplying
    # with the given number of standard deviations.
    scale_x = np.sqrt(cov[0, 0]) * n_std
    mean_x = np.mean(x)

    # calculating the standard deviation of y ...
    scale_y = np.sqrt(cov[1, 1]) * n_std
    mean_y = np.mean(y)

    transf = transforms.Affine2D() \
        .rotate_deg(45) \
```

```
        .scale(scale_x, scale_y) \
        .translate(mean_x, mean_y)

    ellipse.set_transform(transf + ax.transData)
    return ax.add_patch(ellipse)

def get_correlated_dataset(n, dependency, mu, scale):
    latent = np.random.randn(n, 2)
    dependent = latent.dot(dependency)
    scaled = dependent * scale
    scaled_with_offset = scaled + mu
    # return x and y of the new, correlated dataset
    return scaled_with_offset[:, 0], scaled_with_offset[:, 1]
```

[30]:
```
# (16.2) Représentation des individus sur les deux premières composantes, en␣
↪entourant par des ellipses les individus de même espèce
targets = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]
colors = ["C0", "C1", "C2"]
legend = {}
fig, ax = plt.subplots(figsize = (15/2.54, 15/2.54))
for target, color in zip(targets, colors):
    indicesToKeep = iris_pca_scores_cos2["Species"] == target
    ax.scatter(iris_pca_scores_cos2.loc[indicesToKeep, "PC1"],␣
↪iris_pca_scores_cos2.loc[indicesToKeep, "PC2"], c = color, s = 25)
    confidence_ellipse(iris_pca_scores_cos2.loc[indicesToKeep, "PC1"],␣
↪iris_pca_scores_cos2.loc[indicesToKeep, "PC2"], ax = ax, n_std = 2.5,␣
↪edgecolor = color)
    legend[str(target)] = ax.scatter(x = 100, y = 100, c = color, s = 25,␣
↪label = target)
ax.set_xlabel("PC1", fontsize = 12, labelpad = 8)
ax.set_ylabel("PC2", fontsize = 12, labelpad = 8)
ax.set_xlim(-3.49,3.49)
ax.set_ylim(-3.49,3.49)
ax.legend(handles = [legend["Iris-setosa"], legend["Iris-versicolor"],␣
↪legend["Iris-virginica"]], bbox_to_anchor = (0.98, 1.09), ncol = 3, frameon␣
↪= False)
ax.spines[:].set_visible(False)
ax.axline(xy1 = (0, 0), xy2 = (0, 1), linewidth = 1.5, c = "black", ls = "--")
ax.axline(xy1 = (0, 0), xy2 = (1, 0), linewidth = 1.5, c = "black", ls = "--")
ax.grid()
plt.show()
```



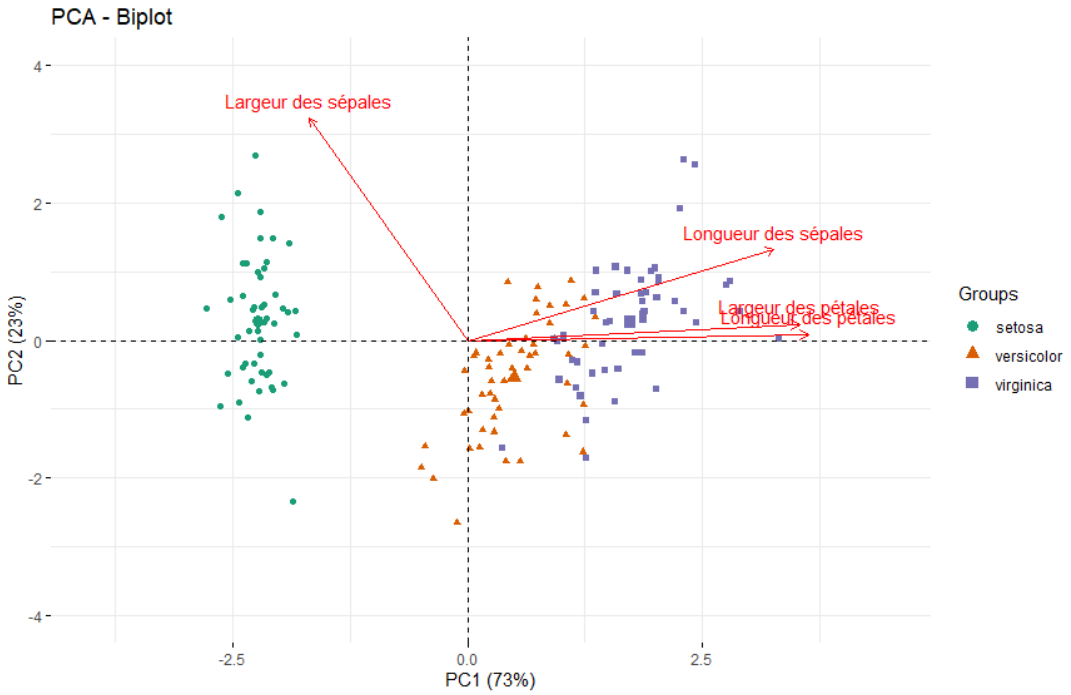
Les ellipses affichées dans ce graphique diffèrent de clusters définis par des algorithmes tels que le kmeans car ici les labels sont définis à l'avance (chaque groupe correspond à une espèce d'iris), alors que dans le kmeans l'algorithme définit lui-même les clusters sans intervention de l'utilisateur (apprentissage non-supervisé).

1.17 Exercice 17

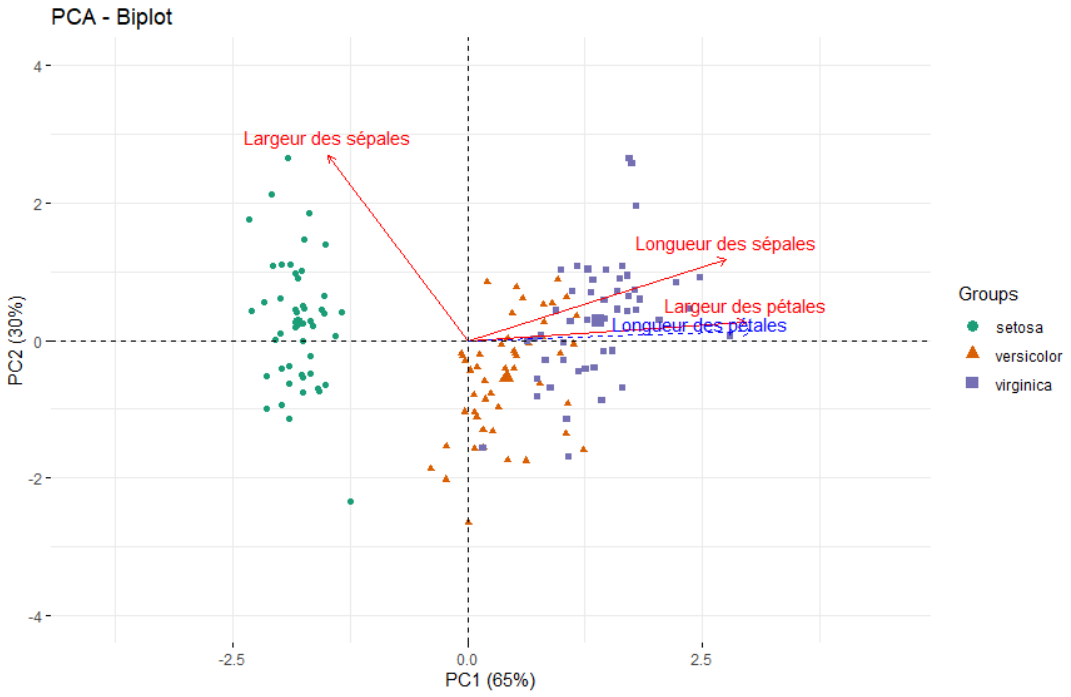
Cet exercice a été réalisé sous R avec le code ci-dessous :

```
library(data.table)
library("factoextra")
library("FactoMineR")

data(iris)
colnames(iris) <- c("Longueur des sépales", "Largeur des sépales", "Longueur des
   pétales", "Largeur des pétales", "Species")

# création d'un biplot avec les quatre variables initiales
iris.pca <- PCA(iris[,-5], graph = FALSE)
fviz_pca_biplot(iris.pca, habillage = iris$Species, col.var = "red",
```

```
   label = "var", xlim = c(-4,4), ylim = c(-4,4), xlab = "PC1 (73%)",
   ylab = "PC2 (23%)") + scale_color_brewer(palette = "Dark2")
```




```
# création d'un biplot en considérant la longueur des pétales comme une variable
supplémentaire (en bleu dans le graphique)
iris.pca.quant <- PCA(iris[,-5], graph = FALSE, quanti.sup=c(3))
fviz_pca_biplot(iris.pca.quant, habillage = iris$Species, col.var = "red",
   label = "var", xlim = c(-4,4), ylim = c(-4,4), xlab = "PC1 (65%)",
   ylab = "PC2 (30%)") + scale_color_brewer(palette = "Dark2")
```



La prise en compte de la longueur des pétales comme variable supplémentaire induit comme principal changement la diminution de la variance expliquée par la première composante (65% au lieu de 73%) et une augmentation de la variance expliquée par la seconde (30% au lieu de 23%). Un deuxième changement est la diminution de la dispersion des fleurs dans le nouveau biplot et un rapprochement des groupes de chaque espèce. Cela montre que la variabilité entre les fleurs et entre espèces est mieux représentée lorsque la longueur des pétales est prise en compte dans l'ACP.

1.18 Exercice 18

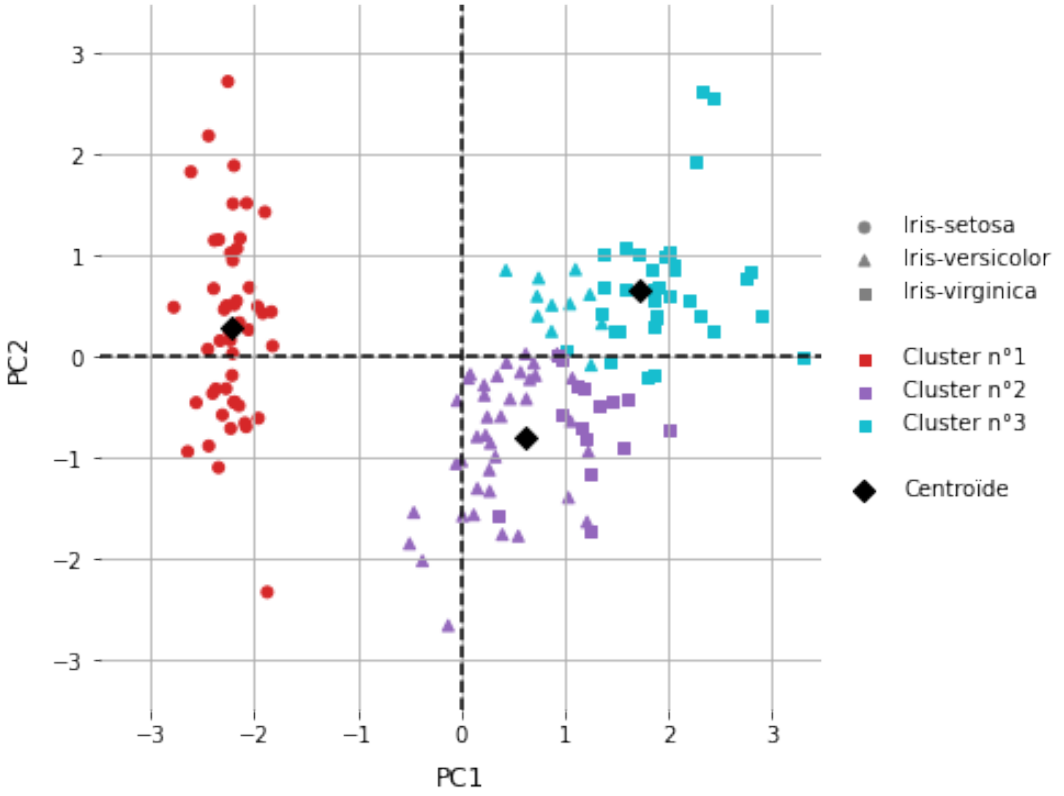
Le k-means est un algorithme de clustering faisant partie de l'apprentissage non-supervisé. Il permet de répartir des individus dans des clusters non-labellisés, dont le nombre est défini au préalable par l'utilisateur. Le fonctionnement du k-means dans un plan en deux dimensions suit les étapes suivantes : (1) un ou plusieurs centroïdes sont ajoutés sur le plan, soit de manière complètement aléatoire, soit en les éloignant le plus possible les uns des autres ; (2) les points sont affectés au cluster associé au centroïde le plus proche (étape d'expectation) ; (3) les coordonnées du centroïde de chaque nouveau cluster sont calculées pour y déplacer le centroïde d'origine (étape de maximisation) ; (4) ces deux dernières étapes sont de nouveau exécutées (nouvelle itération) jusqu'à ce que la position des centroïdes reste fixe entre deux itérations. Ainsi, au fil des itérations, les centroïdes vont suivre une trajectoire les amenant progressivement de leur position aléatoire initiale au centre de leurs clusters respectifs.

1.19 Exercice 19

[33]:
```
# (19.1) Application de l'algorithme de k-means pour classifier les fleurs␣
↪dans trois clusters distincts
my_model = KMeans(n_clusters = 3)
my_model.fit(iris_pca_scores[["PC1", "PC2"]])
labels = my_model.predict(iris_pca_scores[["PC1", "PC2"]])
centroids = my_model.cluster_centers_
iris_pca_scores_avec_clusters = iris_pca_scores.copy(deep = True)
iris_pca_scores_avec_clusters["Cluster"] = list(labels)
```

[34]:
```
# (19.2) Représentation des fleurs dans le plan de l'ACP en distinguant les␣
↪clusters par couleur et les espèces par type de marqueur
targets = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]
markers = ["o", "^", "s"]
colors = ["C3", "C4", "C9"]
couleur_cluster = []
for i in iris_pca_scores_avec_clusters["Cluster"]:
    couleur_cluster.append(colors[i])
iris_pca_scores_avec_clusters["Couleur_cluster"] = couleur_cluster
legend = {}
fig, ax = plt.subplots(figsize = (15/2.54, 15/2.54))
for target, marker, color in zip(targets, markers, colors):
    indicesToKeep = iris_pca_scores_avec_clusters["Species"] == target
```

```
    ax.scatter(iris_pca_scores_avec_clusters.loc[indicesToKeep, "PC1"],␣
 ↪iris_pca_scores_avec_clusters.loc[indicesToKeep, "PC2"], c = np.
 ↪array(iris_pca_scores_avec_clusters.loc[indicesToKeep, "Couleur_cluster"]),␣
 ↪s = 25, marker = marker)
    legend[str(target)] = ax.scatter(x = 100, y = 100, c = "grey", marker =␣
 ↪marker, s = 25, label = target)
for i in iris_pca_scores_avec_clusters["Cluster"].unique():
    legend["Cluster n°" + str(i+1)] = ax.scatter(x = 100, y = 100, c =␣
 ↪colors[i], s = 25, marker = "s", label = "Cluster n°" + str(i+1))
centroids_points = ax.scatter(centroids[:,0], centroids[:,1], marker = "D", s␣
 ↪= 50, c = "black", label = "Centroïde")
ax.set_xlabel("PC1", fontsize = 12, labelpad = 8)
ax.set_ylabel("PC2", fontsize = 12, labelpad = 8)
ax.set_xlim(-3.49,3.49)
ax.set_ylim(-3.49,3.49)
empty_legend = ax.scatter(x = 100, y = 100, c = "white", label = " ")
ax.legend(handles = [legend["Iris-setosa"], legend["Iris-versicolor"],␣
 ↪legend["Iris-virginica"], empty_legend, legend["Cluster n°1"],␣
 ↪legend["Cluster n°2"], legend["Cluster n°3"], empty_legend,␣
 ↪centroids_points], bbox_to_anchor = (1, 0.73), frameon = False)
ax.spines[:].set_visible(False)
ax.axline(xy1 = (0, 0), xy2 = (0, 1), linewidth = 1.5, c = "black", ls = "--")
ax.axline(xy1 = (0, 0), xy2 = (1, 0), linewidth = 1.5, c = "black", ls = "--")
ax.grid()
plt.show()
```



[35]:
```
# (19.3) Calcul des caractéristiques de chaque cluster :
# ___(19.3.1) moyenne et erreur standard de chacune des quatre variables␣
 ↪quantitatives initiales
moyennes_clusters = {}
sem_clusters = {}
variables_eng = ["SepalLengthCm", "SepalWidthCm", "PetalLengthCm",␣
 ↪"PetalWidthCm"]
variables_fr = ["Long_sépales", "Larg_sépales", "Long_pétales",␣
 ↪"Larg_pétales"]
culsters_names = ["Cluster n°1", "Cluster n°2", "Cluster n°3"]
for i, j in zip(variables_eng, variables_fr):
    moy = []
    sem = []
    for k in range(3):
        moy.append(iris.
 ↪loc[list(iris_pca_scores_avec_clusters[iris_pca_scores_avec_clusters␣
 ↪["Cluster"] == k].index), i].mean())
        sem.append(iris.
 ↪loc[list(iris_pca_scores_avec_clusters[iris_pca_scores_avec_clusters␣
 ↪["Cluster"] == k].index), i].sem())
    moyennes_clusters[str(j)] = moy
    sem_clusters[str(j)] = sem
df_moyennes_clusters = pd.DataFrame(moyennes_clusters, index = culsters_names)
df_sem_clusters = pd.DataFrame(sem_clusters, index = culsters_names)
print("Moyenne des variables initiales pour chaque cluster :")
print(df_moyennes_clusters)
print("")
print("Erreurs standards :")
print(df_sem_clusters)
print("")

# ___(19.3.2) proportion d'individus de chaque espèce
prop_sp_par_cluster = pd.crosstab(iris_pca_scores_avec_clusters["Species"],␣
 ↪iris_pca_scores_avec_clusters["Cluster"], normalize = "columns").transpose()
prop_sp_par_cluster.index = culsters_names
print("Proportion d'individus de chaque espèce :")
print(prop_sp_par_cluster)
```

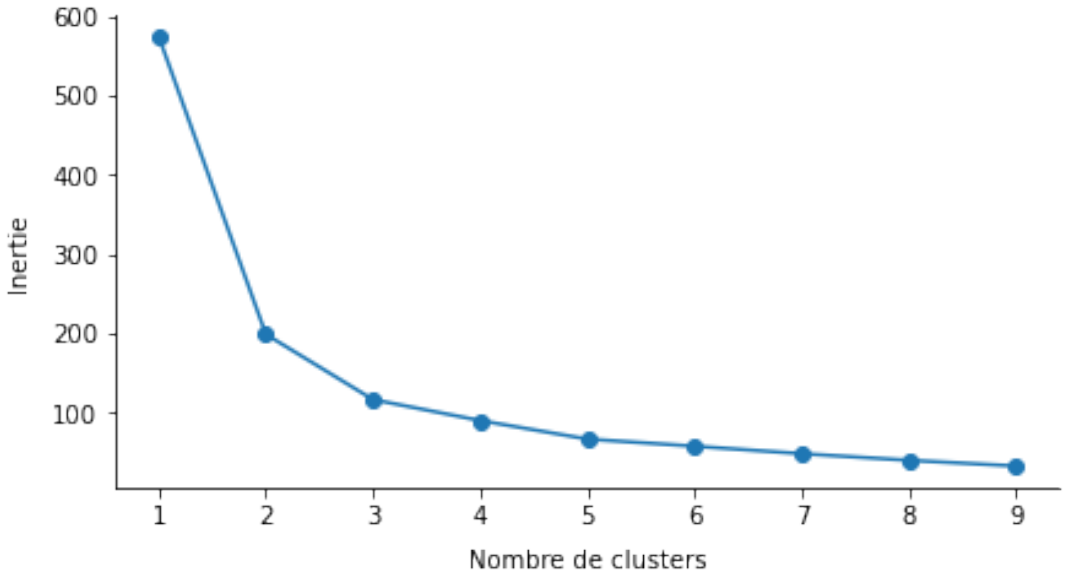
```
Moyenne des variables initiales pour chaque cluster :
             Long_sépales  Larg_sépales  Long_pétales  Larg_pétales
Cluster n°1      5.006000      3.418000      1.464000      0.244000
Cluster n°2      5.829091      2.670909      4.412727      1.429091
Cluster n°3      6.791111      3.117778      5.508889      1.977778

Erreurs standards :
             Long_sépales  Larg_sépales  Long_pétales  Larg_pétales
Cluster n°1      0.049850      0.053885      0.024538      0.015162
Cluster n°2      0.057071      0.033802      0.080280      0.042693
Cluster n°3      0.074252      0.036416      0.096913      0.050241
```

```
Proportion d'individus de chaque espèce :
Species      Iris-setosa  Iris-versicolor  Iris-virginica
Cluster n°1          1.0         0.000000        0.000000
Cluster n°2          0.0         0.709091        0.290909
Cluster n°3          0.0         0.244444        0.755556
```

Les trois clusters définis par l'algorithme de k-means diffèrent d'une part selon la moyenne de chaque variable initiale (longueur et largeur des sépales et des pétales) et d'autre part selon la proportion d'individus de chaque espèce dans le cluster. Le cluster 2 regroupe tous les individus de l'espèce Iris setosa, caractérisés par des sépales de 5cm de long et 3.4cm de large et des pétales de 1.5cm de long et 0.2cm de large. Les deux autres clusters contiennent à la fois des individus de l'espèce Iris versicolor (74% pour le cluster 1 et 23% pour le cluster 3) et des individus de l'espèce Iris virginica (26% pour le cluster 1 et 77% pour le cluster 3). Les fleurs du cluster 3 ont des sépales et des pétales 1 à 1.2cm plus longs et 0.4 à 0.5cm plus larges en moyenne que les fleurs du cluster 1.

1.20 Exercice 20

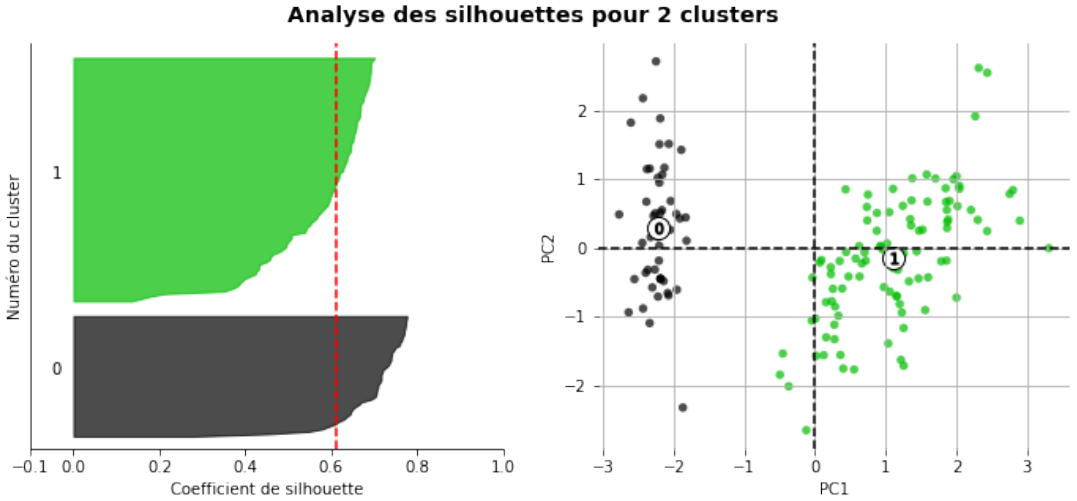
[36]:
```
# (20.1) Représentation de l'évolution de la variance intra-cluster avec le␣
↪nombre de clusters (scree plot)
inertias = []
ks = range(1, 10)
for k in ks:
    model = KMeans(n_clusters = k)
    model.fit(iris_pca_scores[["PC1", "PC2"]])
    inertias.append(model.inertia_)
fig, ax = plt.subplots(figsize = (17.5/2.54, 9/2.54))
ax.plot(ks, inertias, '-o')
ax.set_xlabel("Nombre de clusters", labelpad = 8)
ax.set_ylabel("Inertie", labelpad = 8)
ax.spines[["right", "top"]].set_visible(False)
plt.show()
```

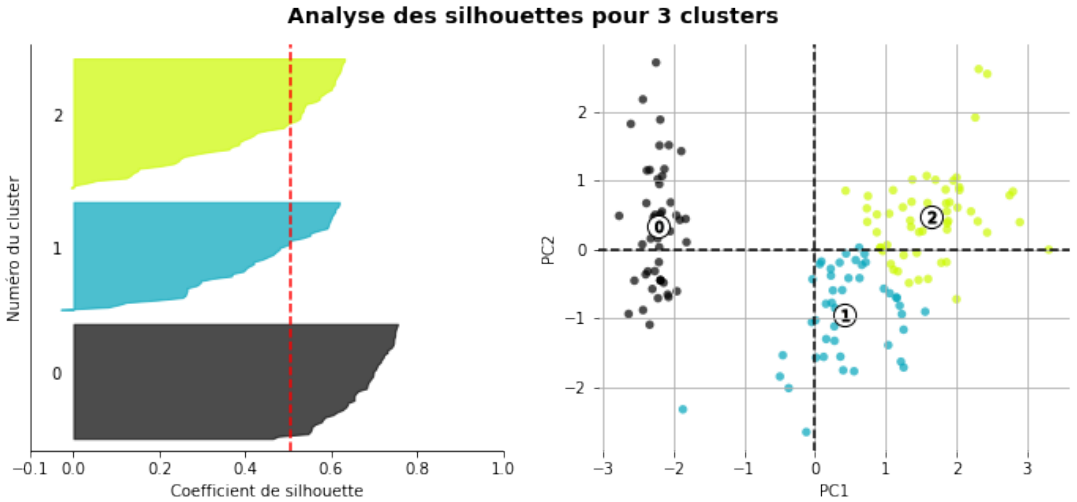


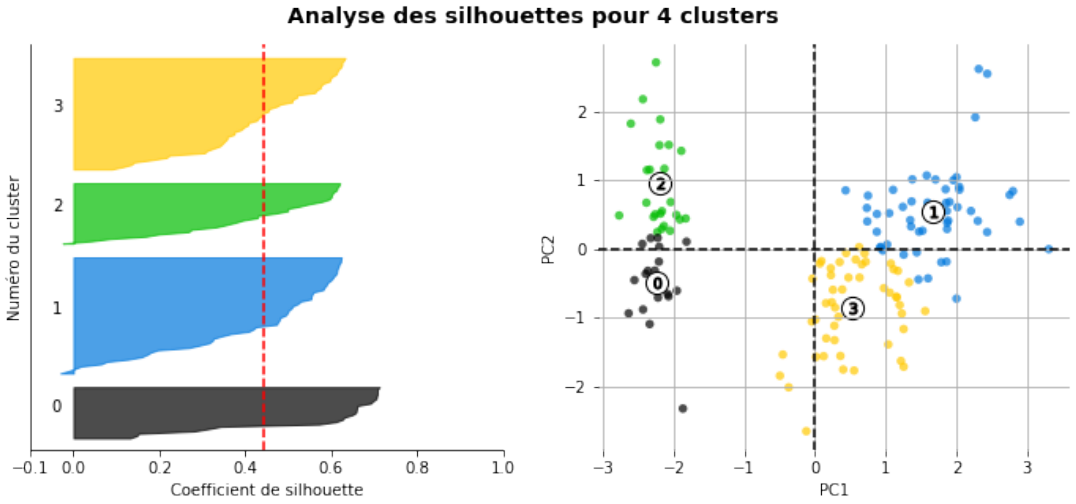
Le principe de la méthode du coude (ou des bâtons brisés) est d'identifier à partir de quel nombre de clusters la variance intra-cluster (ou "inertie") cesse de chuter fortement. Ce nombre de clusters correspondra au nombre optimal du clusters à retenir. D'après le scree plot ci-dessous, le nombre optimal de clusters est de trois : au-delà, l'inertie varie très faiblement avec l'ajout de nouveaux de clusters.

[37]:
```
# (20.2) Représentation des silhouettes pour un nombre de clusters variant␣
↪entre 2 et 4 (source : https://scikit-learn.org/stable/auto_examples/
↪cluster/plot_kmeans_silhouette_analysis.html)
range_n_clusters = [2, 3, 4]
for n_clusters in range_n_clusters:
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (30/2.54, 12/2.54))
    ax1.set_xlim([-0.1, 1])
    ax1.set_ylim([0, len(iris_pca_scores[["PC1", "PC2"]]) + (n_clusters + 1)␣
↪* 5])
    clusterer = KMeans(n_clusters = n_clusters, random_state = 10)
    cluster_labels = clusterer.fit_predict(iris_pca_scores[["PC1", "PC2"]])
    silhouette_avg = silhouette_score(iris_pca_scores[["PC1", "PC2"]],␣
↪cluster_labels)
    sample_silhouette_values = silhouette_samples(iris_pca_scores[["PC1",␣
↪"PC2"]], cluster_labels)
    y_lower = 5
    for i in range(n_clusters):
        ith_cluster_silhouette_values =␣
↪sample_silhouette_values[cluster_labels == i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper), 0,␣
↪ith_cluster_silhouette_values, facecolor = color, edgecolor = color, alpha␣
↪= 0.7)
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
        y_lower = y_upper + 5
    ax1.set_xlabel("Coefficient de silhouette")
    ax1.set_ylabel("Numéro du cluster")
    ax1.axvline(x = silhouette_avg, color = "red", linestyle = "--")
    ax1.set_yticks([])
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
    ax1.spines[["right", "top"]].set_visible(False)
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(iris_pca_scores["PC1"], iris_pca_scores["PC2"], marker = "o",␣
↪s = 30, lw = 0, alpha = 0.7, c = colors, edgecolor = "k")
    centers = clusterer.cluster_centers_
    ax2.scatter(centers[:, 0], centers[:, 1], marker = "o", c = "white",␣
↪alpha = 1, s = 200, edgecolor = "k")
    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker = "$%d$" % i, alpha = 1, s = 50,␣
↪edgecolor = "k")
```

```
    ax2.set_xlabel("PC1")
    ax2.set_ylabel("PC2")
    ax2.spines[:].set_visible(False)
    ax2.axline(xy1 = (0, 0), xy2 = (0, 1), linewidth = 1.5, c = "black", ls =␣
 ↪"--")
    ax2.axline(xy1 = (0, 0), xy2 = (1, 0), linewidth = 1.5, c = "black", ls =␣
 ↪"--")
    ax2.grid()
    plt.suptitle("Analyse des silhouettes pour %d clusters" % n_clusters, y =␣
 ↪0.95, fontsize = 14, fontweight = "bold")
plt.show()
```







L'analyse des silhouettes est une autre méthode permettant de définir le nombre de clusters à retenir dans un clustering. Le principe est de calculer, pour chaque point, la distance moyenne entre ce point et tous les autres de son cluster, puis la distance moyenne entre ce point et ceux des autres clusters. Le coefficient de silhouette est la différence entre entre ces deux distances : s'il est positif, cela indique que le point est plus proche des points de son cluster que des autres et qu'il est donc bien classé, et inversement pour un coefficient négatif.

Le graphique de silhouette peut permettre de déterminer si le nombre de clusters choisi est adéquat, en comparant les coefficients de silhouette entre les clusters. En général, un nombre de clusters sous-optimal se traduit par un ou plusieurs clusters dont les coefficients de silhouette sont inférieurs à la moyenne de tous les coefficients. Dans les graphiques ci-dessus, aucun cluster n'est dans ce cas, quel que soit le nombre de clusters initiaux. Un clustering inadéquat peut également se traduire par des fluctuations fortes dans la taille des coefficients de silhouette entre les clusters, ce qui n'est pas retrouvé non plus ici. Enfin, des disparités dans la taille des clusters (représentée par l'épaisseur des silhouettes) peuvent aussi suggérer un clustering sous-optimal. Dans les graphiques précédents, des disparités sont retrouvées lorsque deux ou quatre clusters sont choisis, mais pas pour un nombre de trois clusters. Ce nombre semble donc le plus optimal, comme l'indique aussi le scree plot.

2 Projet 2 : S'entraîner sur les données de décathlon

2.1 Exercice 1

[39]:
```
# (1.1) Import du jeu de données "decathlon.csv"
decat_orig = pd.read_csv(r"C:\Users\quent\Documents\DU Data Analyst 2022\UE␣
↪n°5 - Réduction de dimensionalité et algorithmes de clustering\Jeux de␣
↪données\decathlon.csv", low_memory = False, encoding = "latin-1")
decat = decat_orig.copy(deep = True)
decat.head()
```

[39]:

	Athlets	100m	Long.jump	Shot.put	High.jump	400m	110m.hurdle	Discus
0	SEBRLE	11.04	7.58	14.83	2.07	49.81	14.69	43.75
1	CLAY	10.76	7.40	14.26	1.86	49.37	14.05	50.72
2	KARPOV	11.02	7.30	14.77	2.04	48.37	14.09	48.95
3	BERNARD	11.02	7.23	14.25	1.92	48.93	14.99	40.87
4	YURKOV	11.34	7.09	15.19	2.10	50.42	15.31	46.26

	Pole.vault	Javeline	1500m	Rank	Points	Competition
0	5.02	63.19	291.7	1	8217	Decastar
1	4.92	60.15	301.5	2	8122	Decastar
2	4.92	50.31	300.2	3	8099	Decastar
3	5.32	62.77	280.1	4	8067	Decastar
4	4.72	63.44	276.4	5	8036	Decastar

[40]:
```
# (1.2) Inspection des variables
print(decat.info())
print()
print("Nom des athlètes :")
print(decat["Athlets"].unique())
print()
print("Nom des compétitions :")
print(decat["Competition"].unique())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41 entries, 0 to 40
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Athlets      41 non-null     object
 1   100m         41 non-null     float64
 2   Long.jump    41 non-null     float64
 3   Shot.put     41 non-null     float64
 4   High.jump    41 non-null     float64
 5   400m         41 non-null     float64
 6   110m.hurdle  41 non-null     float64
 7   Discus       41 non-null     float64
 8   Pole.vault   41 non-null     float64
 9   Javeline     41 non-null     float64
 10  1500m        41 non-null     float64
 11  Rank         41 non-null     int64
```

```
 12  Points       41 non-null     int64
 13  Competition  41 non-null     object
dtypes: float64(10), int64(2), object(2)
memory usage: 4.6+ KB
None

Nom des athlètes :
['SEBRLE' 'CLAY' 'KARPOV' 'BERNARD' 'YURKOV' 'WARNERS' 'ZSIVOCZKY'
 'McMULLEN' 'MARTINEAU' 'HERNU' 'BARRAS' 'NOOL' 'BOURGUIGNON' 'Sebrle'
 'Clay' 'Karpov' 'Macey' 'Warners' 'Zsivoczky' 'Hernu' 'Nool' 'Bernard'
 'Schwarzl' 'Pogorelov' 'Schoenbeck' 'Barras' 'Smith' 'Averyanov'
 'Ojaniemi' 'Smirnov' 'Qi' 'Drews' 'Parkhomenko' 'Terek' 'Gomez' 'Turi'
 'Lorenzo' 'Karlivans' 'Korkizoglou' 'Uldal' 'Casarsa']

Nom des compétitions :
['Decastar' 'OlympicG']
```

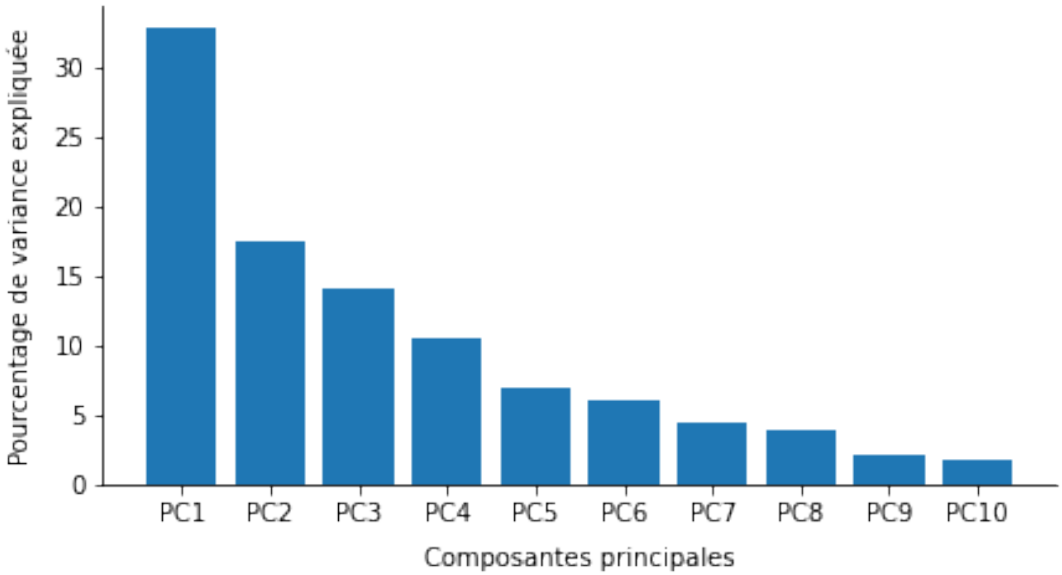
[41]:
```
# (1.3) Réalisation d'une ACP et affichage des coordonnées des individus dans␣
 ↪les nouvelles composantes
disciplines_decat = ["100m", "Long.jump", "Shot.put", "High.jump", "400m",␣
 ↪"110m.hurdle", "Discus", "Pole.vault", "Javeline", "1500m"]
disciplines_decat_fr = ["100m", "Saut en longueur", "Lancer du poids", "Saut␣
 ↪en hauteur", "400m", "110m haies", "Lancer du disque", "Saut à la perche",␣
 ↪"Lancer du javelot", "1500m"]
scaler = StandardScaler()
pca_decat = PCA()
pipeline_decat = make_pipeline(scaler, pca_decat)
pca_decat_scores = pipeline_decat.fit_transform(decat[disciplines_decat])
pc_list_decat = ["PC" + str(i) for i in list(range(1,␣
 ↪len(pca_decat_scores[0])+1))]
df_pca_decat_scores = pd.DataFrame(data = pca_decat_scores, columns =␣
 ↪pc_list_decat)
df_pca_decat_scores = pd.concat([decat[["Athlets", "Rank", "Points",␣
 ↪"Competition"]], df_pca_decat_scores], axis = 1)
df_pca_decat_scores.head()
```

[41]:

	Athlets	Rank	Points	Competition	PC1	PC2	PC3	PC4
0	SEBRLE	1	8217	Decastar	0.791628	0.771611	0.826841	1.174627
1	CLAY	2	8122	Decastar	1.234991	0.574578	2.141247	-0.354845
2	KARPOV	3	8099	Decastar	1.358215	0.484021	1.956258	-1.856524
3	BERNARD	4	8067	Decastar	-0.609515	-0.874629	0.889941	2.220612
4	YURKOV	5	8036	Decastar	-0.585968	2.130954	-1.225157	0.873579

	PC5	PC6	PC7	PC8	PC9	PC10
0	-0.707159	-1.030620	-0.551523	-0.435655	-0.137559	0.500774
1	1.974571	0.690126	-0.707974	-0.603419	-0.649244	-0.266119
2	-0.795215	0.732751	-0.189939	-0.250297	-0.800654	0.523269
3	-0.361636	0.275598	0.049611	0.067458	-0.723281	0.188459
4	-1.251369	-0.104606	-0.573925	0.094604	-0.202216	0.056443

[42]:
```
# (1.4) Représentation du scree plot
fig, ax = plt.subplots(figsize = (17.5/2.54, 9/2.54))
ax.bar(x = pc_list_decat, height = pca_decat.explained_variance_ratio_*100)
ax.set_xlabel("Composantes principales", labelpad = 8)
ax.set_ylabel("Pourcentage de variance expliquée", labelpad = 8)
ax.spines[["right", "top"]].set_visible(False)
plt.show()
```



[43]:
```
# (1.5) Création d'un tableau indiquant pour chaque composante les valeurs␣
↪propres, le pourcentage de variance expliquée et le pourcentage cumulé de␣
↪variance expliquée
df_decat_eigenvalues = pd.DataFrame({"Valeurs propres" : pca_decat.
↪explained_variance_, "Variance expliquée (%)" : pca_decat.
↪explained_variance_ratio_*100, "Variance expliquée cumulée (%)" : np.
↪cumsum(pca_decat.explained_variance_ratio_*100)})
df_decat_eigenvalues.index = pc_list_decat
df_decat_eigenvalues
```

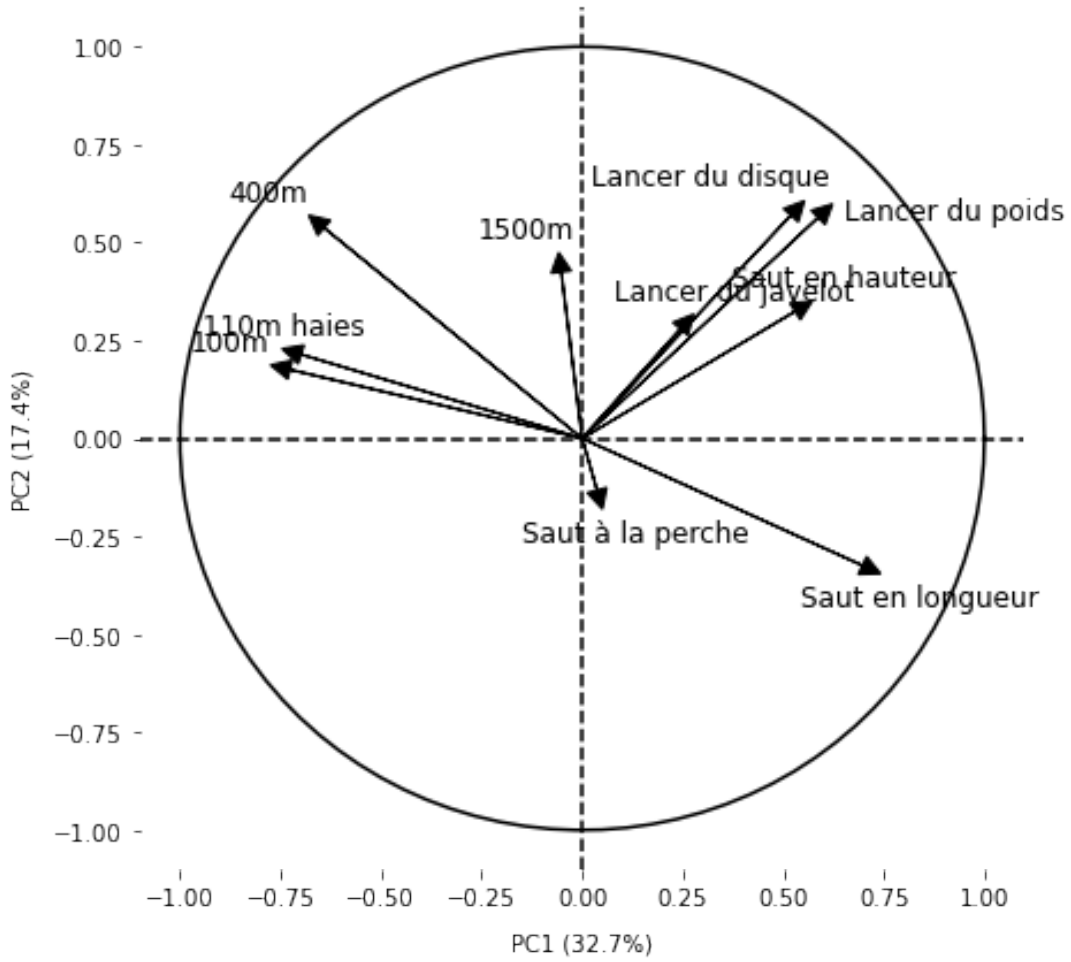
[43]:

	Valeurs propres	Variance expliquée (%)	Variance expliquée cumulée (%)
PC1	3.353703	32.719055	32.719055
PC2	1.780559	17.371310	50.090366
PC3	1.440040	14.049167	64.139532
PC4	1.083272	10.568504	74.708036
PC5	0.701893	6.847735	81.555771
PC6	0.614250	5.992687	87.548458
PC7	0.462516	4.512353	92.060811
PC8	0.406799	3.968766	96.029577
PC9	0.220185	2.148149	98.177725
PC10	0.186783	1.822275	100.000000

[44]:
```
# (1.6) Calcul de la saturation des variables par composante
loadings_decat = pca_decat.components_
loadings_decat_df = pd.DataFrame.from_dict(dict(zip(pc_list_decat,␣
↪loadings_decat)))
loadings_decat_df.index = disciplines_decat_fr
loadings_decat_df.head()
```

[44]:
```
                      PC1       PC2       PC3       PC4       PC5       PC6 ␣
↪\
100m             -0.428296  0.141989 -0.155580 -0.036787 -0.365187  0.296077
Saut en longueur  0.410152 -0.262079  0.153727  0.099010 -0.044323 -0.306125
Lancer du poids   0.344144  0.453947 -0.019724  0.185395 -0.134320  0.305473
Saut en hauteur   0.316194  0.265776 -0.218943 -0.131897 -0.671218 -0.467771
400m             -0.375716  0.432046  0.110918  0.028503  0.105970 -0.332522

                      PC7       PC8       PC9      PC10
100m             -0.381776 -0.461602  0.104758  0.424283
Saut en longueur -0.627693  0.021012  0.482669  0.081044
Lancer du poids   0.309725  0.313930  0.427291  0.390284
Saut en hauteur   0.091450 -0.125092 -0.243661 -0.106427
400m              0.124421 -0.213398  0.552129 -0.413995
```

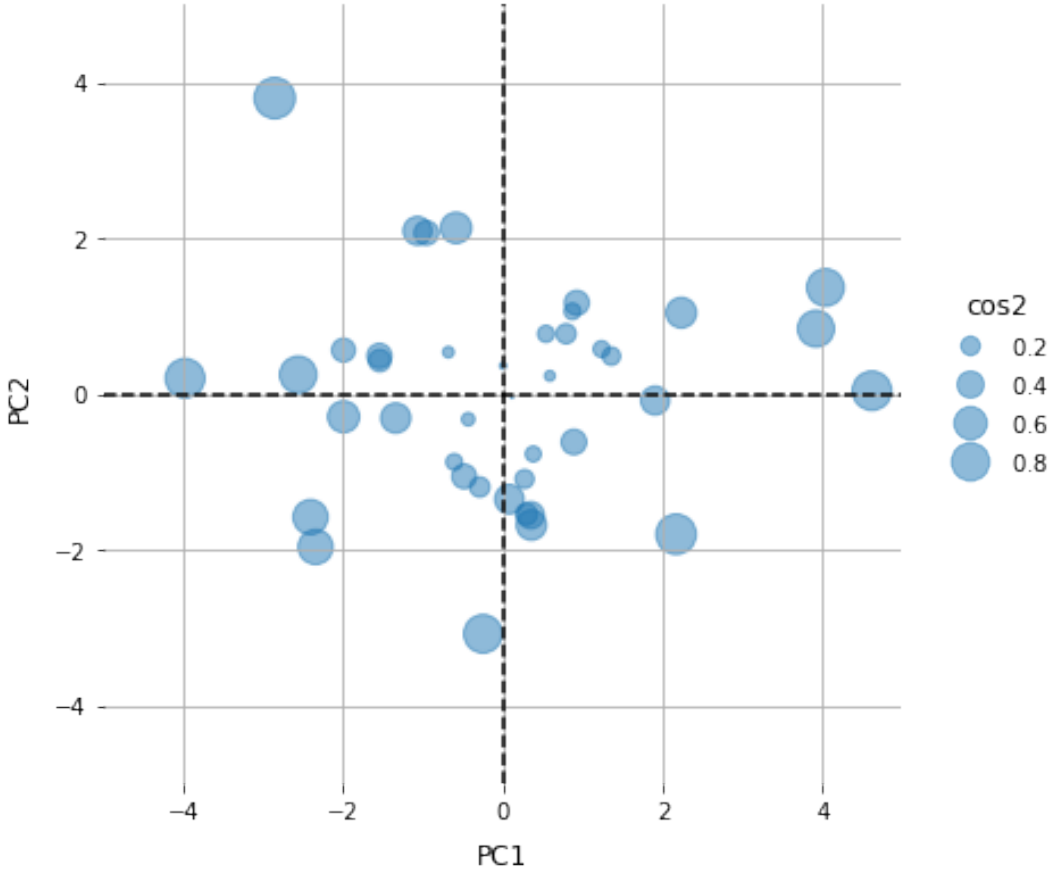
[45]:
```
# (1.7) Représentation du cercle des corrélations (import des coordonnées␣
↪obtenues sous R)
decat_pca_var = pd.read_csv(r"C:\Users\quent\Documents\DU Data Analyst␣
↪2022\UE n°5 - Réduction de dimensionalité et algorithmes de clustering\Jeux␣
↪de données\ACP_var_decat.csv", low_memory = False, encoding = "latin-1")
decat_pca_var["name_fr"] = disciplines_decat_fr
fig, ax = plt.subplots(figsize = (17.5/2.54, 17.5/2.54))
for i in range(0, decat_pca_var.shape[0]):
    ax.arrow(0, 0, decat_pca_var["x"][i], decat_pca_var["y"][i], head_width =␣
↪0.05, head_length = 0.05, length_includes_head = True, color = "black")
circle = np.linspace(0, 2 * np.pi, 100)
ax.plot(np.cos(circle), np.sin(circle), color = "black")
ax.set_xlabel("PC1 (32.7%)", labelpad = 10)
ax.set_ylabel("PC2 (17.4%)", labelpad = 8)
for i in decat_pca_var["name_fr"][[0,1,3,4,5,7,8,9]]:
    if decat_pca_var[decat_pca_var["name_fr"] == i]["y"].values > 0:
        ax.text(decat_pca_var[decat_pca_var["name_fr"] == i]["x"]-0.2,␣
↪decat_pca_var[decat_pca_var["name_fr"] == i]["y"]+0.04, i, fontsize = 12)
    else:
        ax.text(decat_pca_var[decat_pca_var["name_fr"] == i]["x"]-0.2,␣
↪decat_pca_var[decat_pca_var["name_fr"] == i]["y"]-0.08, i, fontsize = 12)
ax.text(0.65, 0.56, decat_pca_var["name_fr"][2], fontsize = 12)
ax.text(0.02, 0.65, decat_pca_var["name_fr"][6], fontsize = 12)
ax.spines[:].set_visible(False)
ax.axline(xy1 = (0, 0), xy2 = (0, 1), linewidth = 1.5, c = "black", ls = "--")
ax.axline(xy1 = (0, 0), xy2 = (1, 0), linewidth = 1.5, c = "black", ls = "--")
plt.show()
```



2.2 Exercice 2

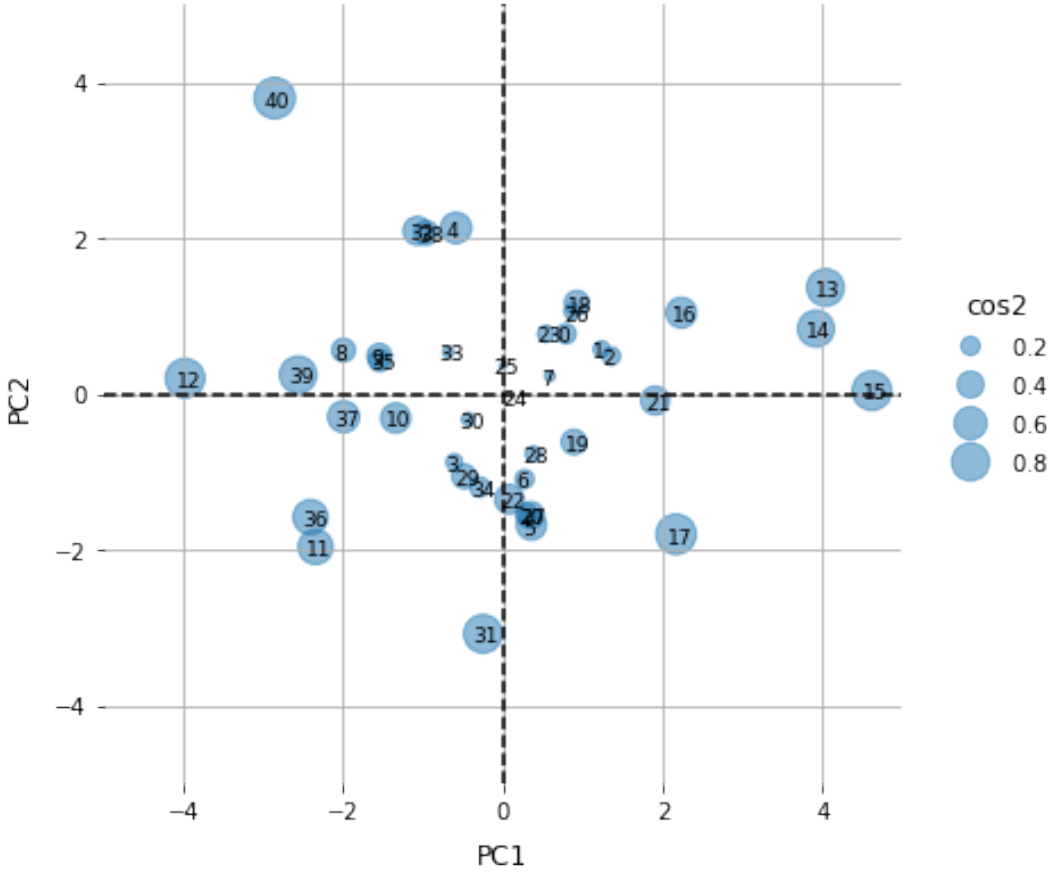
[46]:
```
# (2.1) Représentation des individus sur les deux premières composantes, en␣
 ↪représentant la qualité de la représentation (cos2) par la taille des␣
 ↪points
decat_pca_ind = pd.read_csv(r"C:\Users\quent\Documents\DU Data Analyst␣
 ↪2022\UE n°5 - Réduction de dimensionalité et algorithmes de clustering\Jeux␣
 ↪de données\ACP_ind_decat.csv", low_memory = False, encoding = "latin-1")
df_pca_decat_scores_cos2 = df_pca_decat_scores.copy(deep = True)
df_pca_decat_scores_cos2["cos2"] = decat_pca_ind["cos2"]
fig, ax = plt.subplots(figsize = (16/2.54, 16/2.54))
sc = ax.scatter(df_pca_decat_scores_cos2["PC1"],␣
 ↪df_pca_decat_scores_cos2["PC2"], marker = "o", alpha = 0.5, s =␣
 ↪df_pca_decat_scores_cos2["cos2"]*350, label =␣
 ↪df_pca_decat_scores_cos2["cos2"])
ax.set_xlabel("PC1", fontsize = 12, labelpad = 8)
ax.set_ylabel("PC2", fontsize = 12, labelpad = 8)
ax.set_xlim(-5,5)
ax.set_ylim(-5,5)
ax.spines[:].set_visible(False)
```

```
ax.axline(xy1 = (0, 0), xy2 = (0, 1), linewidth = 1.5, c = "black", ls = "--")
ax.axline(xy1 = (0, 0), xy2 = (1, 0), linewidth = 1.5, c = "black", ls = "--")
kw = dict(prop = "sizes", num = 5, color = "C0", func = lambda s: s/350)
ax.legend(*sc.legend_elements(**kw), title = "cos2", title_fontsize = 12,␣
 ↪bbox_to_anchor = (1.03, 0.66), frameon = False)
ax.grid()
plt.show()
```



[47]:
```
# (2.2) Même représentation en affichant l'identifiant de chaque athlète
decat_pca_ind = pd.read_csv(r"C:\Users\quent\Documents\DU Data Analyst␣
 ↪2022\UE n°5 - Réduction de dimensionalité et algorithmes de clustering\Jeux␣
 ↪de données\ACP_ind_decat.csv", low_memory = False, encoding = "latin-1")
df_pca_decat_scores_cos2 = df_pca_decat_scores.copy(deep = True)
df_pca_decat_scores_cos2["cos2"] = decat_pca_ind["cos2"]
fig, ax = plt.subplots(figsize = (16/2.54, 16/2.54))
sc = ax.scatter(df_pca_decat_scores_cos2["PC1"],␣
 ↪df_pca_decat_scores_cos2["PC2"], marker = "o", alpha = 0.5, s =␣
 ↪df_pca_decat_scores_cos2["cos2"]*350, label =␣
 ↪df_pca_decat_scores_cos2["cos2"])
ax.set_xlabel("PC1", fontsize = 12, labelpad = 8)
ax.set_ylabel("PC2", fontsize = 12, labelpad = 8)
ax.set_xlim(-5,5)
```

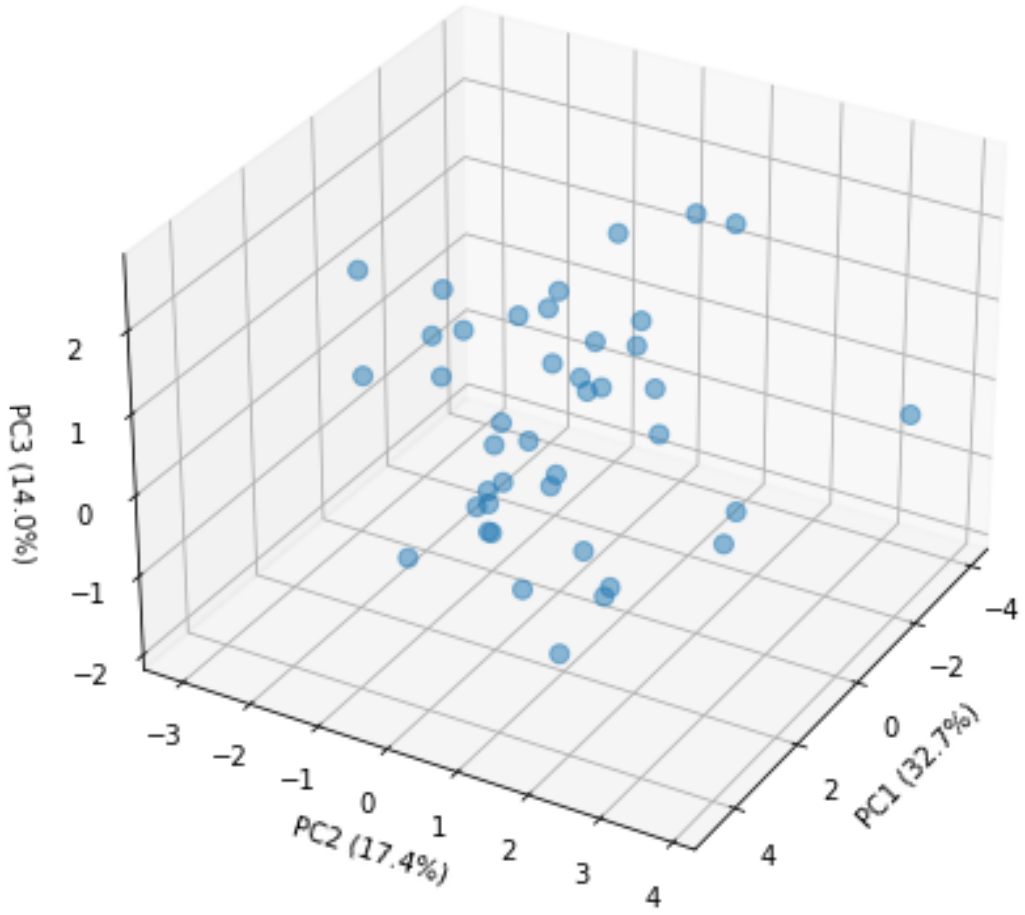
```
ax.set_ylim(-5,5)
for i in df_pca_decat_scores_cos2["Athlets"]:
    ax.text(df_pca_decat_scores_cos2[df_pca_decat_scores_cos2["Athlets"] ==␣
 ↪i]["PC1"]-0.12,␣
 ↪df_pca_decat_scores_cos2[df_pca_decat_scores_cos2["Athlets"] == i]["PC2"]-0.
 ↪1, df_pca_decat_scores_cos2[df_pca_decat_scores_cos2["Athlets"] == i].
 ↪index[0], fontsize = 9)
ax.spines[:].set_visible(False)
ax.axline(xy1 = (0, 0), xy2 = (0, 1), linewidth = 1.5, c = "black", ls = "--")
ax.axline(xy1 = (0, 0), xy2 = (1, 0), linewidth = 1.5, c = "black", ls = "--")
kw = dict(prop = "sizes", num = 5, color = "C0", func = lambda s: s/350)
ax.legend(*sc.legend_elements(**kw), title = "cos2", title_fontsize = 12,␣
 ↪bbox_to_anchor = (1.03, 0.66), frameon = False)
ax.grid()
plt.show()
```



2.3 Exercice 3

[48]:
```
# (3.1) Représentation des individus en 3D (les axes représentant les trois␣
 ↪premières composantes)
fig = plt.figure(figsize = (17.5/2.54, 17.5/2.54))
ax = fig.add_subplot(projection = "3d")
```

```
ax.scatter(df_pca_decat_scores_cos2["PC1"], df_pca_decat_scores_cos2["PC2"],␣
↪df_pca_decat_scores_cos2["PC3"], marker = "o", s = 50, alpha = 0.5)
ax.set_xlabel("PC1 (32.7%)")
ax.set_ylabel("PC2 (17.4%)")
ax.set_zlabel("PC3 (14.0%)")
ax.view_init(30,30)
plt.show()
```



2.4 Exercice 4

[49]:
```
# (4.1) Affichage de la saturation des variables pour les cinq premières␣
↪composantes, en triant les variables par ordre décroissant des valeurs sur␣
↪la PC1
loadings_decat_df.sort_values("PC1", ascending = False)[["PC1", "PC2", "PC3",␣
↪"PC4", "PC5"]]
```

[49]:

	PC1	PC2	PC3	PC4	PC5
Saut en longueur	0.410152	-0.262079	0.153727	0.099010	-0.044323
Lancer du poids	0.344144	0.453947	-0.019724	0.185395	-0.134320

```
Saut en hauteur    0.316194  0.265776 -0.218943 -0.131897 -0.671218
Lancer du disque   0.305426  0.460024  0.036238 -0.252591  0.126678
Lancer du javelot  0.153198  0.240507 -0.328742  0.692855  0.368731
Saut à la perche   0.027831 -0.136841  0.583617  0.536495 -0.398737
1500m             -0.032107  0.359805  0.659874 -0.156696  0.185571
400m              -0.375716  0.432046  0.110918  0.028503  0.105970
110m haies        -0.412554  0.173591 -0.078156  0.282901 -0.198573
100m              -0.428296  0.141989 -0.155580 -0.036787 -0.365187
```

D'après cette table de saturation, les variables les plus saturées sur la PC1 sont d'une part le saut en longueur et en hauteur, le lancer du poids et du disque, et d'autre part le 100m, le 110m haies et le 400m. Cet axe pourrait donc être nommé "Sauts et lancers vs sprint". De la même manière, la PC2 représente le plus les lancers du poids et du disque ainsi que le 400m et le 1500m, et pourrait donc être nommée "Lancers et courses de demi-fond". Le label de la PC3 serait plutôt "Saut à la perche et 1500m", celui de la PC4 "Lancer du javelot et saut à la perche" et celui de la PC5 "Saut en hauteur et saut à la perche".

2.5 Exercice 5

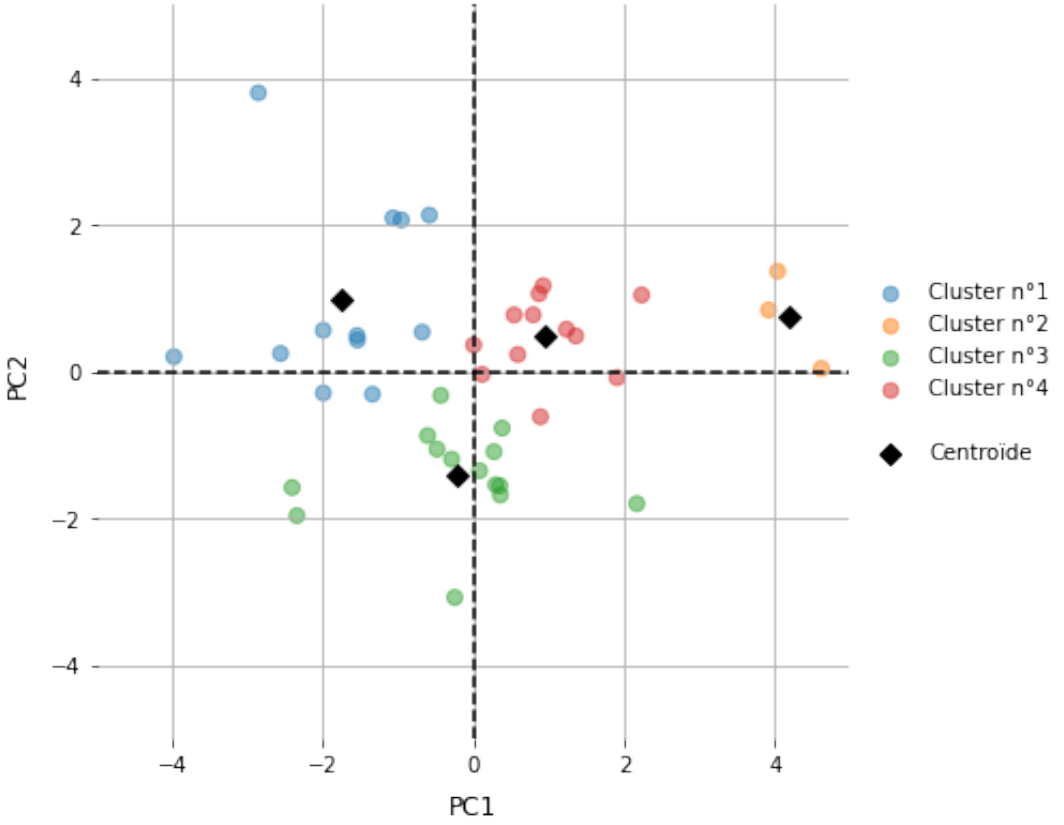
Les algorithmes de k-means et de CAH (classification ascendante hiérarchique) sont tous les deux des algorithmes de clustering liés à l'apprentissage non-supervisé. Le principal avantage du k-means par rapport à la CAH est le fait qu'il soit peut coûteux en temps de calcul, ce qui permet de l'appliquer sur des grands jeux de données. Il est particulièrement adapté pour classifier des données globulaires, i.e. agrégées en différents endroits dans un plan. En revanche, ses résultats sont difficilement reproductibles, du fait de la position initiale aléatoire des centroïdes. La CAH est quant à elle reproductible, et son autre avantage est de permettre le choix des clusters a posteriori (contrairement au k-means où le nombre de clusters doit être défini dès le départ).

2.6 Exercice 6

[50]:
```
# (6.1) Application du k-means pour classifier les athlètes dans quatre␣
↪clusters distincts
model_kmeans_decat = KMeans(n_clusters = 4)
model_kmeans_decat.fit(df_pca_decat_scores_cos2[["PC1", "PC2"]])
labels_kmeans_decat = model_kmeans_decat.
↪predict(df_pca_decat_scores_cos2[["PC1", "PC2"]])
centroids_kmeans_decat = model_kmeans_decat.cluster_centers_
df_pca_decat_scores_kmeans = df_pca_decat_scores_cos2.copy(deep = True)
df_pca_decat_scores_kmeans["Cluster"] = list(labels_kmeans_decat)
```

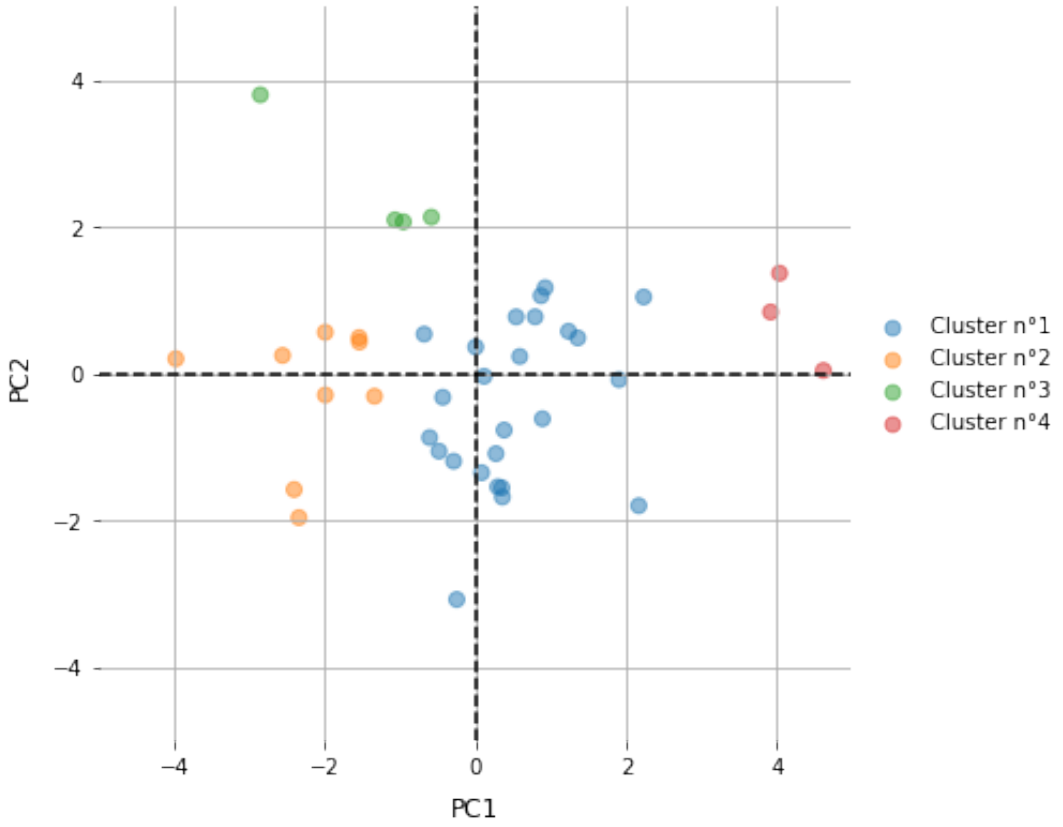
[51]:
```
# (6.2) Représentation des athlètes dans le plan de l'ACP en distinguant par␣
↪couleur les clusters obtenus par k-means
colors = ["C0", "C1", "C2", "C3"]
couleur_cluster_kmeans_decat = []
for i in df_pca_decat_scores_kmeans["Cluster"]:
    couleur_cluster_kmeans_decat.append(colors[i])
df_pca_decat_scores_kmeans["Couleur_cluster"] = couleur_cluster_kmeans_decat
legend = {}
fig, ax = plt.subplots(figsize = (16/2.54, 16/2.54))
```

```
ax.scatter(df_pca_decat_scores_kmeans["PC1"],␣
 ↪df_pca_decat_scores_kmeans["PC2"], c =␣
 ↪df_pca_decat_scores_kmeans["Couleur_cluster"], marker = "o", alpha = 0.5, s␣
 ↪= 50)
for i in df_pca_decat_scores_kmeans["Cluster"].unique():
    legend["Cluster n°" + str(i+1)] = ax.scatter(x = 100, y = 100, c =␣
 ↪colors[i], s = 50, alpha = 0.5, marker = "o", label = "Cluster n°" +␣
 ↪str(i+1))
centroids = ax.scatter(centroids_kmeans_decat[:,0], centroids_kmeans_decat[:
 ↪,1], marker = "D", s = 50, c = "black", label = "Centroïde")
ax.set_xlabel("PC1", fontsize = 12, labelpad = 8)
ax.set_ylabel("PC2", fontsize = 12, labelpad = 8)
ax.set_xlim(-5,5)
ax.set_ylim(-5,5)
empty_legend = ax.scatter(x = 100, y = 100, c = "white", label = " ")
ax.legend(handles = [legend["Cluster n°1"], legend["Cluster n°2"],␣
 ↪legend["Cluster n°3"], legend["Cluster n°4"], empty_legend, centroids],␣
 ↪bbox_to_anchor = (1, 0.65), frameon = False)
ax.spines[:].set_visible(False)
ax.axline(xy1 = (0, 0), xy2 = (0, 1), linewidth = 1.5, c = "black", ls = "--")
ax.axline(xy1 = (0, 0), xy2 = (1, 0), linewidth = 1.5, c = "black", ls = "--")
ax.grid()
plt.show()
```



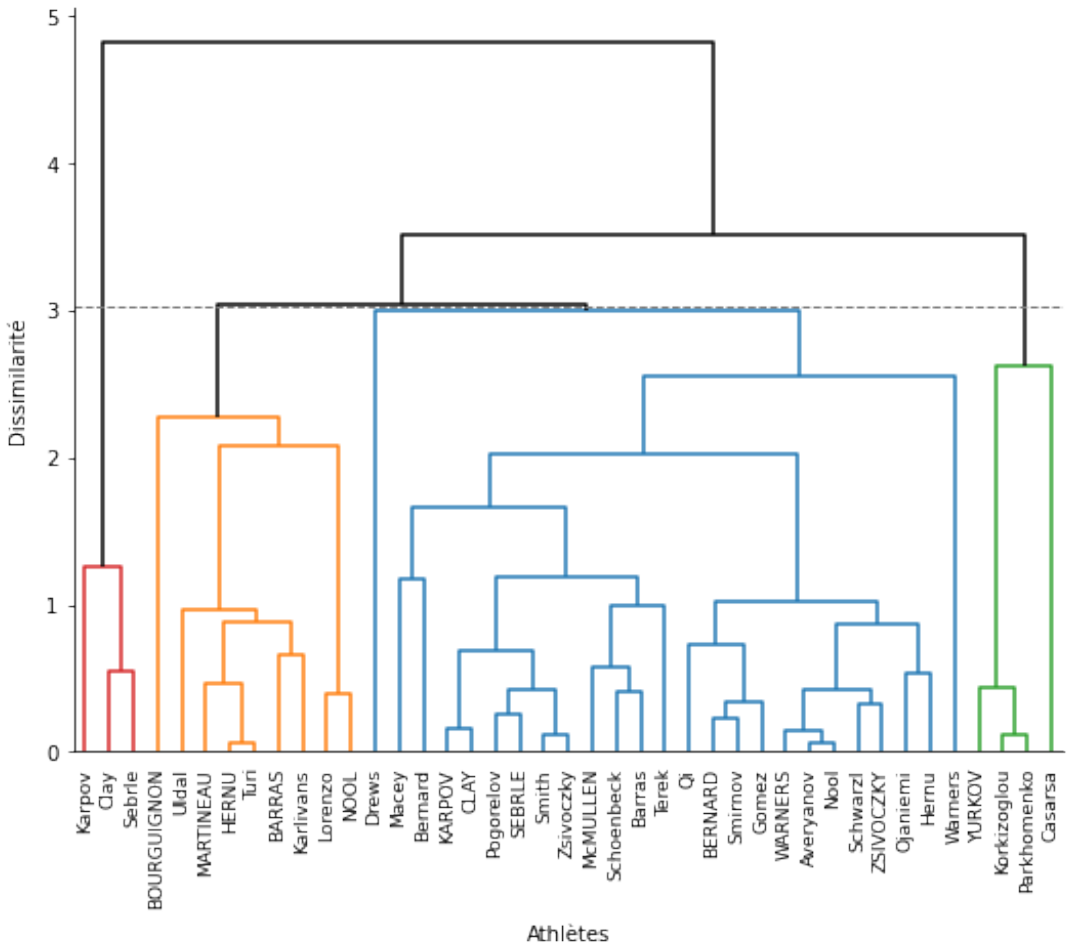
[52]:
```
# (6.3) Application de la CAH pour classifier les athlètes (utilisation de la␣
↪méthode "average")
model_CAH_decat = AgglomerativeClustering(n_clusters = 4, affinity =␣
↪"euclidean", linkage = "average")
clusters_CAH_decat = model_CAH_decat.fit(df_pca_decat_scores_cos2[["PC1",␣
↪"PC2"]])
df_pca_decat_scores_CAH = df_pca_decat_scores_cos2.copy(deep = True)
df_pca_decat_scores_CAH["Cluster"] = list(clusters_CAH_decat.labels_)
```

[53]:
```
# (6.4) Représentation des athlètes dans le plan de l'ACP en distinguant par␣
↪couleur les clusters obtenus par CAH
colors = ["C0", "C1", "C2", "C3"]
couleur_cluster_CAH_decat = []
for i in df_pca_decat_scores_CAH["Cluster"]:
    couleur_cluster_CAH_decat.append(colors[i])
df_pca_decat_scores_CAH["Couleur_cluster"] = couleur_cluster_CAH_decat
legend = {}
fig, ax = plt.subplots(figsize = (16/2.54, 16/2.54))
ax.scatter(df_pca_decat_scores_CAH["PC1"], df_pca_decat_scores_CAH["PC2"], c␣
↪= df_pca_decat_scores_CAH["Couleur_cluster"], marker = "o", alpha = 0.5, s␣
↪= 50)
for i in df_pca_decat_scores_CAH["Cluster"].unique():
    legend["Cluster n°" + str(i+1)] = ax.scatter(x = 100, y = 100, c =␣
↪colors[i], s = 50, alpha = 0.5, marker = "o", label = "Cluster n°" +␣
↪str(i+1))
ax.set_xlabel("PC1", fontsize = 12, labelpad = 8)
ax.set_ylabel("PC2", fontsize = 12, labelpad = 8)
ax.set_xlim(-5,5)
ax.set_ylim(-5,5)
ax.legend(handles = [legend["Cluster n°1"], legend["Cluster n°2"],␣
↪legend["Cluster n°3"], legend["Cluster n°4"]], bbox_to_anchor = (1, 0.605),␣
↪frameon = False)
ax.spines[:].set_visible(False)
ax.axline(xy1 = (0, 0), xy2 = (0, 1), linewidth = 1.5, c = "black", ls = "--")
ax.axline(xy1 = (0, 0), xy2 = (1, 0), linewidth = 1.5, c = "black", ls = "--")
ax.grid()
plt.show()
```



2.7 Exercice 7

[54]:
```
# (7.1) Création d'un dendrogramme en 2D pour la CAH (utilisation de la␣
 ↪méthode "average" pour le calcul des distances entre individus)
dendrogram_decat = sch.linkage(df_pca_decat_scores_cos2[["PC1", "PC2"]],␣
 ↪method = "average", optimal_ordering = True)
fig, ax = plt.subplots(figsize = (22/2.54, 17/2.54))
sch.set_link_color_palette(["C3", "C1", "C0", "C2"])
sch.dendrogram(dendrogram_decat, leaf_rotation = 90, leaf_font_size = 9,␣
 ↪labels = list(df_pca_decat_scores_cos2["Athlets"]), color_threshold = 3.02,␣
 ↪above_threshold_color = "black")
ax.set_xlabel("Athlètes", labelpad = 8)
ax.set_ylabel("Dissimilarité", labelpad = 8)
ax.axhline(y = 3.02, c = "grey", lw = 1, linestyle = "dashed")
ax.spines[["right", "top"]].set_visible(False)
plt.show()
```



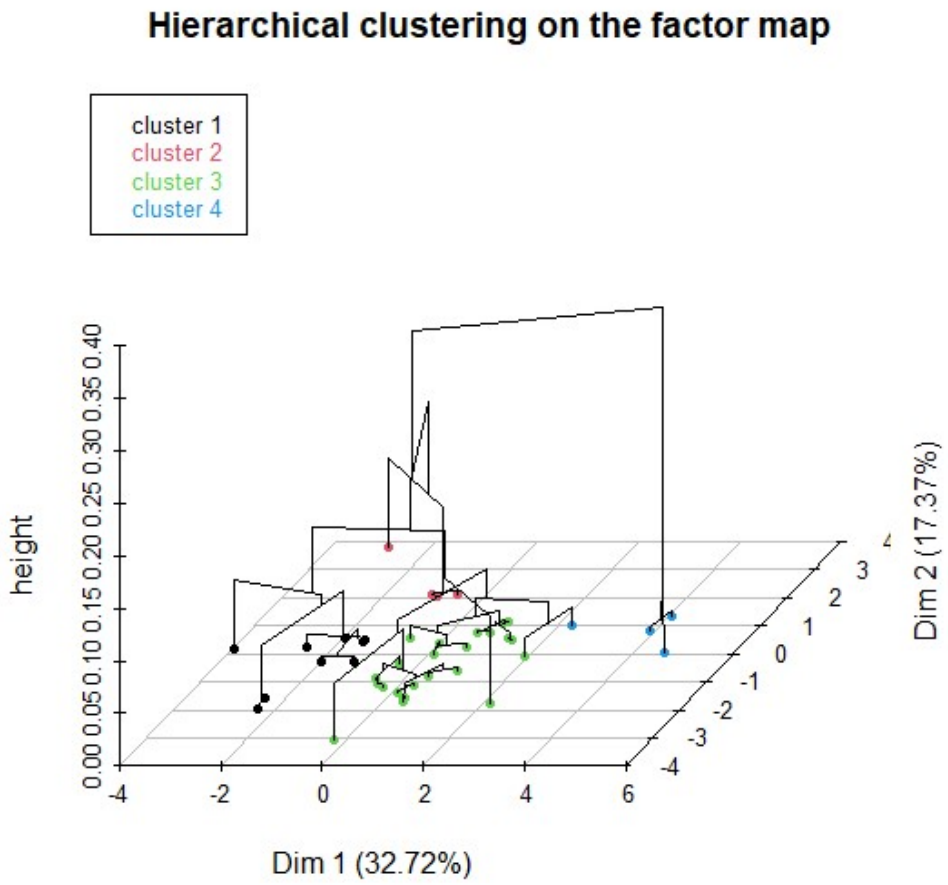
Le nombre de clusters choisi a été défini en faisant en sorte que tous les individus se retrouvent dans un groupe et de façon à différencier les individus selon leur position sur le plan de l'ACP. Par exemple, le cluster n°4 (en rouge) regroupe les trois athlètes ayant un score positif élevé sur la PC1 et faible sur la PC2. D'après la nomination des axes dans l'exercice 4, ces athlètes seraient très bons en saut en longueur et en hauteur et en lancer du poids et du disque, et moins bons en 100m, 110m haies et 400m. A l'inverse, le cluster n°2 (en orange) rassemble les athlètes qui ont un score opposé sur la PC1, et qui sont donc a priori meilleurs en sprint qu'en sauts et lancers. Le cluster n°3 correspond aux athlètes performants sur la PC2, c'est-à-dire en 400m et 1500m et en lancer du poids et du disque. Enfin, le cluster n°1 (en bleu) regroupe tous les individus qui se trouvent à peu près au centre des deux axes, et qui se différencient peut-être davantage dans les disciplines peu représentées par ces deux axes, à savoir le saut à la perche et le javelot.

2.8 Exercice 8

Le code suivant a été utilisé dans R pour générer un dendrogramme en 3D dans le plan de l'ACP :

```
library("factoextra")
library("FactoMineR")

df_decat <- read.table("decathlon.txt", header = TRUE)
```

```
res.pca_2PC <- PCA(df_decat[2:11], ncp = 2, graph = FALSE)
HCPC_res_ind_PC1_PC2 = HCPC(res.pca_2PC, nb.clust = 4, graph = FALSE,
   method = "average")
plot(HCPC_res_ind_PC1_PC2, choice = "3D.map", ind.names = FALSE, angle = 60)
```



Remarque : malgré l'utilisation de la même méthode ("average") pour réaliser la CAH dans Python et R, le résultat est légèrement différent entre les deux langages, un des individus du cluster n°1 dans Python (cluster n°3 dans le précédent graphe) passant dans le cluster n°4 dans R.

3 Projet 3 : Dating et analyse des correspondances multiples

3.1 Exercice 1

Une analyse factorielle des correspondances (AFC) est mobilisée lorsque l'on souhaite étudier le lien entre deux variables qualitatives. Une analyse des correspondances multiples (ACM) est utilisée pour étudier les liens entre trois variables qualitatives ou plus.

3.2 Exercice 2

[55]:
```
# (2.1) Import du jeu de données fictif "users.db.csv"
users_orig = pd.read_csv(r"C:\Users\quent\Documents\DU Data Analyst 2022\UE n°5 - Réduction de dimensionalité et algorithmes de clustering\Jeux de données\users.db.csv", low_memory = False, encoding = "latin-1")
users = users_orig.copy(deep = True)
users.head()
```

[55]:

	userid	date.crea	score	n.matches	n.updates.photo	n.photos	\
0	1	2011-09-17	1.495834	11	5	6	
1	2	2017-01-17	8.946863	56	2	6	
2	3	2019-05-14	2.496199	13	3	4	
3	4	2015-11-27	2.823579	32	5	2	
4	5	2014-11-28	2.117433	21	1	4	

	last.connex	last.up.photo	last.pr.update	gender	sent.ana	length.prof	\
0	2011-10-07	2011-10-02	NaN	1	6.490446	0.000000	
1	2017-01-31	2017-02-03	NaN	1	4.589125	20.722862	
2	2019-06-17	2019-06-19	NaN	1	6.473182	31.399277	
3	2016-01-15	2015-12-09	NaN	0	5.368982	0.000000	
4	2015-01-15	2015-01-02	NaN	0	5.573949	38.510225	

	voyage	laugh	photo.keke	photo.beach
0	0	0	0	0
1	0	0	0	1
2	0	0	0	1
3	0	0	0	1
4	0	1	0	0

[56]:
```
# (2.2) Inspection des variables
print(users.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   userid           3000 non-null   int64
 1   date.crea        3000 non-null   object
 2   score            3000 non-null   float64
 3   n.matches        3000 non-null   int64
 4   n.updates.photo  3000 non-null   int64
 5   n.photos         3000 non-null   int64
 6   last.connex      3000 non-null   object
```

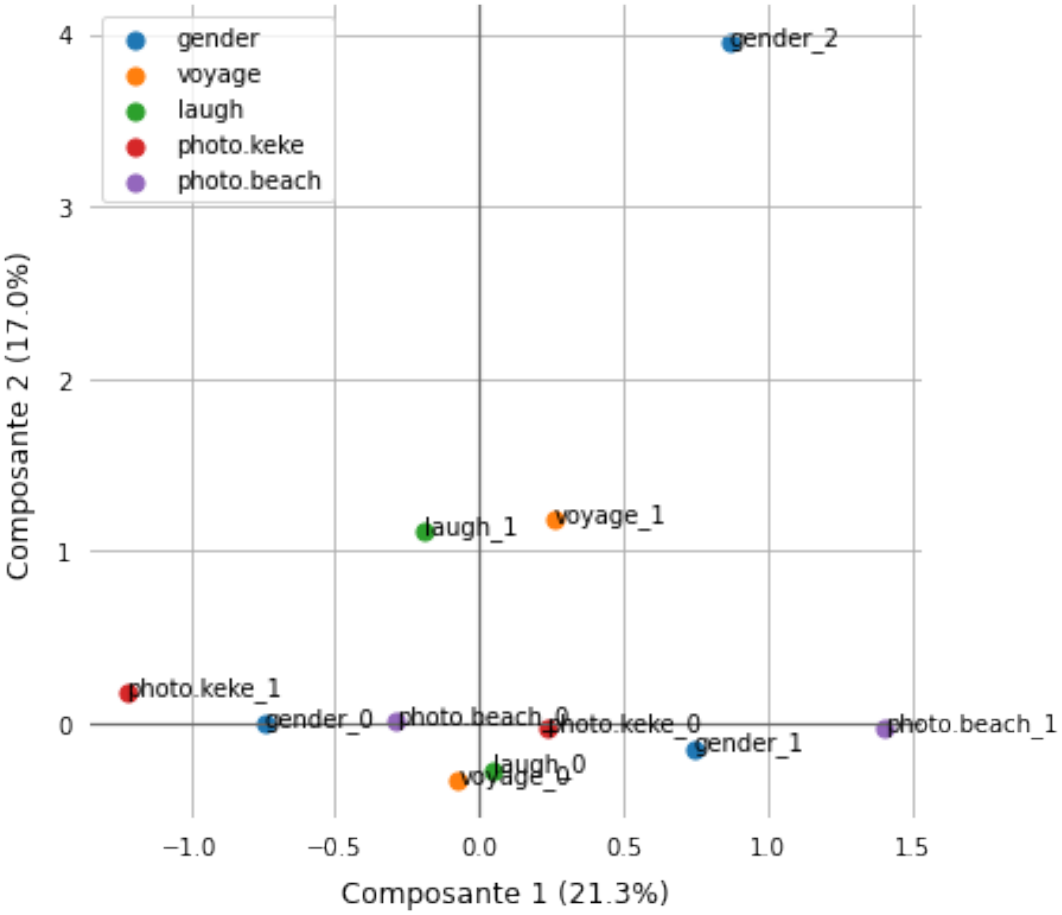
```
 7   last.up.photo    3000 non-null   object
 8   last.pr.update   0 non-null      float64
 9   gender           3000 non-null   int64
 10  sent.ana         3000 non-null   float64
 11  length.prof      3000 non-null   float64
 12  voyage           3000 non-null   int64
 13  laugh            3000 non-null   int64
 14  photo.keke       3000 non-null   int64
 15  photo.beach      3000 non-null   int64
dtypes: float64(4), int64(9), object(3)
memory usage: 375.1+ KB
None
```

[57]:
```
# (2.3) Sélection de cinq variables binaires puis conversion des données en␣
 ↪texte pour rendre ces variables qualitatives
users_qualit = users[["gender", "voyage", "laugh", "photo.keke", "photo.
 ↪beach"]]
for i in users_qualit.columns:
    users_qualit[i] = users_qualit[i].astype("str")
print(users_qualit.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   gender       3000 non-null   object
 1   voyage       3000 non-null   object
 2   laugh        3000 non-null   object
 3   photo.keke   3000 non-null   object
 4   photo.beach  3000 non-null   object
dtypes: object(5)
memory usage: 117.3+ KB
None
```

[58]:
```
# (2.4) Réalisation d'une ACM sur ces cinq variables qualitatives
mca = prince.MCA()
mca_users_qualit = mca.fit(users_qualit)
```

[59]:
```
# (2.5) Représentation des variables sur les deux premières composantes de␣
 ↪l'ACM
mca_users_qualit.plot_coordinates(X = users_qualit, figsize = (16/2.54, 16/2.
 ↪54), show_row_points = False, show_column_points = True, column_points_size␣
 ↪= 50, show_column_labels = True)
plt.title("")
plt.xlabel("Composante 1 (21.3%)", labelpad = 8, fontsize = 12)
plt.ylabel("Composante 2 (17.0%)", labelpad = 8, fontsize = 12)
plt.gca().spines[:].set_visible(False)
plt.show()
```



3.3 Exercice 3

La variance représentée dans le plan de l'ACM est de 38.3%. Cette variance est particulièrement faible : elle signifie que plus de la moitié de la variance du jeu de données n'est pas expliquée par les deux premiers axes.

3.4 Exercice 4

Malgré cette faible variance, il est possible de distinguer dans le plan plusieurs liens entre les variables. Par exemple, des différences générales de comportement sont observées selon le genre : les hommes tendent à afficher des photos d'eux-mêmes en maillot de bain, dans un ascenseur ou avec des lunettes de soleil (variable "photo.keke"), tandis que les femmes montrent plutôt des photos prises à la plage (variable "photo.beach"). Aucun lien n'est observé entre les personnes de genre "autre" et le type de photos affichées. Par ailleurs, la proximité des variables "laugh" et "voyage" dans le plan montre que les personnes qui écrivent le mot-clé "voyage" dans leur profil mentionnent également le mot "rire", quel que soit le genre.

3.5 Exercice 5

Le code suivant a été utilisé dans R pour récupérer les données de saturation des variables sur les deux premiers axes de l'ACM (ces données ont ensuite été stockées dans le tableau "ACM_var.csv") :

```
library("factoextra")
library("FactoMineR")

df_users <- read.table("users.db.txt", header = TRUE)
df_users_qualit <- df_users[c(10,13,14,15,16)]
df_users_qualit$gender <- as.factor(df_users_qualit$gender)
df_users_qualit$voyage <- as.factor(df_users_qualit$voyage)
df_users_qualit$laugh <- as.factor(df_users_qualit$laugh)
df_users_qualit$photo.keke <- as.factor(df_users_qualit$photo.keke)
df_users_qualit$photo.beach <- as.factor(df_users_qualit$photo.beach)
res.mca <- MCA(df_users_qualit, graph = FALSE)
res.mca$var$coord
```

[60]:
```
# (5.1) Représentation de la saturation des variables sur les deux premiers␣
↪axes de l'ACM
users_qualit_saturation = pd.read_csv(r"C:\Users\quent\Documents\DU Data␣
↪Analyst 2022\UE n°5 - Réduction de dimensionalité et algorithmes de␣
↪clustering\Jeux de données\ACM_var.csv", low_memory = False, encoding =␣
↪"latin-1")
users_qualit_saturation = users_qualit_saturation.set_index("Variable")
users_qualit_saturation[["Dim1", "Dim2"]]
```

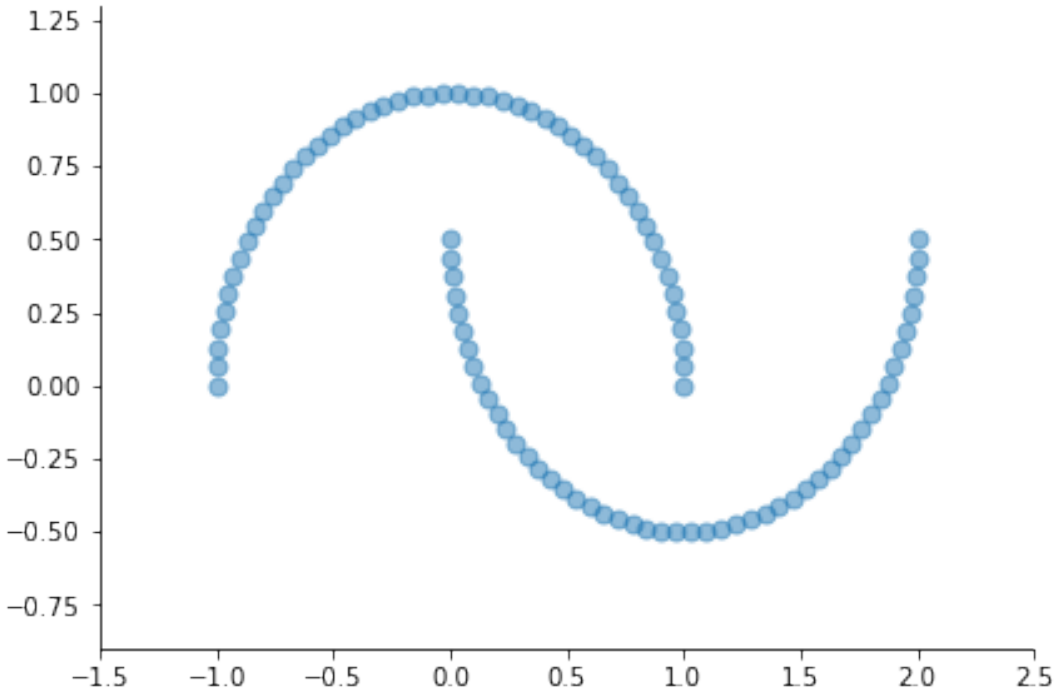
[60]:

Variable	Dim1	Dim2
gender_0	-0.741773	-0.008877
gender_1	0.744985	-0.150586
gender_2	0.866998	3.953923
voyage_0	-0.073505	-0.336529
voyage_1	0.258096	1.181647
laugh_0	0.046401	-0.271454
laugh_1	-0.190743	1.115876
photo.keke_0	0.233292	-0.034335
photo.keke_1	-1.221754	0.179810
photo.beach_0	-0.291215	0.006470
photo.beach_1	1.405181	-0.031221

4 Projet 4 : Classification non supervisée par DBSCAN

4.1 Exercice 1

[61]:
```
# (1.1) Production et représentation d'un jeu de données aléatoire en forme␣
↪de lunes se faisant face
noisy_moons = pd.DataFrame(datasets.make_moons(n_samples = 100, shuffle =␣
↪True, noise = False, random_state = True)[0])
fig, ax = plt.subplots(figsize = (17/2.54, 12/2.54))
ax.scatter(noisy_moons[0], noisy_moons[1], marker = "o", alpha = 0.5, s = 50)
ax.set_xlim(-1.5,2.5)
ax.set_ylim(-0.9,1.3)
ax.spines[["right", "top"]].set_visible(False)
plt.show()
```



4.2 Exercice 2

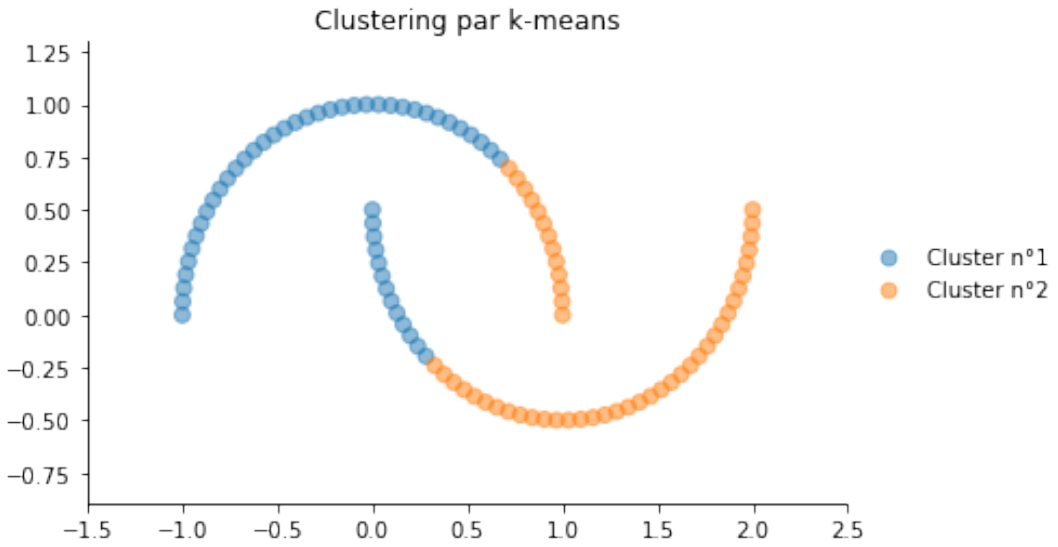
[62]:
```
# (2.1) Application du k-means sur le jeu de données créé
model_kmeans_moons = KMeans(n_clusters = 2)
model_kmeans_moons.fit(noisy_moons)
labels_kmeans_moons = model_kmeans_moons.predict(noisy_moons)
centroids_kmeans_moons = model_kmeans_moons.cluster_centers_
noisy_moons_kmeans = noisy_moons.copy(deep = True)
noisy_moons_kmeans["Cluster"] = list(labels_kmeans_moons)
```

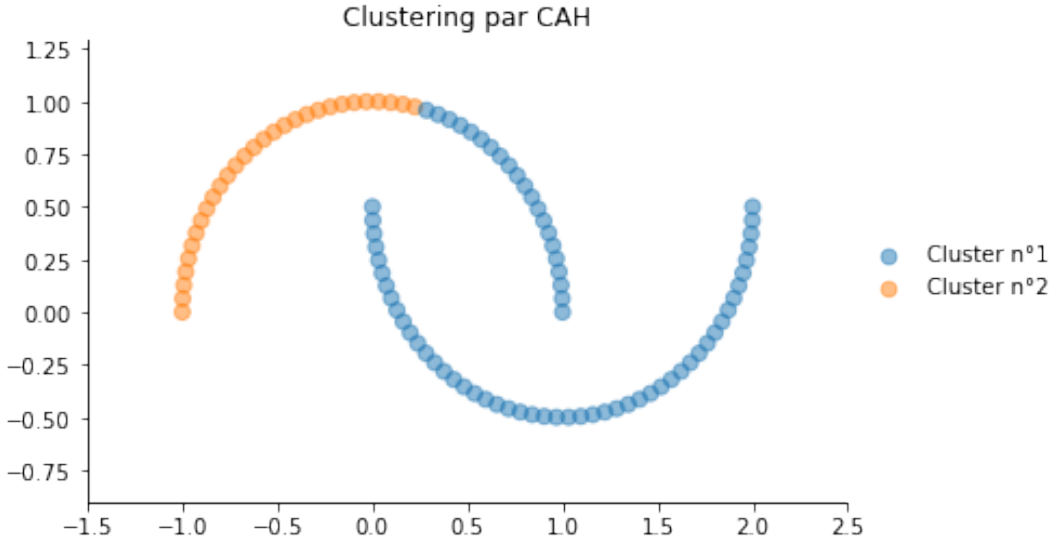
[63]:
```
# (2.2) Application de la CAH
model_CAH_moons = AgglomerativeClustering(n_clusters = 2, affinity =␣
↪"euclidean", linkage = "average")
```

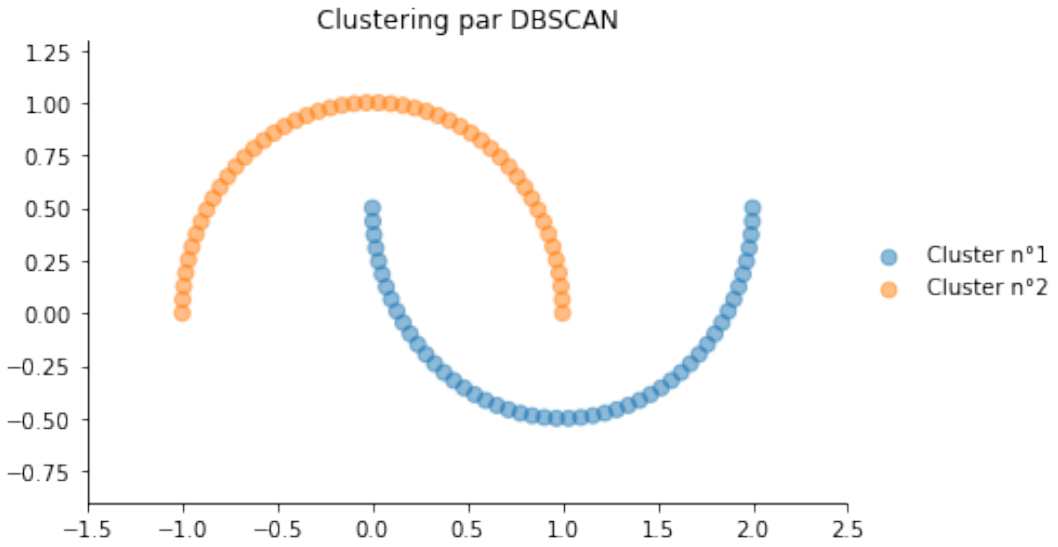
```
clusters_CAH_moons = model_CAH_moons.fit(noisy_moons)
noisy_moons_CAH = noisy_moons.copy(deep = True)
noisy_moons_CAH["Cluster"] = list(clusters_CAH_moons.labels_)
```

[64]:
```
# (2.3) Application du DBSCAN
model_DBSCAN_moons = DBSCAN().fit(noisy_moons)
noisy_moons_DBSCAN = noisy_moons.copy(deep = True)
noisy_moons_DBSCAN["Cluster"] = list(model_DBSCAN_moons.labels_)
```

[65]:
```
# (2.4) Représentation en couleur des clusters définis par ces trois méthodes␣
 ↪de clustering (un graphique pour chaque algorithme)
colors = ["C0", "C1"]
kmeans_CAH_DBSCAN = {"k-means" : noisy_moons_kmeans, "CAH" : noisy_moons_CAH,␣
 ↪"DBSCAN" : noisy_moons_DBSCAN}
for i in ["k-means", "CAH", "DBSCAN"]:
    couleur_cluster = []
    for j in kmeans_CAH_DBSCAN[i]["Cluster"]:
        couleur_cluster.append(colors[j])
    kmeans_CAH_DBSCAN[i]["Couleur_cluster"] = couleur_cluster
fig, ax = plt.subplots(nrows = 3, ncols = 1, figsize = (16/2.54, 36/2.54))
row = 0
for i in ["k-means", "CAH", "DBSCAN"]:
    ax[row].scatter(kmeans_CAH_DBSCAN[i][0], kmeans_CAH_DBSCAN[i][1], c =␣
 ↪kmeans_CAH_DBSCAN[i]["Couleur_cluster"], marker = "o", alpha = 0.5, s = 50)
    legend = {}
    for j in kmeans_CAH_DBSCAN[i]["Cluster"].unique():
        legend["Cluster n°" + str(j+1)] = ax[row].scatter(x = 100, y = 100, c␣
 ↪= colors[j], s = 50, alpha = 0.5, marker = "o", label = "Cluster n°" +␣
 ↪str(j+1))
    ax[row].set_title("Clustering par " + i)
    ax[row].set_xlim(-1.5,2.5)
    ax[row].set_ylim(-0.9,1.3)
    ax[row].legend(handles = [legend["Cluster n°1"], legend["Cluster n°2"]],␣
 ↪bbox_to_anchor = (1, 0.6), frameon = False)
    ax[row].spines[["right", "top"]].set_visible(False)
    row = row + 1
plt.subplots_adjust(hspace = 0.3)
plt.show()
```


Clustering par k-means
Cluster n°1
Cluster n°2


Clustering par CAH
Cluster n°1
Cluster n°2


Clustering par DBSCAN
Cluster n°1
Cluster n°2