Analyse exploratoire multidimensionnelle des données

Application sous Python avec scientisttools 0.1.4

Duvérier DJIFACK ZEBAZE

Table des matières

1

Analyse en Composantes Principales

Sommaire

Ce chapitre a pour objectif de présenter rapidement les principales fonctionnalités offertes par le package « scientisttools » pour réaliser une Analyse en Composantes Principales.

1.1 Présentation des données

On utilise ici l'exemple du tableau de données décathlon qui contient les performances réalisées par des athlètes lors de deux compétitions.

Vous pouvez charger le jeu de données http ://factominer.free.fr/factomethods/datasets/decathlon

```
# Importation des données
import pandas as pd
url = "http://factominer.free.fr/factomethods/datasets/decathlon.txt"
decathlon = pd.read_table(url,header=0)
decathlon.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 41 entries, SEBRLE to Casarsa
## Data columns (total 13 columns):
##  #   Column       Non-Null Count  Dtype
## ---  ------       --------------  -----
##  0   100m         41 non-null     float64
##  1   Long.jump    41 non-null     float64
##  2   Shot.put     41 non-null     float64
##  3   High.jump    41 non-null     float64
```

```
## 4   400m          41 non-null    float64
## 5   110m.hurdle   41 non-null    float64
## 6   Discus        41 non-null    float64
## 7   Pole.vault    41 non-null    float64
## 8   Javeline      41 non-null    float64
## 9   1500m         41 non-null    float64
## 10  Rank          41 non-null    int64
## 11  Points        41 non-null    int64
## 12  Competition   41 non-null    object
## dtypes: float64(10), int64(2), object(1)
## memory usage: 4.5+ KB
```

Table 1.1 – Données Decathlon

	100m	Long.jump	Shot.put	High.jump	400m	110m.hurdle	Discus	Pole.vault	Javeline	1500m	Rank	Points	Competition
SEBRLE	11.04	7.58	14.83	2.07	49.81	14.69	43.75	5.02	63.19	291.70	1	8217	Decastar
CLAY	10.76	7.40	14.26	1.86	49.37	14.05	50.72	4.92	60.15	301.50	2	8122	Decastar
KARPOV	11.02	7.30	14.77	2.04	48.37	14.09	48.95	4.92	50.31	300.20	3	8099	Decastar
BERNARD	11.02	7.23	14.25	1.92	48.93	14.99	40.87	5.32	62.77	280.10	4	8067	Decastar
YURKOV	11.34	7.09	15.19	2.10	50.42	15.31	46.26	4.72	63.44	276.40	5	8036	Decastar
WARNERS	11.11	7.60	14.31	1.98	48.68	14.23	41.10	4.92	51.77	278.10	6	8030	Decastar
ZSIVOCZKY	11.13	7.30	13.48	2.01	48.62	14.17	45.67	4.42	55.37	268.00	7	8004	Decastar
McMULLEN	10.83	7.31	13.76	2.13	49.91	14.38	44.41	4.42	56.37	285.10	8	7995	Decastar
MARTINEAU	11.64	6.81	14.57	1.95	50.14	14.93	47.60	4.92	52.33	262.10	9	7802	Decastar
HERNU	11.37	7.56	14.41	1.86	51.10	15.06	44.99	4.82	57.19	285.10	10	7733	Decastar
BARRAS	11.33	6.97	14.09	1.95	49.48	14.48	42.10	4.72	55.40	282.00	11	7708	Decastar
NOOL	11.33	7.27	12.68	1.98	49.20	15.29	37.92	4.62	57.44	266.60	12	7651	Decastar
BOURGUIGNON	11.36	6.80	13.46	1.86	51.16	15.67	40.49	5.02	54.68	291.70	13	7313	Decastar
Sebrle	10.85	7.84	16.36	2.12	48.36	14.05	48.72	5.00	70.52	280.01	1	8893	OlympicG
Clay	10.44	7.96	15.23	2.06	49.19	14.13	50.11	4.90	69.71	282.00	2	8820	OlympicG
Karpov	10.50	7.81	15.93	2.09	46.81	13.97	51.65	4.60	55.54	278.11	3	8725	OlympicG
Macey	10.89	7.47	15.73	2.15	48.97	14.56	48.34	4.40	58.46	265.42	4	8414	OlympicG
Warners	10.62	7.74	14.48	1.97	47.97	14.01	43.73	4.90	55.39	278.05	5	8343	OlympicG
Zsivoczky	10.91	7.14	15.31	2.12	49.40	14.95	45.62	4.70	63.45	269.54	6	8287	OlympicG
Hernu	10.97	7.19	14.65	2.03	48.73	14.25	44.72	4.80	57.76	264.35	7	8237	OlympicG
Nool	10.80	7.53	14.26	1.88	48.81	14.80	42.05	5.40	61.33	276.33	8	8235	OlympicG
Bernard	10.69	7.48	14.80	2.12	49.13	14.17	44.75	4.40	55.27	276.31	9	8225	OlympicG
Schwarzl	10.98	7.49	14.01	1.94	49.76	14.25	42.43	5.10	56.32	273.56	10	8102	OlympicG
Pogorelov	10.95	7.31	15.10	2.06	50.79	14.21	44.60	5.00	53.45	287.63	11	8084	OlympicG
Schoenbeck	10.90	7.30	14.77	1.88	50.30	14.34	44.41	5.00	60.89	278.82	12	8077	OlympicG
Barras	11.14	6.99	14.91	1.94	49.41	14.37	44.83	4.60	64.55	267.09	13	8067	OlympicG
Smith	10.85	6.81	15.24	1.91	49.27	14.01	49.02	4.20	61.52	272.74	14	8023	OlympicG
Averyanov	10.55	7.34	14.44	1.94	49.72	14.39	39.88	4.80	54.51	271.02	15	8021	OlympicG
Ojaniemi	10.68	7.50	14.97	1.94	49.12	15.01	40.35	4.60	59.26	275.71	16	8006	OlympicG
Smirnov	10.89	7.07	13.88	1.94	49.11	14.77	42.47	4.70	60.88	263.31	17	7993	OlympicG
Qi	11.06	7.34	13.55	1.97	49.65	14.78	45.13	4.50	60.79	272.63	18	7934	OlympicG
Drews	10.87	7.38	13.07	1.88	48.51	14.01	40.11	5.00	51.53	274.21	19	7926	OlympicG
Parkhomenko	11.14	6.61	15.69	2.03	51.04	14.88	41.90	4.80	65.82	277.94	20	7918	OlympicG
Terek	10.92	6.94	15.15	1.94	49.56	15.12	45.62	5.30	50.62	290.36	21	7893	OlympicG
Gomez	11.08	7.26	14.57	1.85	48.61	14.41	40.95	4.40	60.71	269.70	22	7865	OlympicG
Turi	11.08	6.91	13.62	2.03	51.67	14.26	39.83	4.80	59.34	290.01	23	7708	OlympicG
Lorenzo	11.10	7.03	13.22	1.85	49.34	15.38	40.22	4.50	58.36	263.08	24	7592	OlympicG
Karlivans	11.33	7.26	13.30	1.97	50.54	14.98	43.34	4.50	52.92	278.67	25	7583	OlympicG
Korkizoglou	10.86	7.07	14.81	1.94	51.16	14.96	46.07	4.70	53.05	317.00	26	7573	OlympicG
Uldal	11.23	6.99	13.53	1.85	50.95	15.09	43.01	4.50	60.00	281.70	27	7495	OlympicG
Casarsa	11.36	6.68	14.92	1.94	53.20	15.39	48.66	4.40	58.62	296.12	28	7404	OlympicG

Le tableau de données contient 41 lignes et 13 colonnes (*cf.* Table 1.1). Les colonnes de 1 à 12 sont des variables continues : les dix premières colonnes correspondent aux performances des athlètes pour les dix épreuves du décathlon et les colonnes 11 et 12 correspondent respectivement au rang et au nombre de points obtenus. La dernière colonne est une variable qualitative correspondant au nom de la compétition (Jeux Olympiques de 2004 ou Décastar 2004).

Pour une meilleure manipulation des colonnes dans Python, nous remplaçons les points sur les colonnes par les tirets de 8.

```
# Renommer les colonnes
decathlon.columns = [x.replace(".","_") for x in decathlon.columns]
decathlon.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
```

```
## Index: 41 entries, SEBRLE to Casarsa
## Data columns (total 13 columns):
##  #   Column       Non-Null Count  Dtype
## ---  ------       --------------  -----
##  0   100m         41 non-null     float64
##  1   Long_jump    41 non-null     float64
##  2   Shot_put     41 non-null     float64
##  3   High_jump    41 non-null     float64
##  4   400m         41 non-null     float64
##  5   110m_hurdle  41 non-null     float64
##  6   Discus       41 non-null     float64
##  7   Pole_vault   41 non-null     float64
##  8   Javeline     41 non-null     float64
##  9   1500m        41 non-null     float64
##  10  Rank         41 non-null     int64
##  11  Points       41 non-null     int64
##  12  Competition  41 non-null     object
## dtypes: float64(10), int64(2), object(1)
## memory usage: 4.5+ KB
```

Il est important de s'assurer que l'importation a bien été effectuée, et notamment que les variables quantitatives sont bien considérées comme quantitatives et les variables qualitatives bien considérées comme qualitatives.

```
# Variable continues
import numpy as np
stat1 = decathlon.describe(include=np.number).T
```

Table 1.2 – Statistiques descriptives sur les variables continues

	count	mean	std	min	25%	50%	75%	max
100m	41	10.998	0.263	10.44	10.85	10.98	11.14	11.64
Long_jump	41	7.260	0.316	6.61	7.03	7.30	7.48	7.96
Shot_put	41	14.477	0.824	12.68	13.88	14.57	14.97	16.36
High_jump	41	1.977	0.089	1.85	1.92	1.95	2.04	2.15
400m	41	49.616	1.153	46.81	48.93	49.40	50.30	53.20
110m_hurdle	41	14.606	0.472	13.97	14.21	14.48	14.98	15.67
Discus	41	44.326	3.378	37.92	41.90	44.41	46.07	51.65
Pole_vault	41	4.762	0.278	4.20	4.50	4.80	4.92	5.40
Javeline	41	58.317	4.827	50.31	55.27	58.36	60.89	70.52
1500m	41	279.025	11.673	262.10	271.02	278.05	285.10	317.00
Rank	41	12.122	7.919	1.00	6.00	11.00	18.00	28.00
Points	41	8005.366	342.385	7313.00	7802.00	8021.00	8122.00	8893.00

```
stat2 = (decathlon.describe(include=["O"])
                  .reset_index()
                  .rename(columns={"index":"infos"}))
```

1.2 Objectifs

L'ACP permet de décrire un jeu de données, de le résumer, d'en réduire la dimensionnalité. L'ACP réalisée sur les individus du tableau de données répond à différentes questions :

Table 1.3 – Statistiques descriptives sur la variable catégorielle

infos	Competition
count	41
unique	2
top	OlympicG
freq	28

1. Etude des individus (i.e. des athlètes) : deux athlètes sont proches s'ils ont des résultats similaires. On s'intéresse à la variabilité entre individus. Y a-t-il des similarités entre les individus pour toutes les variables ? Peut-on établir des profils d'athlètes ? Peut-on opposer un groupe d'individus à un autre ?
2. Etude des variables (i.e. des performances) : on étudie les liaisons linéaires entre les variables. Les objectifs sont de résumer la matrice des corrélations et de chercher des variables synthétiques : peut-on résumer les performances des athlètes par un petit nombre de variables ?
3. Lien entre les deux études : peut-on caractériser des groups d'individus par des variables ?

1.3 ACP

On étudie les profils d'ahtlètes uniquement en fonction de leur performance. Les variables actives ne seront donc que celles qui concernent les dix épreuves du décathlon. Les autres variables ("*Rank*", "*Points*" et "*Competition*") n'appartiennent pas aux profils d'athlètes et utilisent une information déjà donnée par les autres variables (dans le cas de "*Rank*"et "*Points*") mais il est intéressant de les confronter aux composantes principales. Nous les utiliserons comme variables illustratives.

Dans ce tableau de données, les variables ne sont pas mesurées dans les mêmes unités. On doit les réduire de façon à donner la même influence à chacune.

On charge scientisttools

```
from scientisttools import PCA
```

1.3.1 Individus et variables actifs

On crée une instance de la classe PCA, en lui passant ici des étiquettes pour les lignes et les variables. Ces paramètres sont facultatifs ; en leur absence, le programme détermine automatiquement des étiquettes.

Le constructeur de la classe PCA possède un paramètre `normalize` qui indique si l'ACP est réalisée :

— à partir de données centrées et réduites -> PCA(normalize=True)
— à partir de données centrées mais non réduites -> PCA(normalize=False)

Par défaut, la valeur du paramètre `normalize` est fixée à `True`, car c'est le cas le plus courant.

Réalisez l'ACP sur tous les individus et seulement les variables actives (i.e. les dix premières) en tapant la ligne de code suivante :

```python
# Données actives
actif = decathlon[decathlon.columns[:10]]
# ACP sur les données actives uniquement - Instanciation du modèle
res_pca = PCA()
```

On estime le modèle en appliquant la méthode `.fit` de la classe PCA sur le jeu de données.

```python
# Entraînement du modèle
res_pca.fit(actif)
```

```
## PCA()
```

L'exécution de la méthode `res_pca.fit(actif)` provoque le calcul de plusieurs attributs parmi lesquels `res_pca.eig_`.

```python
print(res_pca.eig_)
```

```
##          eigenvalue  difference  proportion  cumulative
## Dim.1      3.271906    1.534775   32.719055   32.719055
## Dim.2      1.737131    0.332214   17.371310   50.090366
## Dim.3      1.404917    0.348066   14.049167   64.139532
## Dim.4      1.056850    0.372077   10.568504   74.708036
## Dim.5      0.684774    0.085505    6.847735   81.555771
## Dim.6      0.599269    0.148033    5.992687   87.548458
## Dim.7      0.451235    0.054359    4.512353   92.060811
## Dim.8      0.396877    0.182062    3.968766   96.029577
## Dim.9      0.214815    0.032587    2.148149   98.177725
## Dim.10     0.182227         NaN    1.822275  100.000000
```

L'attribut `res_pca.eig_` contient :

- en 1ère ligne : les valeurs propres en valeur absolue
- en 2ème ligne : les différences des valeurs propres
- en 3ème ligne : les valeurs propres en pourcentage de la variance totale (proportions)
- en 4ème ligne : les valeurs propres en pourcentage cumulé de la variance totale.

La fonction `get_eig` retourne les valeurs propres sous forme de tableau de données.

```python
# Valeurs propres
from scientisttools import get_eig
print(get_eig(res_pca))
```

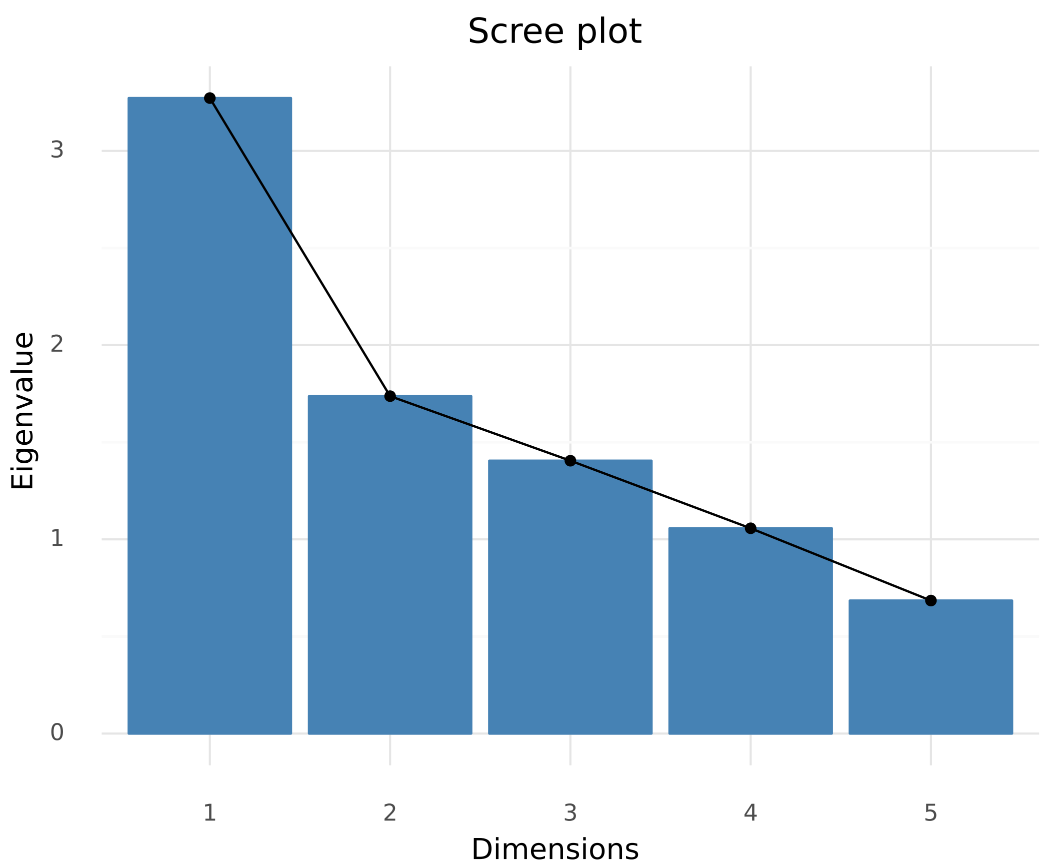
```
##          eigenvalue  difference  proportion  cumulative
## Dim.1      3.271906    1.534775   32.719055   32.719055
## Dim.2      1.737131    0.332214   17.371310   50.090366
## Dim.3      1.404917    0.348066   14.049167   64.139532
## Dim.4      1.056850    0.372077   10.568504   74.708036
## Dim.5      0.684774    0.085505    6.847735   81.555771
```

```
## Dim.6      0.599269     0.148033     5.992687    87.548458
## Dim.7      0.451235     0.054359     4.512353    92.060811
## Dim.8      0.396877     0.182062     3.968766    96.029577
## Dim.9      0.214815     0.032587     2.148149    98.177725
## Dim.10     0.182227          NaN     1.822275   100.000000
```

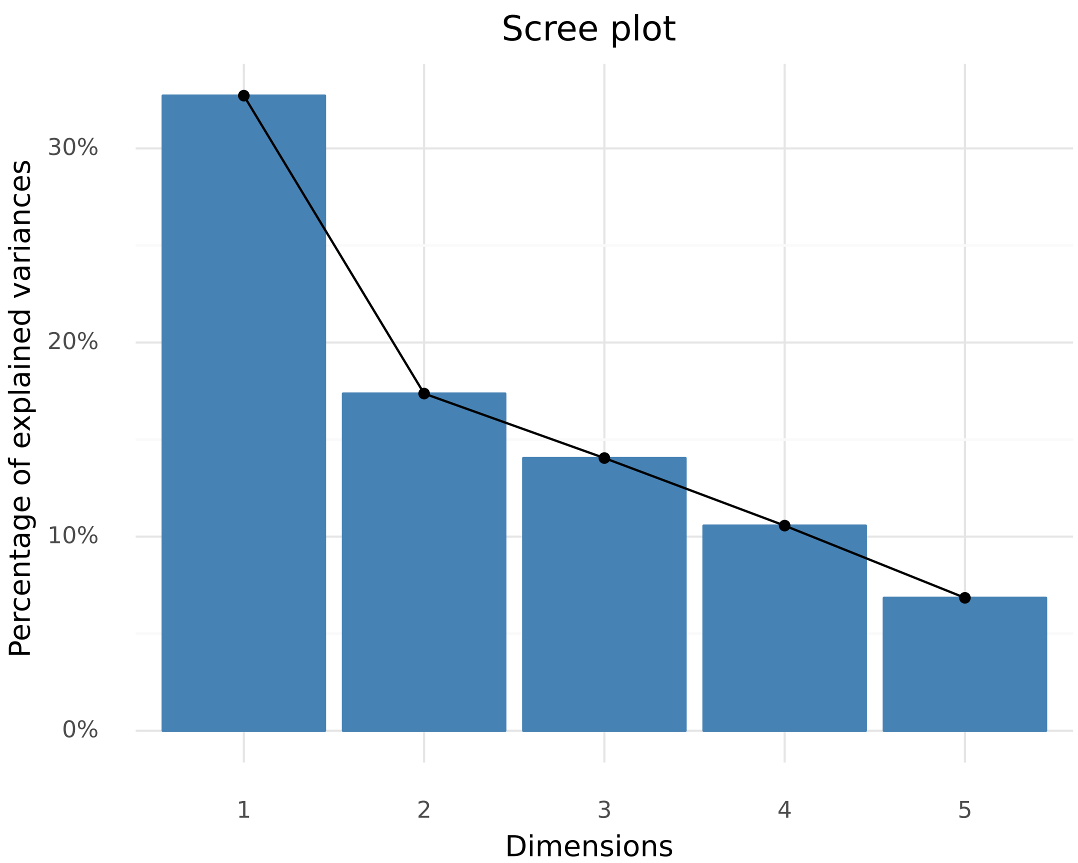
Les deux premières dimensions contiennent 50% de l'inertie totale (l'inertie est la variance totale du tableau de données, i.e. la trace de la matrice des corrélations).

Les valeurs propres peuvent être représentées graphiquement :

```
from scientisttools import fviz_screeplot
print(fviz_screeplot(res_pca,choice="eigenvalue"))
```




```
print(fviz_screeplot(res_pca,choice="proportion"))
```



On peut obtenir un résumé des principaux résultats en utilisant la fonction summaryPCA.

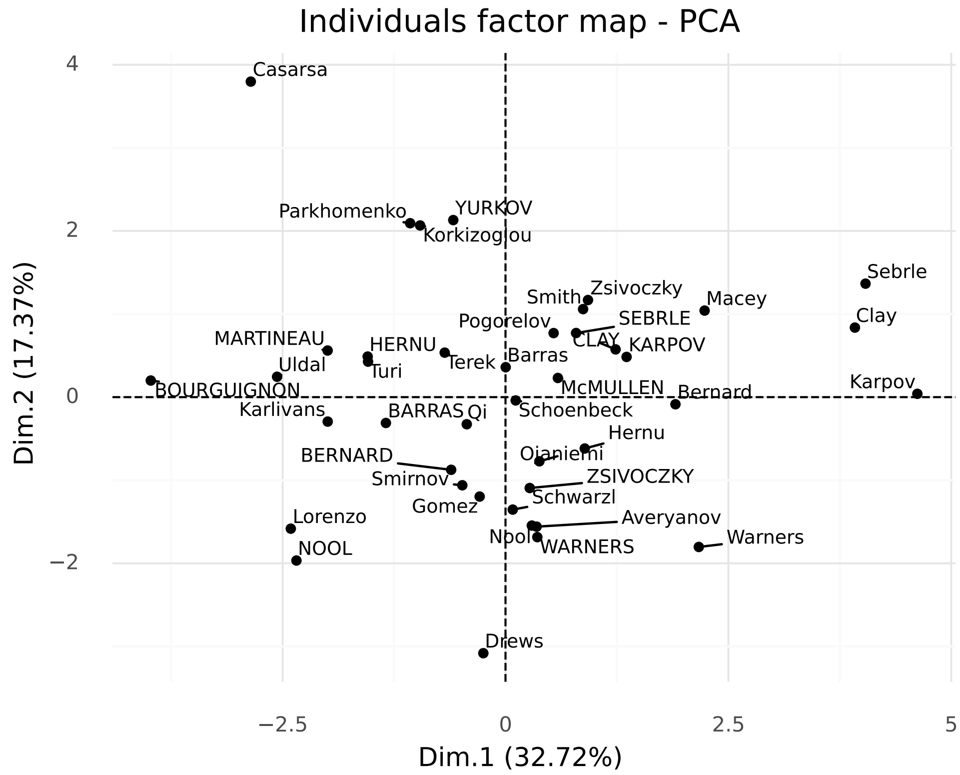
```
from scientisttools import summaryPCA
summaryPCA(res_pca)
```

```
##                        Principal Component Analysis - Results
##
## Importance of components
##                        Dim.1   Dim.2   Dim.3  ...   Dim.8   Dim.9   Dim.10
## Variance               3.272   1.737   1.405  ...   0.397   0.215    0.182
## Difference             1.535   0.332   0.348  ...   0.182   0.033      NaN
## % of var.             32.719  17.371  14.049  ...   3.969   2.148    1.822
## Cumulative % of var.  32.719  50.090  64.140  ...  96.030  98.178  100.000
##
## [4 rows x 10 columns]
##
## Individuals (the 10 first)
##
##               dist  weight  inertia  Dim.1  ...   cos2  Dim.3    ctr   cos2
## SEBRLE       2.369   0.024    0.137  0.792  ...  0.106  0.827  1.187  0.122
## CLAY         3.507   0.024    0.300  1.235  ...  0.027  2.141  7.960  0.373
## KARPOV       3.396   0.024    0.281  1.358  ...  0.020  1.956  6.644  0.332
## BERNARD      2.763   0.024    0.186 -0.610  ...  0.100  0.890  1.375  0.104
## YURKOV       3.018   0.024    0.222 -0.586  ...  0.499 -1.225  2.606  0.165
```

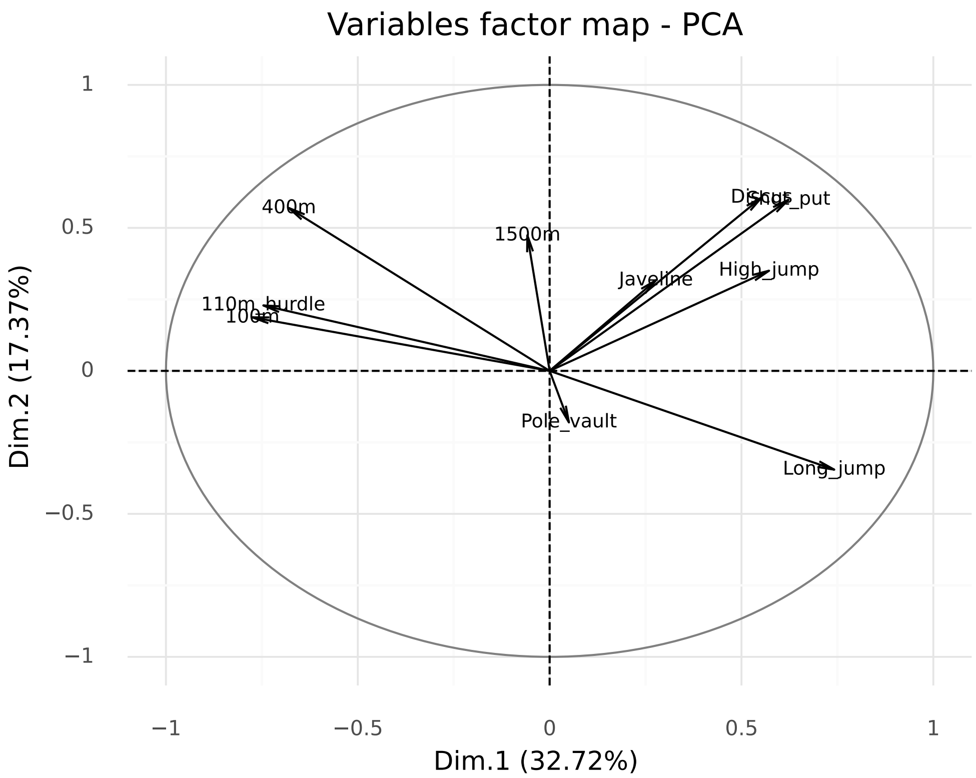
```
## WARNERS    2.428   0.024    0.144  0.357  ...  0.482  0.767  1.020  0.100
## ZSIVOCZKY  2.563   0.024    0.160  0.272  ...  0.182 -1.283  2.857  0.250
## McMULLEN   2.561   0.024    0.160  0.588  ...  0.008 -0.418  0.303  0.027
## MARTINEAU  3.742   0.024    0.342 -1.995  ...  0.022 -0.730  0.925  0.038
## HERNU      2.794   0.024    0.190 -1.546  ...  0.031  0.841  1.227  0.091
##
## [10 rows x 12 columns]
##
## Continuous variables
##
##               dist  weight  inertia  Dim.1  ...   cos2  Dim.3    ctr   cos2
## 100m           1.0     1.0      1.0 -0.775  ...  0.035 -0.184  2.420  0.034
## Long_jump      1.0     1.0      1.0  0.742  ...  0.119  0.182  2.363  0.033
## Shot_put       1.0     1.0      1.0  0.623  ...  0.358 -0.023  0.039  0.001
## High_jump      1.0     1.0      1.0  0.572  ...  0.123 -0.260  4.794  0.067
## 400m           1.0     1.0      1.0 -0.680  ...  0.324  0.131  1.230  0.017
## 110m_hurdle    1.0     1.0      1.0 -0.746  ...  0.052 -0.093  0.611  0.009
## Discus         1.0     1.0      1.0  0.552  ...  0.368  0.043  0.131  0.002
## Pole_vault     1.0     1.0      1.0  0.050  ...  0.033  0.692 34.061  0.479
## Javeline       1.0     1.0      1.0  0.277  ...  0.100 -0.390 10.807  0.152
## 1500m          1.0     1.0      1.0 -0.058  ...  0.225  0.782 43.543  0.612
##
## [10 rows x 12 columns]
```

1.3.1.1 Représentation graphique

```python
# Carte des individus
from scientisttools import fviz_pca_ind
print(fviz_pca_ind(res_pca,repel=True))
```




```python
# Cercle des corrélations
from scientisttools import fviz_pca_var
print(fviz_pca_var(res_pca))
```



La variable “X100m” est négativement corrélée à la variable “long_jump”. Quand un ahtlète réalise un temps faible au 100m, il peut sauter loin. Il faut faire attention ici qu’une petite valeur pour les variables “X100m”, “X400m”, “X110m_hurdle” et “X1500m” correspond à un score élevé : plus un athlète court rapidement, plus il gagne de points.

Le premier axe oppose les athlètes qui sont “bons partout” comme Karpov pendant les Jeux Olympiques à ceux qui sont “mauvais partout” comme Bourguignon pendant le Décastar. Cette dimension est particulièrement liée aux variables de vitesse et de saut en longueur qui constituent un groupe homogène.

Le deuxième axe oppose les athlètes qui sont forts (variables “Discus” et “Shot_put”) à ceux qui ne le sont pas. Les variables “Discus”, “Shot_put” et “High_jump” ne sont pas très corrélées aux variables “X100m”, “X400m”, “X110m_hurdle” et “Long_jump”. Cela signifie que force et vitesse ne sont pas très corrélées.

A l’issue de cette première approche, on peut diviser le premier plan factoriel en quatre parties : les athlètes rapides et puissants (comme Sebrle), les athlètes lents (comme Casarsa), les athlètes rapides mais faibles (comme Warners) et les ahtlètes ni forts ni rapides, relativement parlant (comme Lorenzo).

1.3.2 ACP avec les variables illustratives

Les variables illustratives n’influencent pas la construction des composantes principales de l’analyse. Elles aident à l’interprétation des dimensions de variabilité.

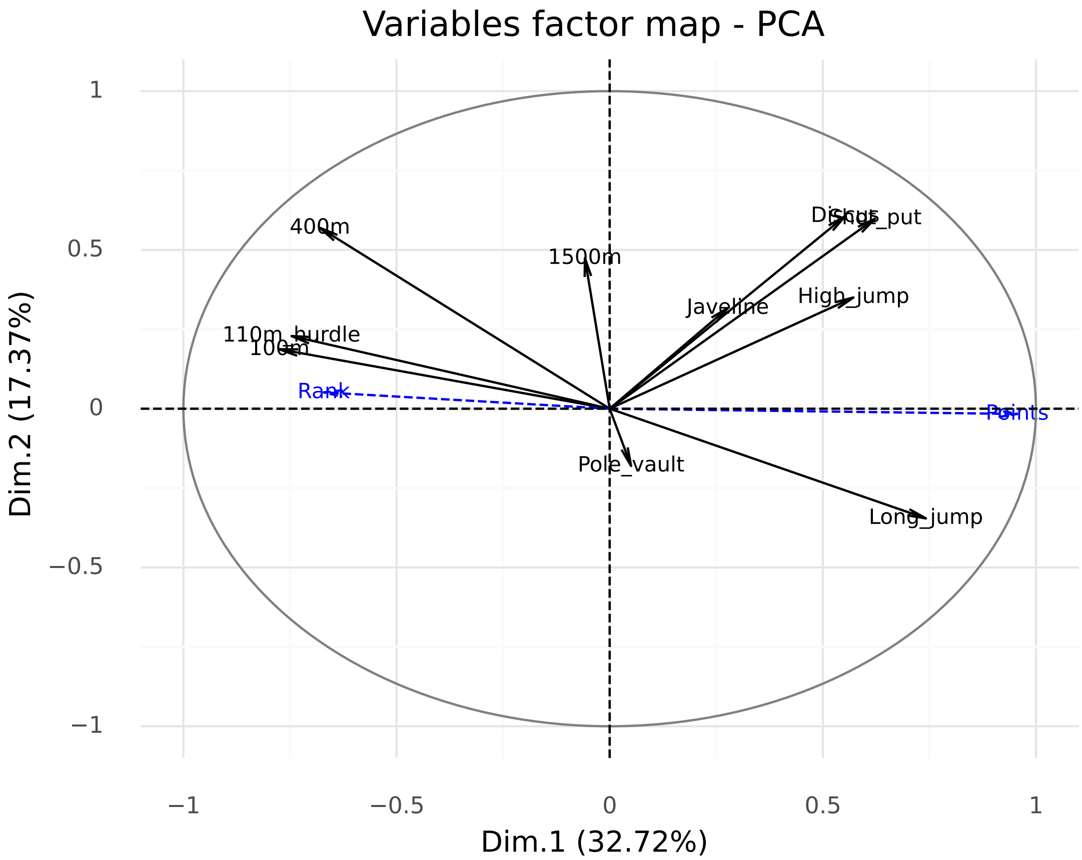
On peut ajouter deux types de variables : continues et qualitatives.

On ajoute les variables “Rank” and “Points” comme variables continues illustratives quantiatives et “Competition” comme variable qualitative illustrative. Tapez la ligne de code suivante :

```
res_pca = PCA(quanti_sup=[10,11],quali_sup=12)
res_pca.fit(decathlon)
```

```
## PCA(quali_sup=12, quanti_sup=[10, 11])
```

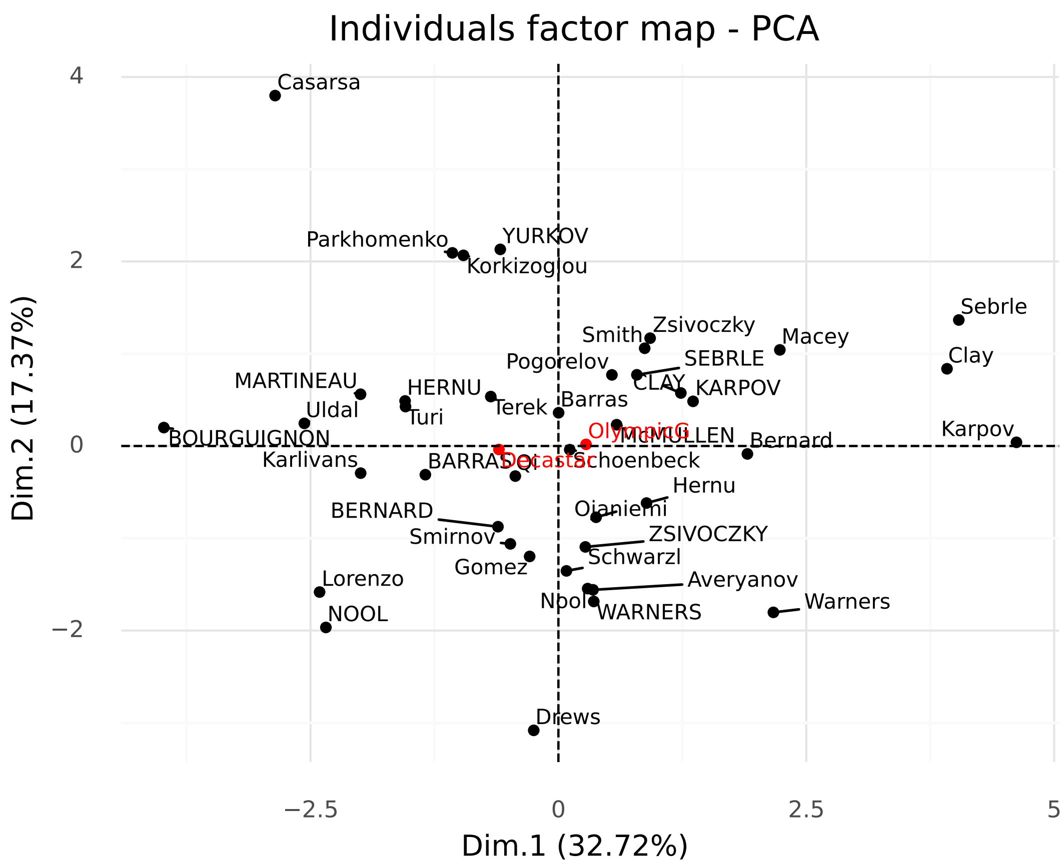
```
print(fviz_pca_var(res_pca))
```



Les gagnants du décathlon sont ceux qui marquent le plus de points (ou ceux dont le rang est faible). Les variables les plus liées au nombre de points sont les variables qui réfèrent à la vitesse ("X100m", "X110m_hurdle", "X400m") et au saut en longueur. Au contraire, "Pole-vault" et "X1500m" n'ont pas une grande influence sur le nombre de points. Les athlètes qui sont bons à ces deux épreuves ne sont pas favorisés.

On ajoute la variable "Competition" comme variable qualitative illustrative.

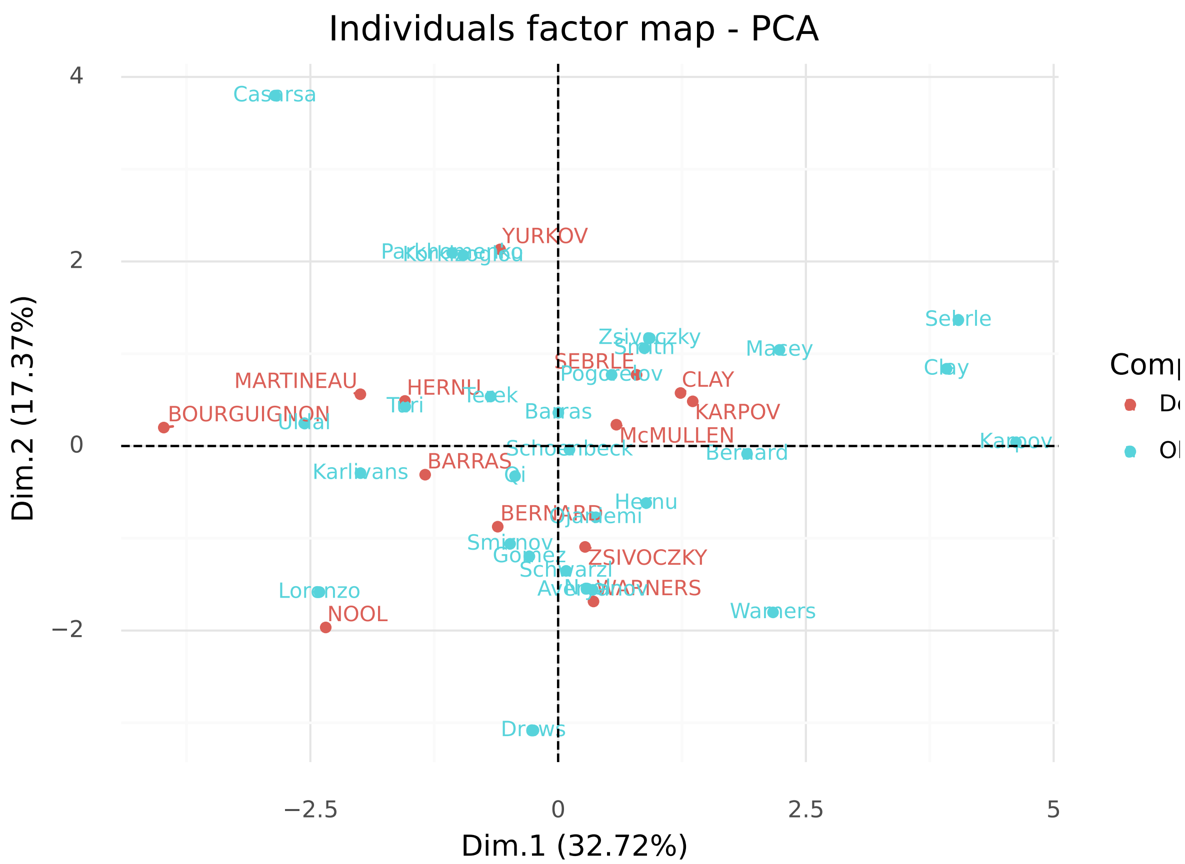
```
print(fviz_pca_ind(res_pca,repel=True))
```



Les centres de gravité des modalités de cette variable supplémentaire apparaissent sur le graphe des individus. Ils sont localisés au barycentre des individus qui les possèdent et représentent un individu moyen.

On peut également colorier les individus selon la couleur des centres de gravité des modalités :

```
print(fviz_pca_ind(res_pca,habillage="Competition",repel=True))
```



En regardant les points qui représentent “Decastar” et “Olympic Games”, on voit que “Olympic Games” a une coordonnée plus élevée sur le premier axe que “Decastar”. Ceci montre une évolution des performances des athlètes. Tous les athlètes qui ont participé aux deux compétitions ont obtenu des résultats légèrement meilleurs aux jeux Olympiques.

Cependant, il n’y a aucune différence entre les points “Decastar” et “Olympic Games” sur le deuxième axe. Cela signifie que les athlètes ont amélioré leurs performances mais n’ont pas changé de profil (à l’exception de Zsivoczky qui est passé de lent et fort pendant le Décastar à rapide et faible pendant les Jeux Olympiques).

Les points qui représentent un même individu vont dans le même direction. Par exemple, Sebrle a obtenu de bons résultats aux deux compétitions mais le point qui représente sa performance aux J.O. est plus extrême. Sebrle a obtenu plus de points pendant les J.O. que pendant le Décastar..

On peut envisager deux interprétations :

1. Les athlètes qui participent aux J.O. sont meilleurs que ceux qui participent au Décastar
2. Les athlètes font de leur mieux aux J.O. (plus motivés, plus entraînés)

```
summaryPCA(res_pca)
```

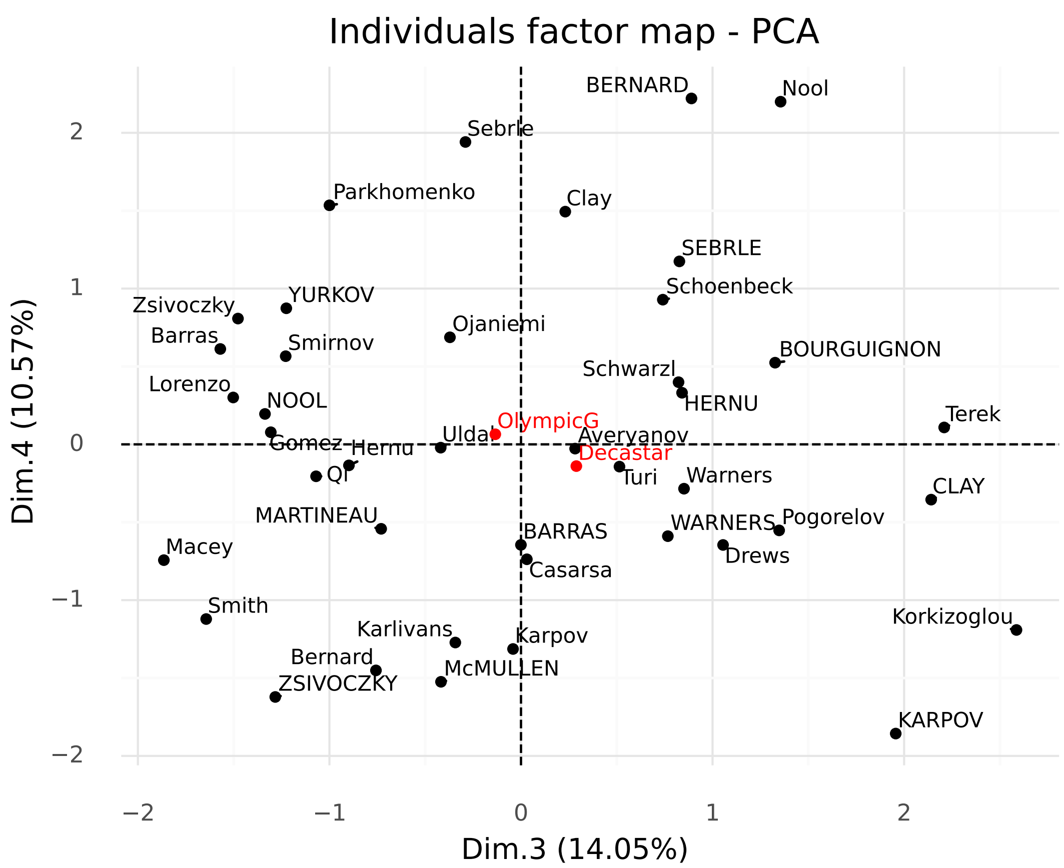
```
##                         Principal Component Analysis - Results
##
```

```
## Importance of components
##                        Dim.1   Dim.2   Dim.3  ...   Dim.8   Dim.9   Dim.10
## Variance               3.272   1.737   1.405  ...   0.397   0.215    0.182
## Difference             1.535   0.332   0.348  ...   0.182   0.033      NaN
## % of var.             32.719  17.371  14.049  ...   3.969   2.148    1.822
## Cumulative % of var.  32.719  50.090  64.140  ...  96.030  98.178  100.000
##
## [4 rows x 10 columns]
##
## Individuals (the 10 first)
##
##              dist  weight  inertia  Dim.1  ...   cos2  Dim.3    ctr   cos2
## SEBRLE      2.369   0.024    0.137  0.792  ...  0.106  0.827  1.187  0.122
## CLAY        3.507   0.024    0.300  1.235  ...  0.027  2.141  7.960  0.373
## KARPOV      3.396   0.024    0.281  1.358  ...  0.020  1.956  6.644  0.332
## BERNARD     2.763   0.024    0.186 -0.610  ...  0.100  0.890  1.375  0.104
## YURKOV      3.018   0.024    0.222 -0.586  ...  0.499 -1.225  2.606  0.165
## WARNERS     2.428   0.024    0.144  0.357  ...  0.482  0.767  1.020  0.100
## ZSIVOCZKY   2.563   0.024    0.160  0.272  ...  0.182 -1.283  2.857  0.250
## McMULLEN    2.561   0.024    0.160  0.588  ...  0.008 -0.418  0.303  0.027
## MARTINEAU   3.742   0.024    0.342 -1.995  ...  0.022 -0.730  0.925  0.038
## HERNU       2.794   0.024    0.190 -1.546  ...  0.031  0.841  1.227  0.091
##
## [10 rows x 12 columns]
##
## Continuous variables
##
##               dist  weight  inertia  Dim.1  ...   cos2  Dim.3     ctr   cos2
## 100m           1.0     1.0      1.0 -0.775  ...  0.035 -0.184   2.420  0.034
## Long_jump      1.0     1.0      1.0  0.742  ...  0.119  0.182   2.363  0.033
## Shot_put       1.0     1.0      1.0  0.623  ...  0.358 -0.023   0.039  0.001
## High_jump      1.0     1.0      1.0  0.572  ...  0.123 -0.260   4.794  0.067
## 400m           1.0     1.0      1.0 -0.680  ...  0.324  0.131   1.230  0.017
## 110m_hurdle    1.0     1.0      1.0 -0.746  ...  0.052 -0.093   0.611  0.009
## Discus         1.0     1.0      1.0  0.552  ...  0.368  0.043   0.131  0.002
## Pole_vault     1.0     1.0      1.0  0.050  ...  0.033  0.692  34.061  0.479
## Javeline       1.0     1.0      1.0  0.277  ...  0.100 -0.390  10.807  0.152
## 1500m          1.0     1.0      1.0 -0.058  ...  0.225  0.782  43.543  0.612
##
## [10 rows x 12 columns]
##
## Supplementary continuous variables
##
##          Dim.1   cos2  Dim.2   cos2  Dim.3   cos2
## Rank    -0.671  0.450  0.051  0.003 -0.058  0.003
## Points   0.956  0.914 -0.017  0.000 -0.066  0.004
##
## Supplementary categories
##
##             dist  Dim.1   cos2  v.test  ...  v.test  Dim.3   cos2  v.test
## Decastar   0.946 -0.600  0.403   -1.43  ...  -0.123  0.289  0.093    1.05
```

```
## OlympicG  0.439  0.279  0.403    1.43  ...   0.123 -0.134  0.093   -1.05
##
## [2 rows x 10 columns]
##
## Supplementary categorical variable (eta2)
##
##              Dim.1  Dim.2  Dim.3
## Competition  0.051    0.0  0.028
```

1.3.3 Graphes sur les dimensions 3 et 4

```
print(fviz_pca_ind(res_pca,axis=(2,3),repel=True))
```



1.4 Description des dimensions

On peut décrire les dimensions données par les variables en calculant le coefficient de corrélation entre une variable et une dimension et en réalisant un test de significativité.

```
from scientisttools import dimdesc
dim_desc = dimdesc(res_pca)
dim_desc.keys()
```

```
## dict_keys(['Dim.1', 'Dim.2', 'Dim.3', 'Dim.4', 'Dim.5'])
```

```
dim_desc["Dim.1"]
```

```
##             correlation        pvalue
## Points         0.956154  2.099191e-22
## Long_jump      0.741900  2.849886e-08
## Shot_put       0.622503  1.388321e-05
## High_jump      0.571945  9.362285e-05
## Discus         0.552467  1.802220e-04
## Rank          -0.670510  1.616348e-06
## 400m          -0.679610  1.028175e-06
## 110m_hurdle   -0.746245  2.136962e-08
## 100m          -0.774720  2.778467e-09
```

```
dim_desc["Dim.2"]
```

```
##             correlation    pvalue
## Discus         0.606313  0.000027
## Shot_put       0.598303  0.000036
## 400m           0.569438  0.000102
## 1500m          0.474224  0.001734
## High_jump      0.350294  0.024750
## Javeline       0.316989  0.043450
## Long_jump     -0.345421  0.026970
```

Ces tableaux donnent le coefficient de corrélation et la probabilité critique des variables qui sont significativement corrélées aux dimensions principales. Les variables actives et illustratives dont le probabilité critique est inférieure à 0.05 apparaissent.

Les tableaux de la description des deux axes principaux montrent que les variables "Points" et "Long_jump" sont les plus corrélées à la première dimension et que "Discus" est la variable la plus corrélée à la deuxième dimension. Ceci confirme la premièer interprétation.

Si on ne veut pas qu'un (ou plusieurs) individu participe à l'analyse, il est possible de l'ajouter en tant qu'individu illustratif. Ainsi, il ne sera pas actif dans l'analyse mais apportera de l'information supplémentaire.

Pour ajouter des individus illustratifs, utilisez l'argument suivant de la fonction PCA :

```
ind_sup
```

Tous les résultats détaillés peuvent être vus dans l'objet `res_pca`. On peut récupérer les valeurs propres, les résultats des individus actifs et illustratifs, les résultats des variables actives et les résultats des variables continues et qualitatives illustratives en tapant :

```
from scientisttools import get_pca_ind,get_pca_var,get_eig
eig = get_eig(res_pca)
```

```
row = get_pca_ind(res_pca)
print(row.keys())
```

```
## dict_keys(['coord', 'cos2', 'contrib', 'dist', 'infos'])
```

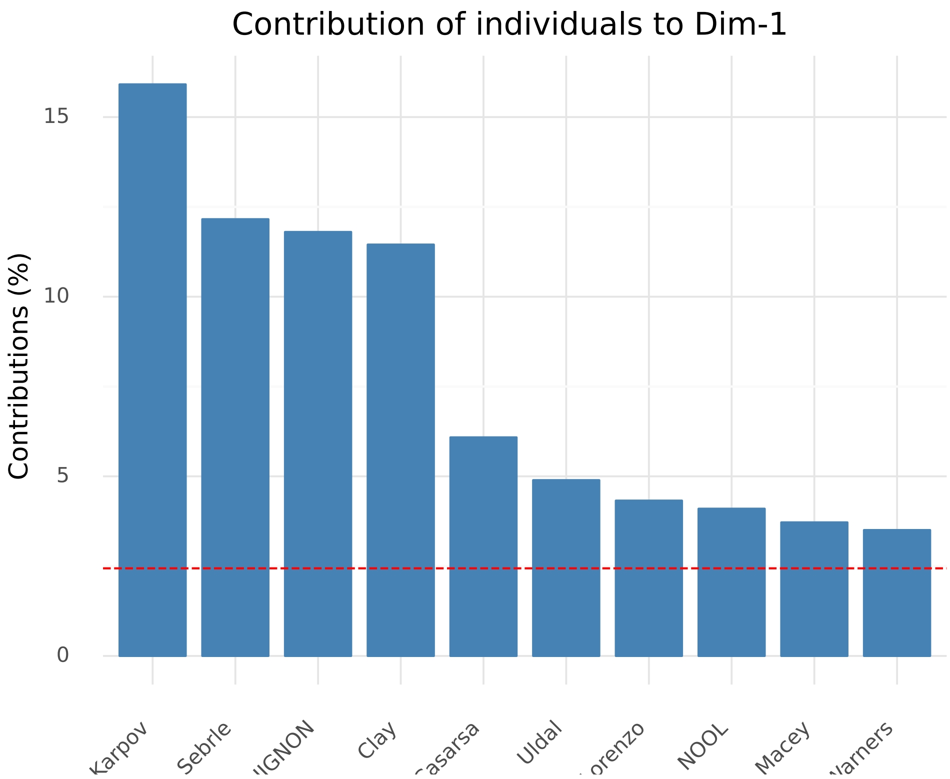
```
var = get_pca_var(res_pca)
print(var.keys())
```

```
## dict_keys(['coord', 'cor', 'cos2', 'contrib', 'weighted_corr', 'infos'])
```

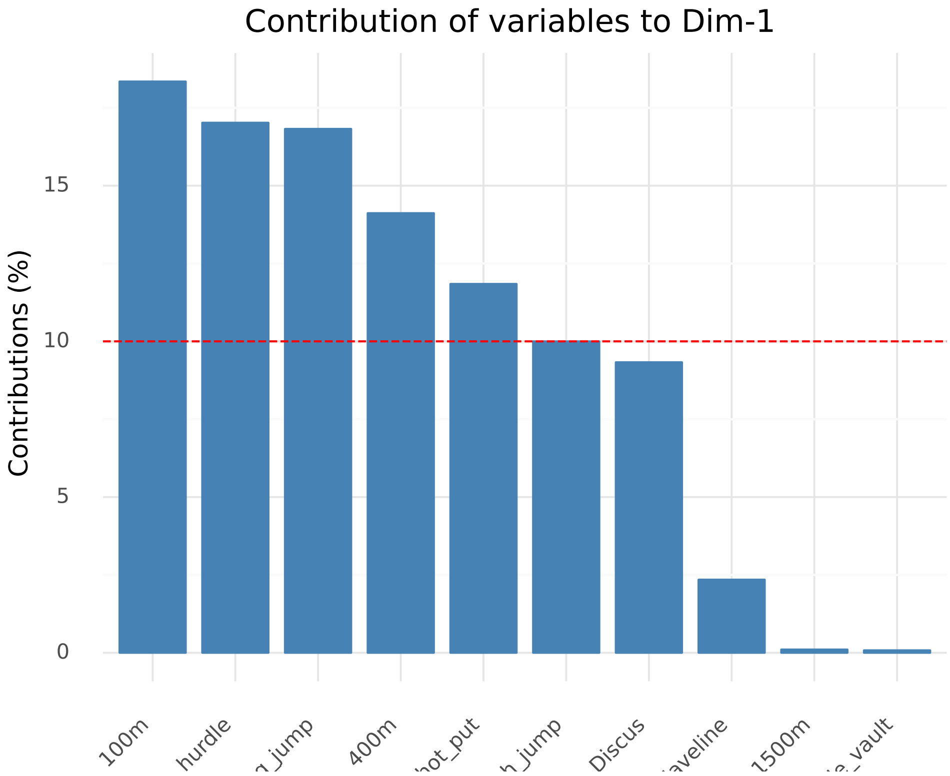
1.5 Interprétation des axes

Des graphiques qui permettent d'interpréter rapidement les axes : on choisit un axe factoriel (le 1er axe dans notre exemple) et on observe quels sont les points lignes et colonnes qui présentent les plus fortes contributions et cos2 pour cet axe.

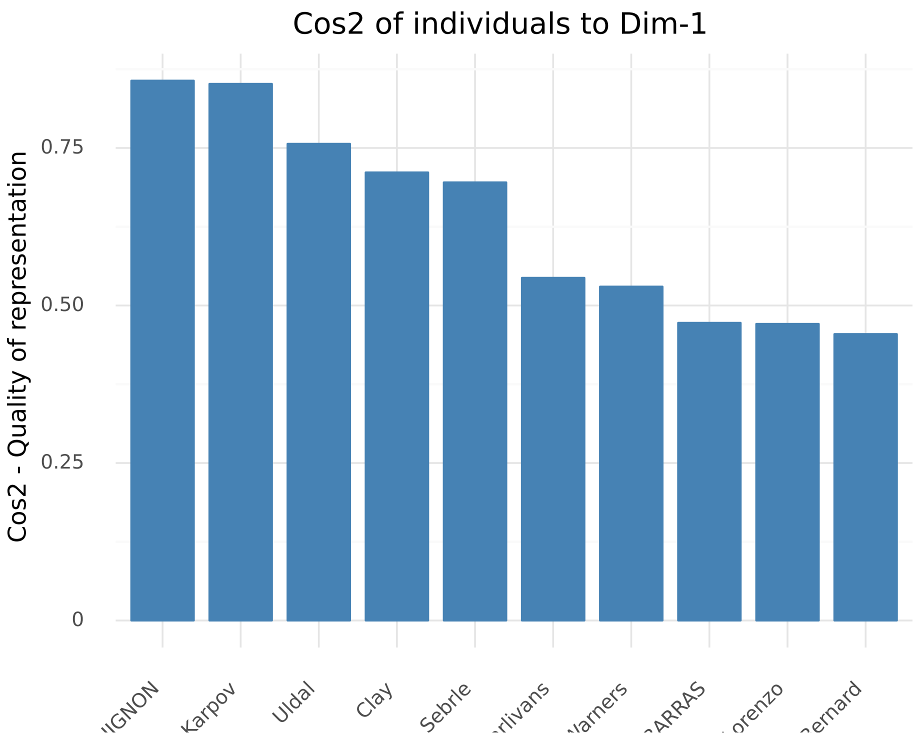
```
# Classement des points lignes en fonction de leur contribution au 1er axe
from scientisttools import fviz_contrib, fviz_cos2
print(fviz_contrib(res_pca,choice="ind",axis=0,top_contrib=10))
```



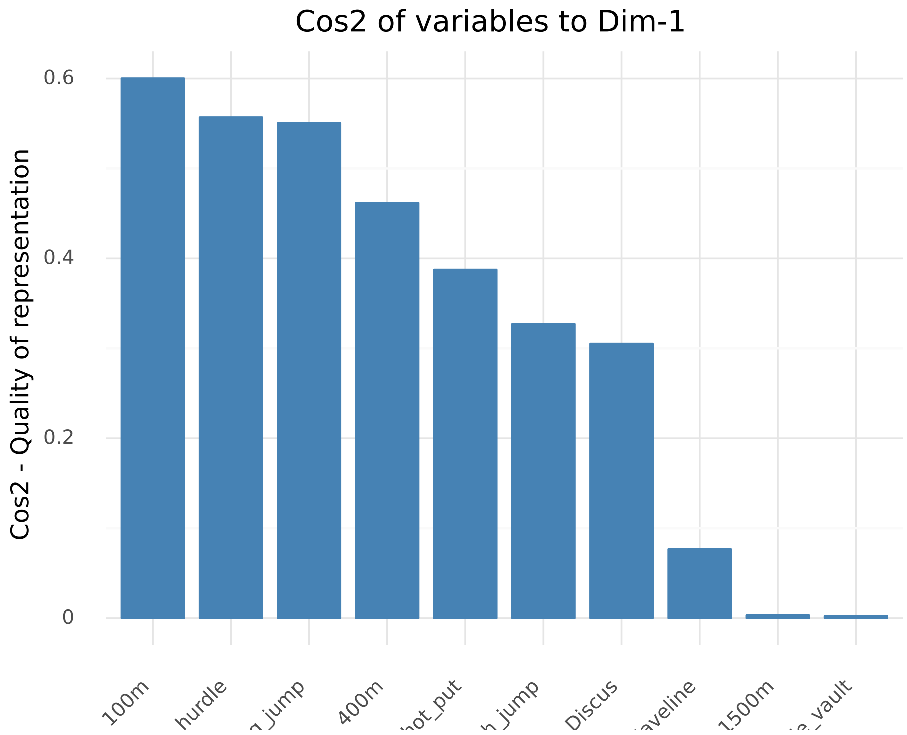

```
# Classement des points colonnes en fonction de leur contribution au 1er axe
print(fviz_contrib(res_pca,choice="var",axis=0))
```




```
# Classement des points lignes en fonction de leur cos2 sur le 1er axe
print(fviz_cos2(res_pca,choice="ind",axis=0,top_cos2=10))
```



```python
# Classement des points colonnes en fonction de leur cos2 sur le 1er axe
print(fviz_cos2(res_pca,choice="var",axis=0))
```



1.6 Approche Machine Learning

Ici, l'objectif est d'utiliser l'Analyse en Composantes Principales en tant que méthode de prétraitement.

La classe PCA implémente les méthodes `fit`, `transform` et `fit_transform` bien connues des utilisateurs de scikit-learn.

```python
res_pca.transform(actif).iloc[:5,:]
```

```
##             Dim.1     Dim.2     Dim.3     Dim.4     Dim.5
## SEBRLE   0.791628  0.771611  0.826841  1.174627  0.707159
## CLAY     1.234991  0.574578  2.141247 -0.354845 -1.974571
## KARPOV   1.358215  0.484021  1.956258 -1.856524  0.795215
## BERNARD -0.609515 -0.874629  0.889941  2.220612  0.361636
## YURKOV  -0.585968  2.130954 -1.225157  0.873579  1.251369
```

```python
res_pca.fit_transform(decathlon).iloc[:5,:]
```

```
##             Dim.1     Dim.2     Dim.3     Dim.4     Dim.5
## SEBRLE   0.791628  0.771611  0.826841  1.174627  0.707159
## CLAY     1.234991  0.574578  2.141247 -0.354845 -1.974571
## KARPOV   1.358215  0.484021  1.956258 -1.856524  0.795215
## BERNARD -0.609515 -0.874629  0.889941  2.220612  0.361636
## YURKOV  -0.585968  2.130954 -1.225157  0.873579  1.251369
```

1.6.1 Intégration dans une Pipeline de scikit-learn

La class PCA peut être intégrée dans une Pipeline de scikit-learn. Dans le cadre de notre exemple, nous cherchons à prédire la 13ème variable (variable "Competition") à partir des 12 premières variables du jeu de données.

"Competition" est une variable catégorielle binaire. Pour la prédire, nous allons utiliser un modèle de régression logistique qui prendra en input des axes issus d'une Analyse en Composantes Principales pratiquée sur les données brutes.

Dans un premier temps, et de façon tout à fait arbitraire, nous fixons le nombre de composantes extraites à 4.

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
import numpy as np

# X = features
X = decathlon.drop(columns=["Competition"])
# y = labels
y = decathlon[["Competition"]]

# Construction de la Pipeline
# On enchaine une Analyse en Composantes Prnicipales (4 axes retenus)
# puis une régression logistique
pipe = Pipeline([("pca", PCA(n_components=4)),
                 ("logistic_regression", LogisticRegression(penalty=None))])
# Estimation du modèle
pipe.fit(X, y)
```

```
## Pipeline(steps=[('pca', PCA(n_components=4)),
##                 ('logistic_regression', LogisticRegression(penalty=None))])
```

On prédit

```
# Prédiction sur l'échantillon de test
print(pipe.predict(X))
```

```
## ['OlympicG' 'OlympicG' 'OlympicG' 'Decastar' 'OlympicG' 'OlympicG'
##  'OlympicG' 'OlympicG' 'OlympicG' 'OlympicG' 'OlympicG' 'OlympicG'
##  'Decastar' 'OlympicG' 'OlympicG' 'OlympicG' 'OlympicG' 'OlympicG'
##  'OlympicG' 'OlympicG' 'Decastar' 'OlympicG' 'OlympicG' 'OlympicG'
##  'OlympicG' 'OlympicG' 'OlympicG' 'OlympicG' 'OlympicG' 'OlympicG'
##  'OlympicG' 'OlympicG' 'OlympicG' 'Decastar' 'OlympicG' 'OlympicG'
##  'OlympicG' 'OlympicG' 'OlympicG' 'OlympicG' 'OlympicG']
```

Le paramètre `n_components` peut faire l'objet d'une optimisation via GridSearchCV de scikit-learn.

Nous reconstruisons donc une Pipeline, sans spécifier de valeur a priori pour n_components.

```
# Reconstruction d'une Pipeline, sans spécifier de valeur
# a priori pour n_components
pipe2 = Pipeline([("pca", PCA()),
                  ("logistic_regression", LogisticRegression(penalty=None))])

# Paramétrage de la grille de paramètres
# Attention à l'étendue des valeurs possibles pour pca__n_components !!!
param = [{"pca__n_components": [x + 1 for x in range(12)]}]

# Construction de l'obet GridSearchCV
grid_search = GridSearchCV(pipe2,
                           param_grid=param,
                           scoring="accuracy",
                           cv=5,
                           verbose=0)
# Estimation du modèle
grid_search.fit(X, y)
```

```
## GridSearchCV(cv=5,
##              estimator=Pipeline(steps=[('pca', PCA()),
##                                        ('logistic_regression',
##                                         LogisticRegression(penalty=None))]),
##              param_grid=[{'pca__n_components': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
##                                                 11, 12]}],
##              scoring='accuracy')
```

```
# Affichage du score optimal
grid_search.best_score_
```

```
## 0.95
```

```
# Affichage du paramètre optimal
grid_search.best_params_
```

```
## {'pca__n_components': 9}
```

```
# Prédiction sur l'échantillon de test
grid_search.predict(X)
```

```
## array(['Decastar', 'Decastar', 'Decastar', 'Decastar', 'Decastar',
##        'Decastar', 'Decastar', 'Decastar', 'Decastar', 'Decastar',
##        'Decastar', 'Decastar', 'Decastar', 'OlympicG', 'OlympicG',
##        'OlympicG', 'OlympicG', 'OlympicG', 'OlympicG', 'OlympicG',
##        'OlympicG', 'OlympicG', 'OlympicG', 'OlympicG', 'OlympicG',
##        'OlympicG', 'OlympicG', 'OlympicG', 'OlympicG', 'OlympicG',
##        'OlympicG', 'OlympicG', 'OlympicG', 'OlympicG', 'OlympicG',
##        'OlympicG', 'OlympicG', 'OlympicG', 'OlympicG', 'OlympicG',
##        'OlympicG'], dtype=object)
```

Pour plus d'informations sur l'ACP sous scientisttools, consulter le notebook
https ://github.com/enfantbenidedieu/scientisttools/blob/master/notebooks/pca_example.ipynb

2 Analyse Factorielle des Correspondances

Sommaire

Ce chapitre a pour objectif de présenter rapidement les principales fonctionnalités offertes par le package « scientisttools » pour réaliser une Analyse Factorielle des Correspondances.

2.1 Présentation des données

Les données sur lesquelles nous allons travailler proviennent du site http ://factominer.free.fr/facto factorielle-des-correspondances.html. Il s'agit des données issues d'un questionnaire réalisé sur des françaises en 1974.

Ces données sont issues d'une enquête du CREDOC publiée en 1974 par Nicole Tabard, intitulée Besoins et aspirations des familles et des jeunes. 1724 femmes ont répondu à différentes questions à propos du travail des femmes, parmi lesquelles :

1. Quelle est selon vous la famille parfaite ?
   — L'homme et la femme travaillent
   — L'homme travaille plus que la femme
   — Seul l'homme travaille
2. Quelle activité est la meilleure pour une mère quand les enfants vont à l'école ?
   — Rester à la maison
   — Travailler à mi - temps
   — Travailler à temps complet
3. Que pensez - vous de la phrase suivante : les femmes qui ne travaillent pas se sentent coupées du monde ?
   — Complètement d'accord
   — Plutôt d'accord

— Plutôt en désaccord
— Complètement en désaccord

Le tableau de données est formé de deux tableaux de contingence qui croisent les réponses de la première question à celles des deux autres.

Nous pouvons charger les données sur http ://factominer.free.fr/factomethods/datasets/women_w

```
# Chargement des données
import pandas as pd
url = "http://factominer.free.fr/factomethods/datasets/women_work.txt"
women_work = pd.read_table(url,header=0)
women_work.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 3 entries, both.man.and.woman.work to only.man.works
## Data columns (total 7 columns):
##  #   Column                                 Non-Null Count  Dtype
## ---  ------                                 --------------  -----
##  0   stay.at.home                           3 non-null      int64
##  1   part-time.work                         3 non-null      int64
##  2   full-time.work                         3 non-null      int64
##  3   housewives.cut.from.world.totally.agree     3 non-null      int64
##  4   housewives.cut.from.world.quite.agree       3 non-null      int64
##  5   housewives.cut.from.world.quite.disagree    3 non-null      int64
##  6   housewives.cut.from.world.totally.disagree  3 non-null      int64
## dtypes: int64(7)
## memory usage: 192.0+ bytes
```

Table 2.1 – Données d'enquête

	stay.at.home	part-time.work	full-time.work	housewives.cut.from.world.totally.agree	housewives.cut.from.world.quite.agree	housewives.cut.from.world.quite.disagree	housewives.cut.from.world.totally.disagree
both.man.and.woman.work	13	142	106	107	75	40	39
man.morks.more	30	408	117	192	175	100	88
only.man.works	241	573	94	140	215	254	299

Chaque valeur du tableau 2.1 correspond au nombre de femmes ayant donnée la réponse en ligne et la réponse en colonne.

Le point de départ de l'analyse est le tableau de contingence reproduit ci-dessous.

Table 2.2 – Données d'enquête

	stay.at.home	part-time.work	full-time.work
both.man.and.woman.work	13	142	106
man.morks.more	30	408	117
only.man.works	241	573	94

C'est ce type de données (les marges des totaux mis à part) que nous fournirons à la fonction de calcul de l'AFC.

Comme le souligne François Husson dans le MOOC Analyse des données multidimensionnelles sur la plateforme FUN, il est difficile de savoir à partir de ce tableau si les femmes sont favorables ou non au travail féminin. En effet, 908 femmes sur 1724, soit 52% ont répondu que la famille idéale est celle où « Only man works ». Elles sont néanmoins 1123 sur 1724 (65%) à avoir répondu que l'activité convenant le mieux à une mère de famille quand ses enfants vont à l'école est de travailler à mi-temps « part-time work ». L'AFC va nous permettre d'étudier le lien entre ces deux questions et de lever cette apparente contradiction. Elle va notamment nous permettre de visualiser la nature de la liaison entre les deux questions. Mais qu'est ce qu'une liaison ?

Une liaison entre deux variables est l'écart entre les données observées et le modèle d'indépendance. Mettons pour l'instant de côté cette notion, nous y reviendrons plus tard.

2.2 Objectifs

Les objectifs de l'AFC sont similaires à ceux de l'ACP : obtenir une typologie des lignes et des colonnes et étudier le lien entre ces deux typologies.

Cependant, le concept de similarité entre les lignes et les colonnes est diférent. Ici, la similarité entre deux lignes ou deux colonnes est complètement symétrique. Deux lignes (resp. colonnes) sont proches l'une de l'autre si elles s'associent aux colonnes (resp. lignes) de la même façon.

On recherche les lignes (resp. colonnes) dont la distribution est la plus différente de celle de la population. Celles qui semblent le plus ou le moins semblables.

Chaque groupe de lignes (resp. colonnes) est caractérisé par les colonnes (resp. lignes) auxquelles il est particulièrement ou particulièrement peu associé.

Nous travaillons d'abord avec seulement les 3 premières colonnes : « Stay at home », « Part time work » et « Full time work ».

```
# Selection des 3 premières colonnes
wfemmes = women_work.iloc[:,:3]
```

Table 2.3 – Données d'enquête - Tableau des données observées

	stay.at.home	part-time.work	full-time.work
both.man.and.woman.work	13	142	106
man.morks.more	30	408	117
only.man.works	241	573	94

Notons que nous pouvons calculer les marges lignes et les marges colonnes de ce tableau de contingence de la manière suivante :

```python
# Ajout des marges lignes et colonnes
wfemmes_avec_marges = wfemmes.copy()
wfemmes_avec_marges.loc["Total",:] = wfemmes.sum(axis=0)
wfemmes_avec_marges.loc[:,"Total"] = wfemmes_avec_marges.sum(axis=1)
```

Table 2.4 – Données d'enquête avec marge ligne et colonne

	stay.at.home	part-time.work	full-time.work	Total
both.man.and.woman.work	13	142	106	261
man.morks.more	30	408	117	555
only.man.works	241	573	94	908
Total	284	1123	317	1724

Il est aussi intéressant de calculer les pourcentages en ligne et les pourcentages en colonne.

```python
# Pourcentages en ligne
import numpy as np
wfemmes_pourcentage_en_ligne = wfemmes.copy()
wfemmes_pourcentage_en_ligne.loc["Profil ligne moyen",:] = wfemmes.sum(axis=0)
wfemmes_pourcentage_en_ligne = wfemmes_pourcentage_en_ligne.apply(
                            lambda x : 100*x/np.sum(x),axis=1)
wfemmes_pourcentage_en_ligne.loc[:,"Total"] = wfemmes_pourcentage_en_ligne.sum(
                                      axis=1)
```

Table 2.5 – Données d'enquête - Tableau des pourcentages en ligne

	stay.at.home	part-time.work	full-time.work	Total
both.man.and.woman.work	4.98	54.41	40.61	100
man.morks.more	5.41	73.51	21.08	100
only.man.works	26.54	63.11	10.35	100
Profil ligne moyen	16.47	65.14	18.39	100

Pour rappel, la ligne « Profil ligne moyen » correspond à la répartition en pourcentage des modalités à la question sur « l'activité qui convient le mieux à une mère de famille quand les enfants vont à l'école », quelque soit la réponse à la question sur la famille idéale. Le profil ligne moyen peut être comparé aux profils lignes (la répartition en pourcentages ou la distribution de probabilité d'une modalité en ligne). Ici, aucun des trois profils lignes n'est proche du profil ligne moyen.

Calculons maintenant le tableau des pourcentages en colonne

```python
# Pourcentage en colonne
wfemmes_pourcentage_en_colonne = wfemmes.copy()
wfemmes_pourcentage_en_colonne.loc[:,"Profil colonne moyen"] = wfemmes.sum(axis=1)
wfemmes_pourcentage_en_colonne = wfemmes_pourcentage_en_colonne.apply(
                            lambda x : 100*x/np.sum(x),axis=0)
wfemmes_pourcentage_en_colonne.loc["Total",:] = wfemmes_pourcentage_en_colonne.sum(
                                      axis=0)
```

Ce tableau permet de constater que la répartition des réponses sur la famille idéale pour la modalité « Part-time work » est le plus proche de la répartion des réponses à

Table 2.6 – Données d'enquête - Tableau des pourcentages en colonne

	stay.at.home	part-time.work	full-time.work	Profil colonne moyen
both.man.and.woman.work	4.58	12.64	33.44	15.14
man.morks.more	10.56	36.33	36.91	32.19
only.man.works	84.86	51.02	29.65	52.67
Total	100.00	100.00	100.00	100.00

la question sur la famille idéale. Autrement dit, le profil colonne « Part-time work » est le profil colonne le plus proche du profil colonne moyen. Cette similitude se traduira sur le graphe de l'AFC comme nous le verrons plus loin.

Nous verrons également que l'on passera en paramètre à la fonction Python de calcul de l'AFC, le tableau de contingence. Mais l'AFC travaille en réalité sur le tableau de probabilités que l'on peut calculer en divisant les valeurs du tableau de contingence par le nombre d'individus (on effectue le calcul sur le tableau de contingence avec marge pour mieux constater que l'effectif total du tableau de probabilité est bien égal à 1, ce qui est la marque d'une distribution de probabilités) :

```
# Tableau des probabilités
wfemmes_tableau_de_probabilite = wfemmes_avec_marges/1724
```

Table 2.7 – Données d'enquête - Tableau de probabilité

	stay.at.home	part-time.work	full-time.work	Total
both.man.and.woman.work	0.00754	0.08237	0.06148	0.15139
man.morks.more	0.01740	0.23666	0.06787	0.32193
only.man.works	0.13979	0.33237	0.05452	0.52668
Total	0.16473	0.65139	0.18387	1.00000

Rappellons que notre objectif est de visualiser la nature de la liaison entre deux variables qualitatives. Mais faut-il encore que cette liaison soit significative. Pour ce faire, nous réalisons un test du Khi2.

2.2.1 Test du χ^2

Le test du χ^2 mesure la significativité d'une liaison mais pas son intensité. Afin de réaliser ce test du χ^2, nous utilisons la fonction chi_contingency de scipy.

```
# Test de contingence du chi2
import scipy.stats as st
stat, pvalue, dof, expected = st.chi2_contingency(wfemmes)
chisq_test = pd.DataFrame({"statistic":stat,"dof":dof,"pvalue":pvalue},
                          index=["chi2 - test"])
print(chisq_test)
```

```
##               statistic  dof        pvalue
## chi2 - test  233.430417    4  2.410248e-49
```

La fonction chi_contingency nous donne, entre autres, la valeur du χ^2 qui est un indicateur de la significativité de la liaison. Mais ce qui nous interesse ici est la p-value. Nous voyons ici que la p-value est égale à $2.4102475 \times 10^{-49}$. Cela signifie que la probabilité que les variables soient indépendantes est égale à $2.4102475 \times 10^{-49}$. Ce qui nous permet de rejeter l'hypothèse d'indépendance entre les deux variables. Pour autant, cela ne veut pas dire que les variables soient dépendantes. Les réponses à la question sur la famille idéale sont probablement liées aux réponses concernant l'activité convenant le mieux à une mère de famille dont les enfants vont à l'école.

2.2.2 Test de χ^2 - Explications

Le test du χ^2 permet de déterminer la probabilité que les deux variables d'un tableau de contingence soient indépendantes, c'est-à-dire qu'il n'existe pas de relation entre les modalités en ligne et les modalités en colonne (les unes ne conditionnent pas les autres, et réciproquement). Dit autrement et comme le rappelle très clairement Julien Barnier, cela veut dire que le « fait d'appartenir à une modalité de la première variable n'a pas d'influence sur la modalité d'appartenance de la deuxième variable ». Dans ce test, l'hypothèse nulle (H0) suppose qu'il y a indépendance entre les deux variables. Si nous acceptons l'hypothèse d'indépendance (H0), nous n'aurons pas d'utilité à réaliser une AFC car les points projetés seront extrêmement proches ou confondus avec le centre de gravité, confondus avec le centre du graphe. Si nous rejetons l'hypothèse d'indépendance (p-value $< 0,05$), l'hypothèse alternative (H1) suppose que la liaison entre les deux variables est significative sans que nous puissions définir l'intensité de la liaison.

Rappelons que pour que le test du χ^2 soit opératoire, il doit respecter un certain nombre de conditions (pour reprendre les propos de Claude Grasland) :

- L'effectif total du tableau de contingence doit être supérieur ou égal à 20.
- L'effectif marginal du tableau de contingence doit toujours être supérieur ou égal à 5.
- L'effectif théorique des cases du tableau de contingence doit être supérieur à 5 dans 80% des cases du tableau de contingence.

Du fait que nous ayons obtenu une p-value égale à $2.4102475 \times 10^{-49}$ et, par extension, inférieure au seuil de $0,05$, nous rejetons l'hypothèse d'indépendance entre les deux variables.

2.2.3 Test du χ^2 - Aide à l'interprétation

Le test du χ^2 est symétrique. Les lignes et les colonnes du tableau croisé sont interchangeables. Le résultat du test sera exactement le même. Il n'y a pas de « sens de lecture » du tableau.

Nous pouvons afficher le tableau d'indépendance (tableau des effectifs théoriques) en sélectionnant la valeur `expected`. Dans ce contexte, nous calculons le tableau des pourcentages théoriques, en multipliant pour chaque case la proportion observée dans la population des deux modalités correspondantes. Puis, le tableau des effectifs théoriques se calcule en multipliant le tableau des pourcentages théoriques par l'effectif total.

```
# Tableau des effectifs théoriques
effectif_theorik = pd.DataFrame(expected,index=wfemmes.index,
                                columns=wfemmes.columns)
```

Table 2.8 – Données d'enquête - Tableau des effectifs théoriques

	stay.at.home	part-time.work	full-time.work
both.man.and.woman.work	42.99536	170.0133	47.9913
man.morks.more	91.42691	361.5226	102.0505
only.man.works	149.57773	591.4640	166.9582

Le tableau des effectifs théoriques n'a que peu d'intérêt en lui-même mais en a davantage comparativement au tableau des données observées.

Nous pouvons aussi afficher le tableau des résidus standardisés (tableau des écarts à l'indépendance). Un résidu standardisé positif signifie que les effectifs dans la case sont supérieur à ceux attendus sous l'hypothèse d'indépendance. Et l'inverse pour un résidu standardisé négatif.

```
# Residus standardisés
standardized_residuals = (wfemmes - effectif_theorik)/np.sqrt(effectif_theorik)
```

Table 2.9 – Données d'enquête - Résidus standardisés

	stay.at.home	part-time.work	full-time.work
both.man.and.woman.work	-4.57450	-2.14844	8.37359
man.morks.more	-6.42424	2.44441	1.47986
only.man.works	7.47513	-0.75921	-5.64638

Exprimé d'une autre manière, l'écart à l'indépendance représente l'écart entre l'effectif observé et l'effectif théorique, et ceci pour chacune des cases du tableau de contingence. D'ailleurs, comme le note Philippe Cibois, l'écart à l'indépendance « est un effectif et c'est un invariant, indépendant du choix des lignes et des colonnes (c'est la différence entre l'effectif observé et l'effectif théorique : le résultat est donc un effectif). » Par ailleurs,

— Un écart à l'indépendance positif correspond à une attraction entre les deux modalités pour la case observée.
— À l'inverse, un écart à l'indépendance négatif correspond à une opposition entre les deux modalités pour la case observée.

Plus la valeur de l'écart à l'indépendance est importante, plus l'attraction/opposition entre les modalités est forte.

2.3 AFC

Notre objectif est bien de visualiser la nature de la liaison entre les deux variables qualitatives. Sachant qu'une liaison correspond à l'écart entre les données observées et le modèle d'indépendance, nous souhaitons donc visualiser la nature de l'écart à l'indépendance entre deux variables qualitatives.

Par ailleurs, il y a trois façons de caractériser la liaison entre les deux variables qualitatives.

— La significativité de la liaison (qui se mesure avec le test du χ^2).
— L'intensité de la liaison (qui se mesure, entre autre, avec le ϕ^2).
— La nature de la liaison (qui correspond à l'association entre les modalités et qui est représentée par le biais de l'AFC).

Le test du χ^2 a permis d'écarter l'hypothèse d'indépendance. Il y a donc une liaison entre les modalités des deux variables. De fait, nous pouvons faire une AFC pour visualiser la nature de la liaison. Pour notre part, nous avons choisi d'utiliser le package « scientisttools » (dédié à l'analyse multidimensionnelle de données).

On utilisera les trois première colonnes (correspondant aux réponses de la deuxième question) comme variables actives et les quatre dernières (correspondant à la troisième question) comme variables illustratives.

Nous chargeons donc la librairie « scientisttools »

```
# Chargement de la librairie
from scientisttools import CA
```

2.3.1 Lignes et colonnes actives seulement

Lors du précédent test du χ^2, nous avons obtenu une p-value égale à $2.4102475 \times 10^{-49}$. Nous avons donc rejeté l'hypothèse d'indépendance entre les deux variables et admis que la liaison entre ces deux variables est significative. Nous sommes en droit de réaliser une AFC afin de visualiser la nature de la liaison. Pour ce faire, nous allons employer la fonction `CA`, fournie par le package « scientisttools ».

On crée une instance de la classe CA, en lui passant ici des étiquettes pour les lignes et les colonnes. Ces paramètres sont facultatifs ; en leur absence, le programme détermine automatiquement des étiquettes.

```
# Instanciation du modèle
my_ca = CA()
```

On estime le modèle en appliquant la méthode `fit` de la classe CA sur le jeu de données.

```
# Entraînement - Estimation du modèle
my_ca.fit(wfemmes)
```

```
## CA()
```

2.3.2 Valeurs propres

L'exécution de la méthode `my_ca.fit(wfemmes)` provoque le calcul des attributs parmi lesquels `my_ca.eig_` pour les valeurs propres.

```
# Valeurs propres
print(my_ca.eig_)
```

```
##        eigenvalue  difference  proportion  cumulative
## Dim.1     0.11684     0.09828   86.292183   86.292183
## Dim.2     0.01856         NaN   13.707817  100.000000
```

L'attribut `my_ca.eig_` contient :

— en 1ère ligne : les valeurs propres en valeur absolue

— en 2ème ligne : les différences des valeurs propres
— en 3ème ligne : les valeurs propres en pourcentage de la variance totale (proportions)
— en 4ème ligne : les valeurs propres en pourcentage cumulé de la variance totale.

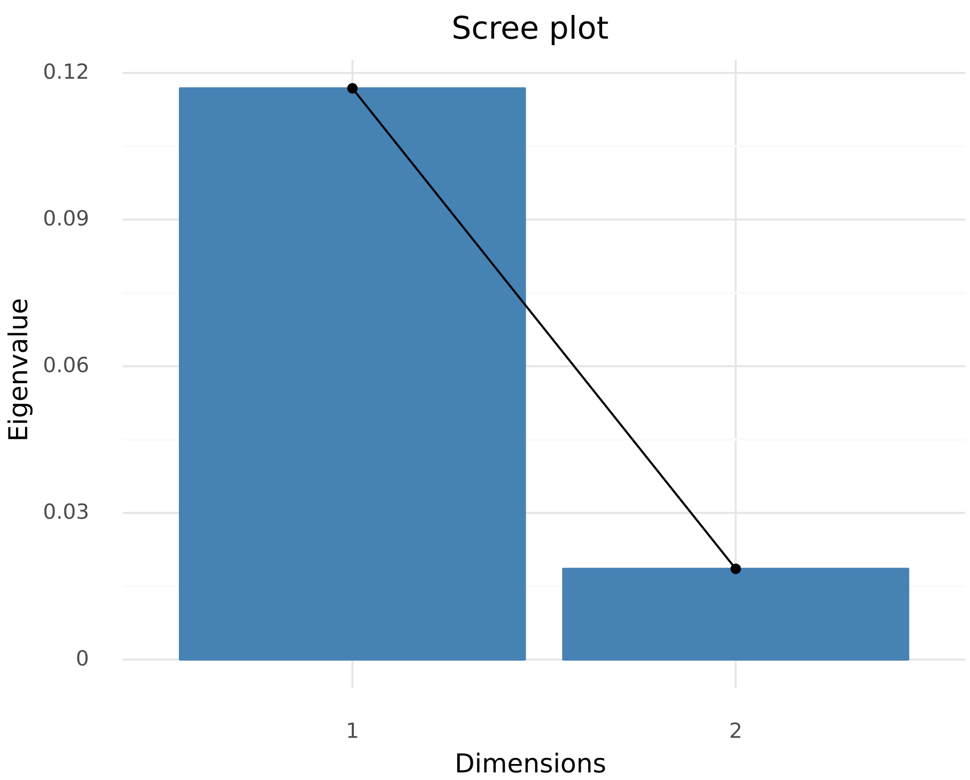
La fonction get_eig retourne les valeurs propres sous forme de tableau de données.

```
# Valeurs propres
from scientisttools import get_eig
print(get_eig(my_ca))
```

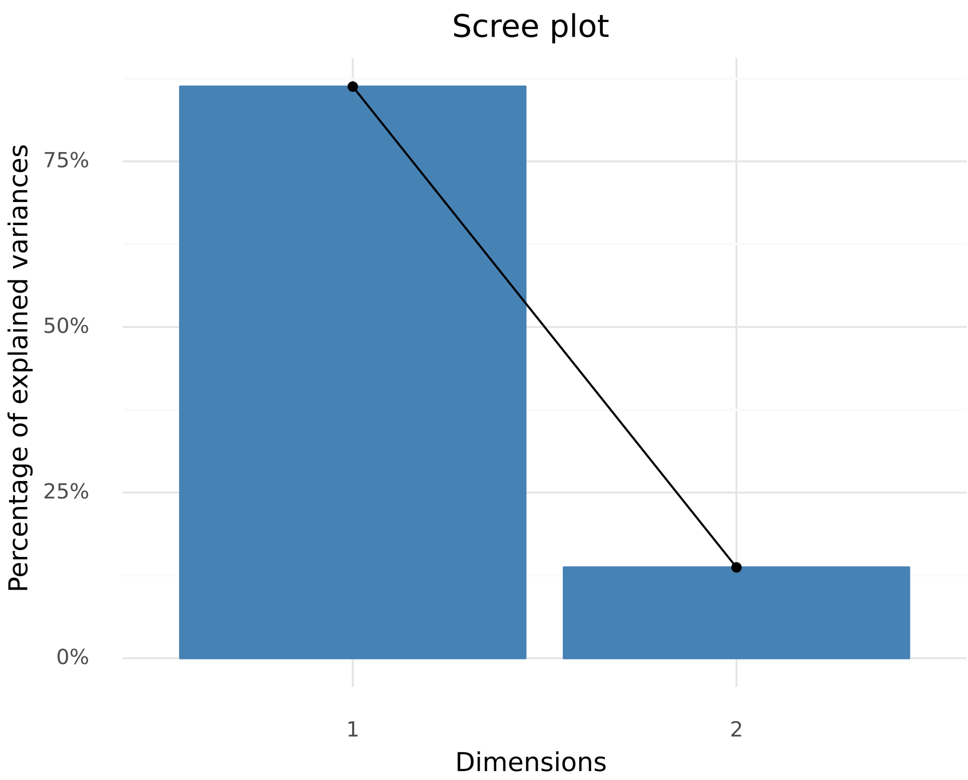
```
##        eigenvalue  difference  proportion  cumulative
## Dim.1     0.11684     0.09828   86.292183   86.292183
## Dim.2     0.01856         NaN   13.707817  100.000000
```

Les valeurs propres peuvent être représentées graphiquement

```
from scientisttools import fviz_eig
print(fviz_eig(my_ca,choice="eigenvalue"))
```




```
print(fviz_eig(my_ca,choice="proportion"))
```



On peut obtenir un résumé des principaux résultats en utilisant la fonction `summaryCA`.

```
from scientisttools import summaryCA
summaryCA(my_ca)
```

```
##                      Correspondence Analysis - Results
##
## Importance of components
##                             Dim.1    Dim.2
## Variance                    0.117    0.019
## Difference                  0.098      NaN
## % of var.                  86.292   13.708
## Cumulative of % of var.    86.292  100.000
##
## Rows
##
##                             dist  marge  inertia  ...  Dim.2     ctr   cos2
## both.man.and.woman.work    0.605  0.151    0.055  ...  0.233  44.429  0.149
## man.works.more             0.298  0.322    0.029  ... -0.172  51.436  0.333
## only.man.works             0.312  0.527    0.051  ...  0.038   4.135  0.015
##
## [3 rows x 9 columns]
##
## Columns
##
##                  dist  marge  inertia  Dim.1  ...   cos2  Dim.2     ctr   cos2
## stay.at.home    0.645  0.165    0.068  0.618  ...  0.920  0.183  29.613  0.080
```

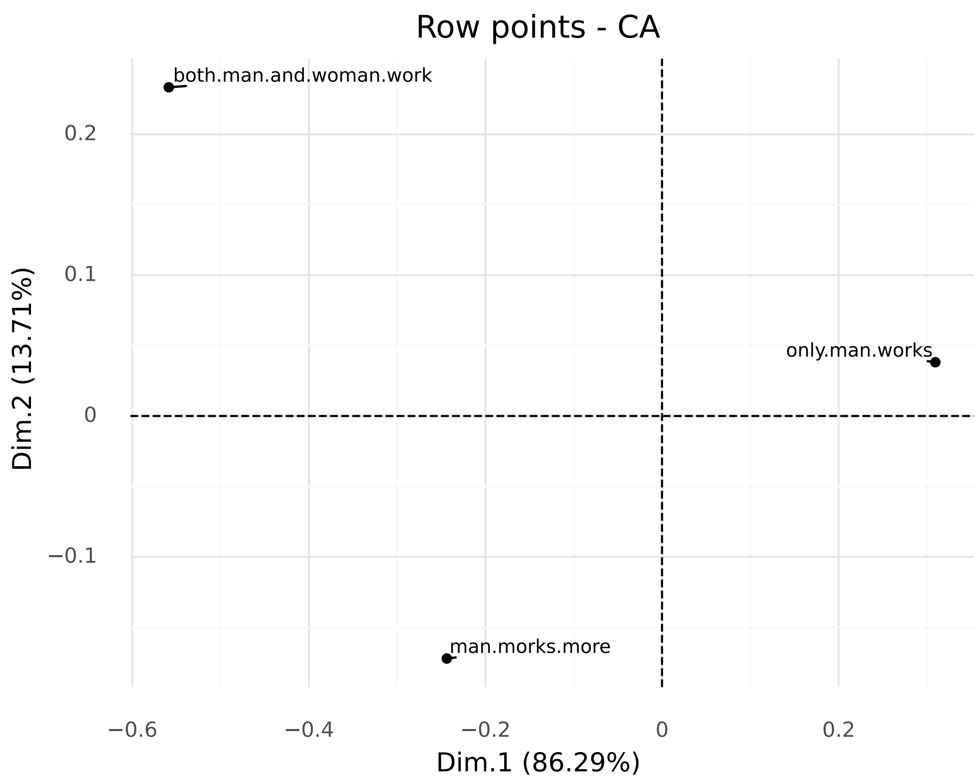
```
## part-time.work  0.100  0.651   0.006 -0.004  ...  0.001 -0.100  34.853  0.999
## full-time.work  0.573  0.184   0.060 -0.541  ...  0.891  0.189  35.533  0.109
##
## [3 rows x 9 columns]
```

Cette fonction `summaryCA` nous permet d'obtenir :

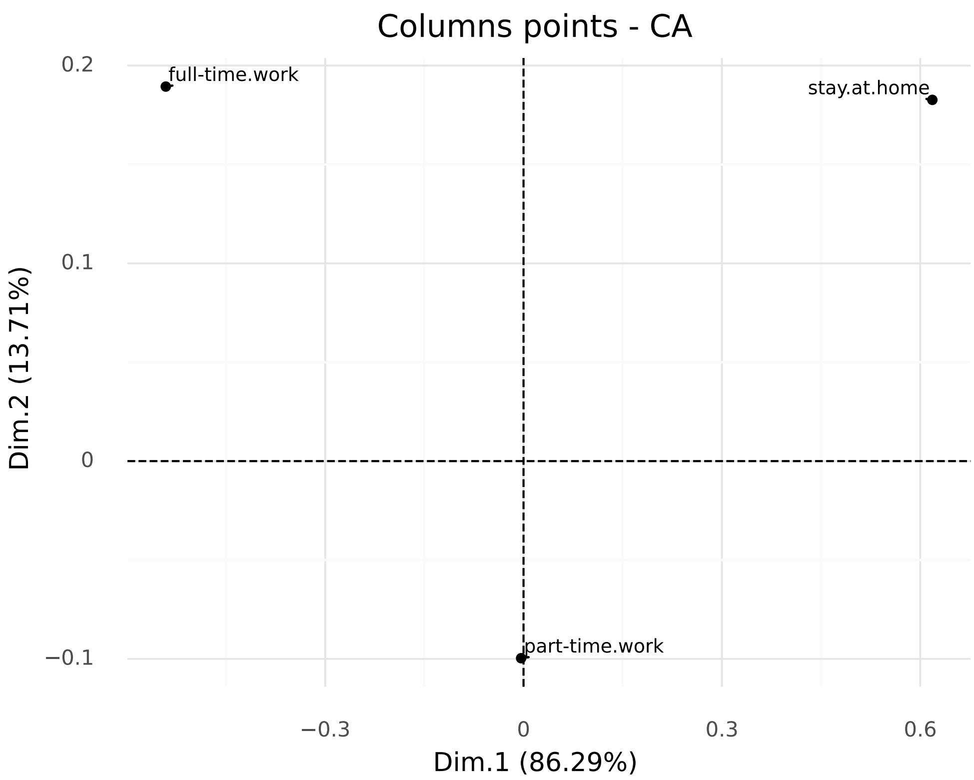
— Un tableau avec les valeurs propres, les différences, les pourcentages et les pourcentages cumulés d'inertie associés à chaque dimension.
— Un tableau avec les résultats sur les lignes actives avec leur coordonnées (Dim.n) sur chaque dimension, leur contribution à la construction (ctr) de chaque dimension et leur qualité de représentation (cos2) sur chaque dimension.
— Un tableau avec les résultats sur les colonnes actives (Dim.n, ctr, cos2)

2.3.3 Représentation graphique

```
# Carte des points lignes
from scientisttools import fviz_ca_row
print(fviz_ca_row(my_ca,repel=True))
```




```
# Carte des points colonnes
from scientisttools import fviz_ca_col
print(fviz_ca_col(my_ca,repel=True))
```



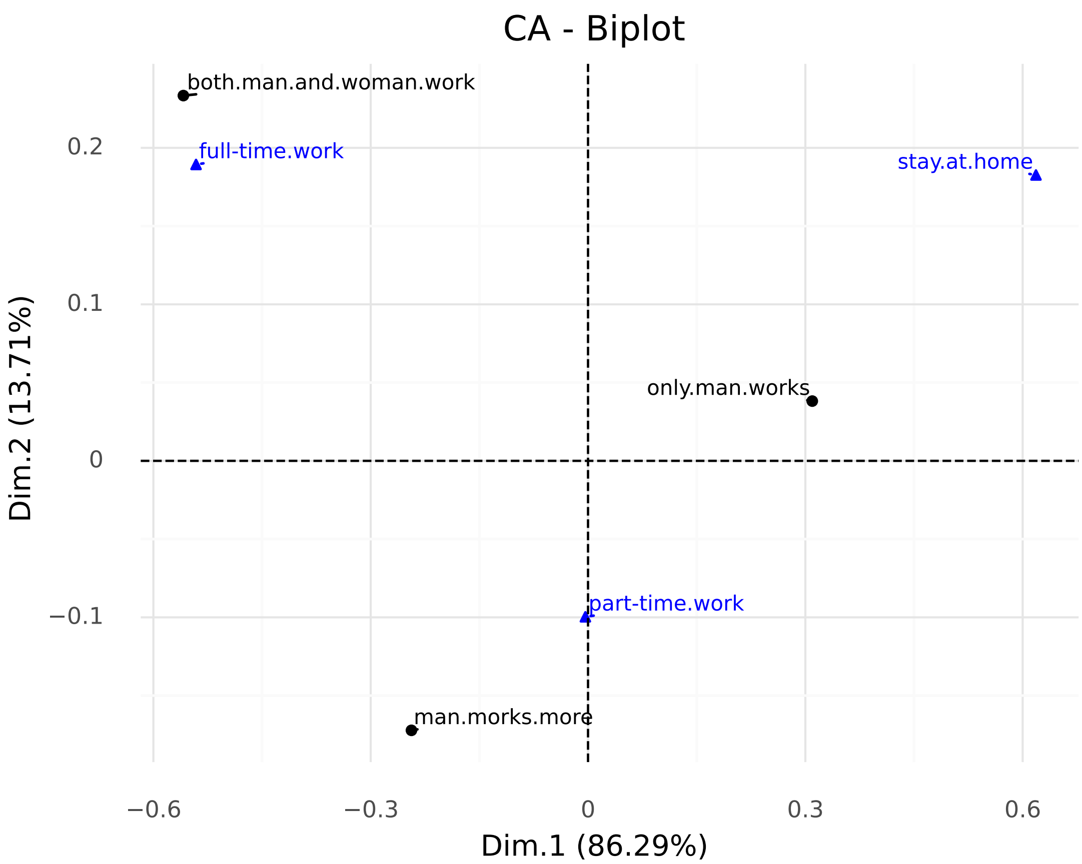
Le nuage des colonnes montre que le premier axe oppose « Stay at home » et « Full-time work », ce qui signifie qu'il oppose deux profils de femmes. Les femmes qui ont répondu « Stay at home » ont répondu « Only husband works » plus souvent que l'ensemble de la population et « Both husband and wife work » moins souvent que l'ensemble de la population.

De même, les femmes qui ont répondu « Full-time work » ont répondu « Only husband works » moins souvent que l'ensemble de la population et « Both husband and wife work » plus souvent que l'ensemble de la population. Le premier axe ordonne les modalités de la deuxième question de la moins à la plus en faveur du travail des femmes.

La même interprétation peut être faite pour le premier axe du nuage des lignes. Les modalités sont triées de la moins (« Only husband works ») à la plus (« Both husband and wife work ») en faveur du travail des femmes.

On peut représenter à la fois les lignes et les colonnes.

```
# Biplot
from scientisttools import fviz_ca_biplot
p = fviz_ca_biplot(my_ca)
print(p)
```



« Stay at home » est associé à « Only husband works » et peu associé aux deux autres modalités.

« Both husband and wife work » est associé à « Full-time work » et opposé à « Stay at home ».

Revenons un instant sur les données du tableau 2.1, issu de l'enquête de Nicole Tabard, croisant les deux variables qualitatives (questions) :

— Quelle est la famille idéale pour vous ?
— Quelle activité convient le mieux à une mère de famille quand ses enfants vont à l'école ?

Il est important de rappeler que les résultats de cette enquête ont été publiés en 1974. Il est fort à parier que la répartition des réponses serait totalement, si ce n'est en grande partie, différente aujourd'hui.

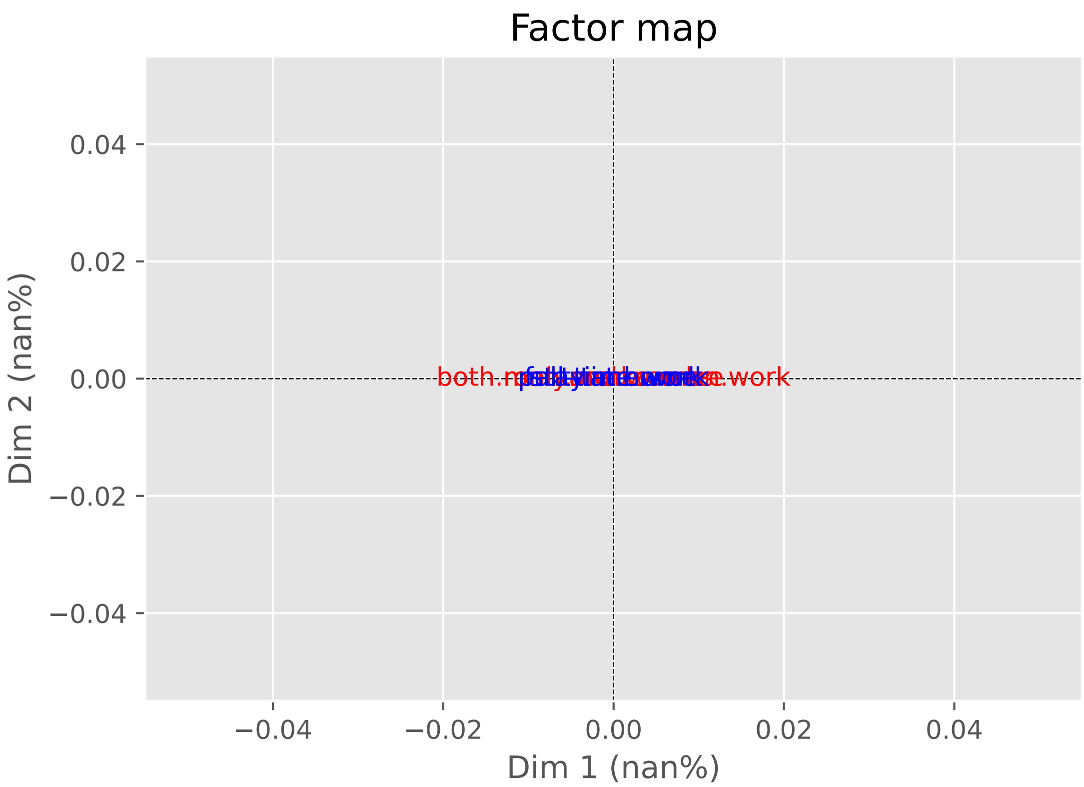
Lors d'une première lecture de ce tableau de contingence, François Husson soulève une apparente contradiction. À la question « Quelle est la famille idéale pour vous ? », nous voyons que 908 femmes sur 1724 (visible dans la marge colonne), soit environ 53% des répondantes, déclarent « Only man works » et seulement 261 femmes sur 1724 (environ 15%) déclarent « Both man and woman work ». Sur la base de ces premières réponses, nous pouvons émettre l'hypothèse, qu'à cette époque, une majorité était en faveur d'un modèle familial où seul le mari travaille.

À côté de ça, à la question « Quelle activité convient le mieux à une mère de famille quand ses enfants vont à l'école ? », elles sont 1440 sur 1724 (visible dans la marge ligne), soit environ 84%, à être en faveur du travail à mi-temps « Part time work » ou à plein-temps « Full time work ». Les réponses à cette question semblent indiquer que les femmes sont moins hostiles au travail féminin (bien au contraire).

Du coup, à ce stade de l'interprétation, nous nous retrouvons a priori face une contradiction. De cela, nous pouvons dire que le tableau de contingence ne permet pas de savoir si les femmes des années 70 sont favorables ou non à l'activité féminine. Par contre, Une première lecture du graphe de l'AFC nous permet de dire que les modalités des réponses s'associent entre elles des plus favorables au travail féminin aux plus défavorables au travail féminin.

Avant d'approfondir, plus en détail, l'interprétation de cette AFC, nous allons faire un pas de côté et voir ce qui se passe dans le cas où il y aurait indépendance entre les deux variables.

Si nous réalisons une AFC avec les données du modèle d'indépendance, on obtient la figure suivante :



La lecture de ce graphique nous permet de voir que les points sont quasiment tous confondus avec le centre de gravité, correspondant au profil moyen. La représentation graphique est trompeuse mais l'échelle des axes va dans le sens de notre interprétation. Simplement, ce qu'il y a retenir de ce graphe, c'est que, lorsqu'il y a indépendance entre les deux variables, tous les points sont confondus avec l'origine. Du fait qu'il n'y ait pas d'écarts à l'indépendance, il n'y a graphiquement rien à exploiter, rien à interpréter, rien à analyser. Ce graphe donne à voir ce que nous avons précédemment énoncé, à savoir que :

— Si nous acceptons l'hypothèse d'indépendance (p-value > 0.05 dans le cas d'un test du χ^2), nous n'aurons pas d'utilité à réaliser une AFC car les points projetés seront extrêmement proches ou confondus avec le centre de gravité, confondus avec le centre du graphe.

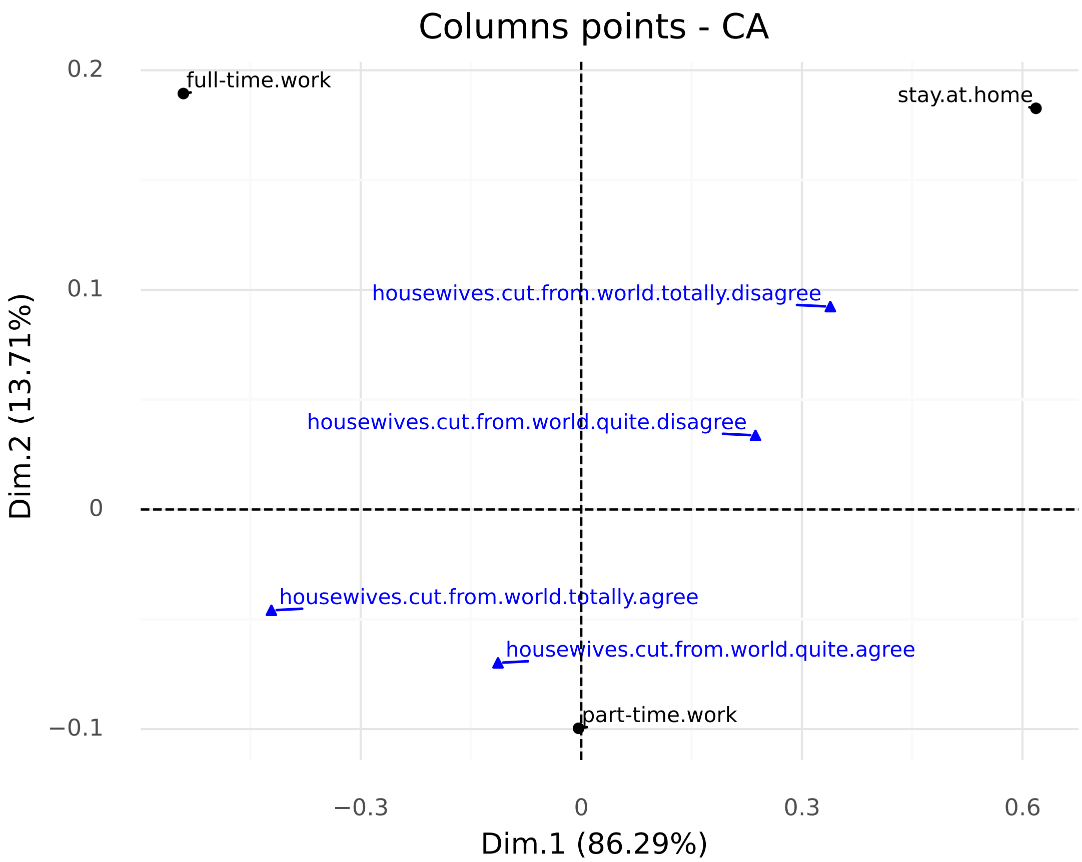
— La réalisation d'un test du χ^2 est donc fortement conseillée avant la réalisation d'une AFC.
— Plus précisément, le test du χ^2 conditionne l'éventuelle réalisation d'une AFC.

2.4 Addition de colonnes illustratives

On ajoute les colonnes qui correspondent à la troisième question en tant que variables illustratives. Tapez :

```
# Modèle avec colonnes supplémentaires
my_ca2 = CA(col_sup=[3,4,5,6]).fit(women_work)

# Carte de modalités colonnes
print(fviz_ca_col(my_ca2,repel=True))
```



« Totally agree » et « Quite agree » pour « Women who do not work feel cut off from the world » sont proches des modalités en faveur du travail des femmes.

« Quite disagree » et « Totally “disagree » sont proches des modalités opposées au travail des femmes.

Pour ajouter des points lignes illustratifs, utilisez l'argument suivant de la fonction PCA :

```
row_sup
```

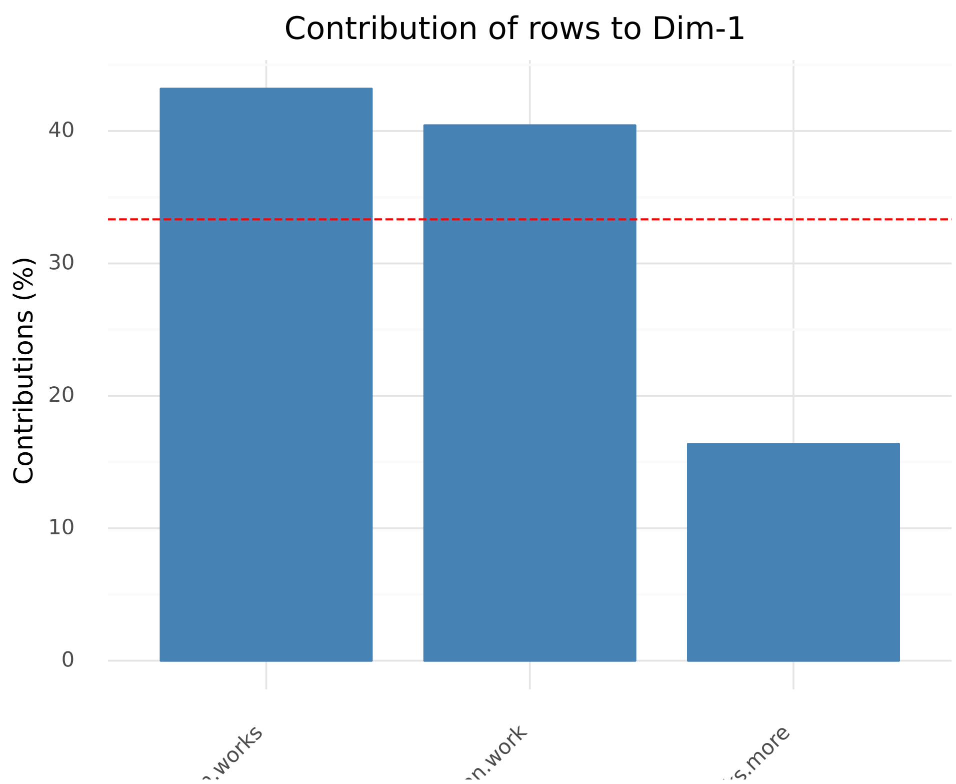
Tous les résultats détaillés peuvent être vus dans l'objet `my_pca2`. On peut récupérer les valeurs propres, les résultats des points lignes actifs et illustratifs, les résultats des points colonnes actifs et supplémentaires en tapant :

```
from scientisttools import get_ca_row,get_ca_col,get_eig
eig = get_eig(my_ca2)
row = get_ca_row(my_ca2)
col = get_ca_col(my_ca2)
```

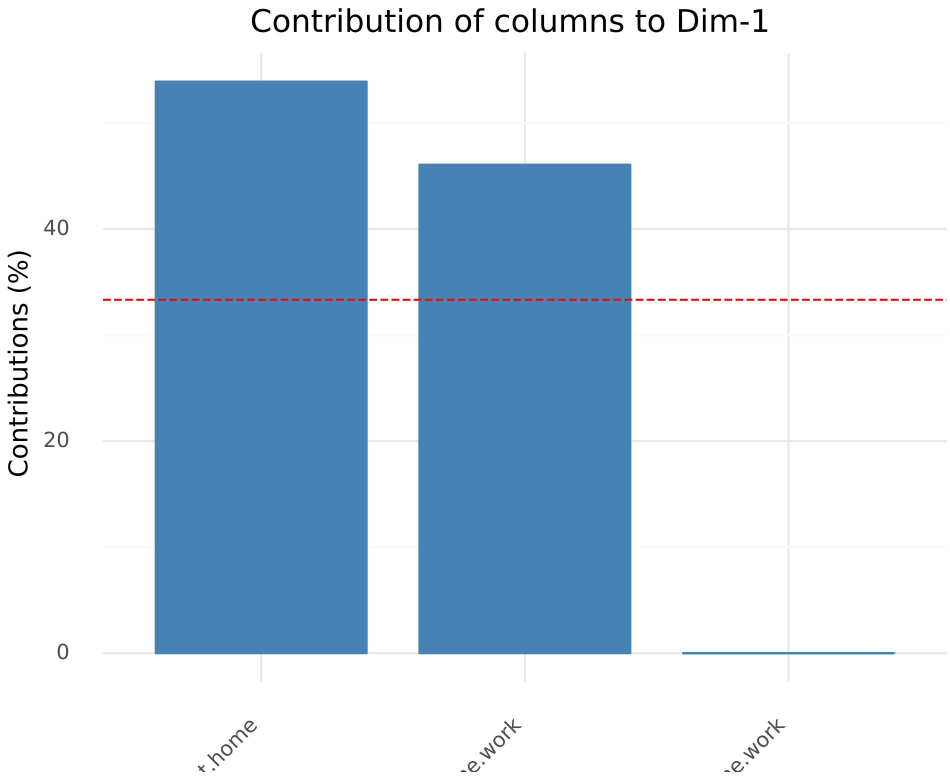
2.5 Interprétation des axes

Des graphiques qui permettent d'interpréter rapidement les axes : on choisit un axe factoriel (le 1er axe dans notre exemple) et on observe quels sont les points lignes et colonnes qui présentent les plus fortes contributions et cos2 pour cet axe.

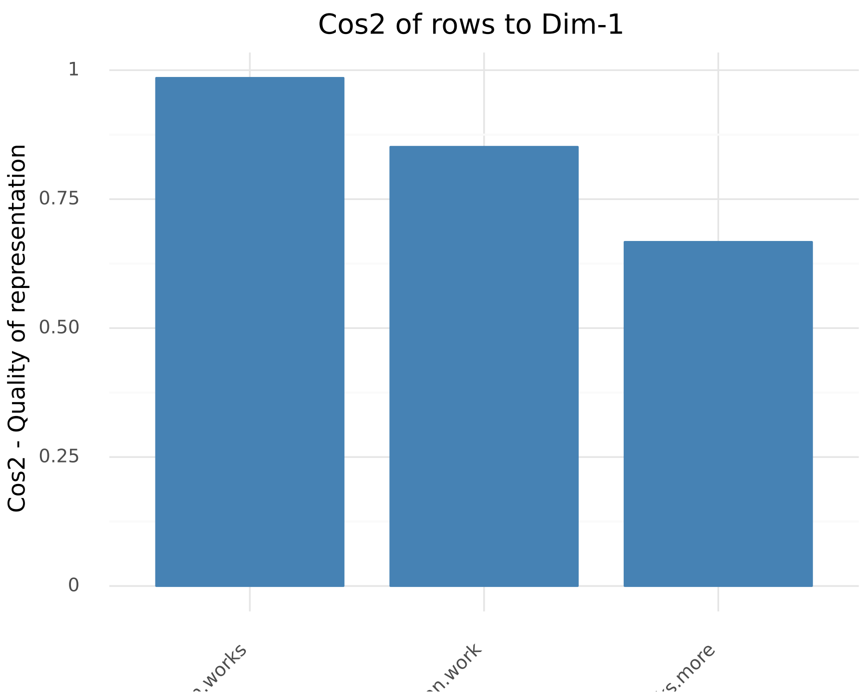
```
# Classement des points lignes en fonction de leur contribution au 1er axe
from scientisttools import fviz_contrib
p = fviz_contrib(my_ca,choice="row",axis=0)
print(p)
```



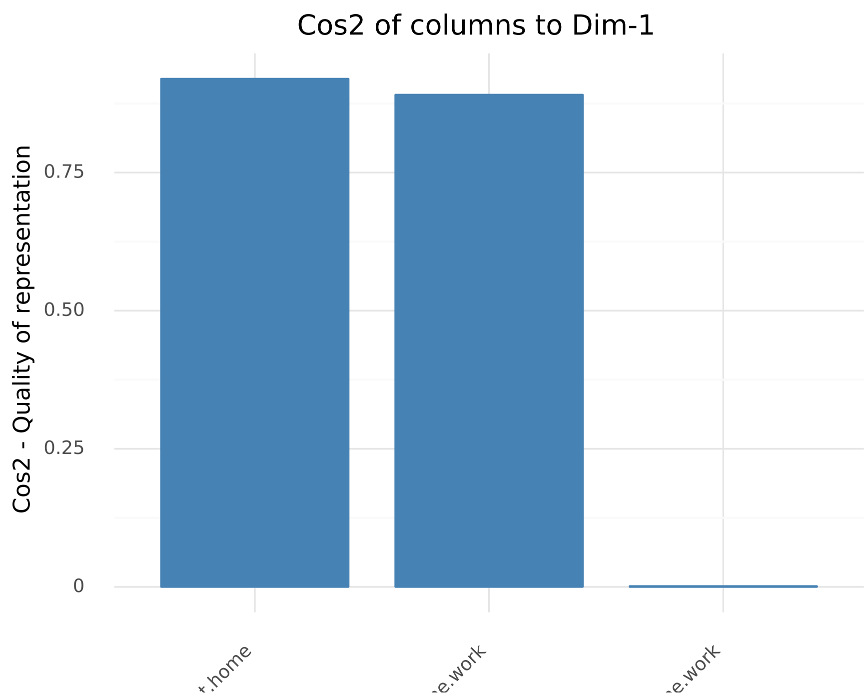

```
# Classement des points colonnes en fonction de leur contribution au 1er axe
p = fviz_contrib(my_ca,choice="col",axis=0)
print(p)
```




```
# Classement des points lignes en fonction de leur cos2 sur le 1er axe
from scientisttools import fviz_cos2
p = fviz_cos2(my_ca,choice="row")
print(p)
```



```
# Classement des points colonnes en fonction de leur cos2 sur le 1er axe
p = fviz_cos2(my_ca,choice="col")
print(p)
```



2.6 Description des dimensions

On peut décrire les dimensions données par les lignes ou les colonnes.

```
from scientisttools import dimdesc
dim_desc = dimdesc(my_ca)
dim_desc.keys()
```

```
## dict_keys(['Dim.1', 'Dim.2'])
```

```
dim_desc["Dim.1"]["row"]
```

```
##                              coord
## both.man.and.woman.work -0.558605
## man.morks.more          -0.243759
## only.man.works           0.309562
```

```
dim_desc["Dim.1"]["col"]
```

```
##                    coord
## full-time.work -0.541113
## part-time.work -0.003638
## stay.at.home    0.618376
```

3 Analyse (Factorielle) des Correspondances Multiples

Sommaire

Ce chapitre a pour objectif de présenter rapidement les principales fonctionnalités offertes par le package « scientisttools » pour réaliser une Analyse des Correspondances Multiples.

3.1 Présentation des données

Nous illustrons l'analyse des correspondances multiples à l'aide d'un exemple sur les données « Races Canines » extraites de l'ouvrage de Tenenhaus.

```
# Chargement des données
import pandas as pd
# Données actives
A = pd.read_excel("./donnee/races_canines_acm.xls",header=0,sheet_name=0,index_col=0)
# Individus supplémentaires
B = pd.read_excel("./donnee/races_canines_acm.xls",header=0,sheet_name=1,index_col=0)
# Variables qualitative supplémentaires
C = pd.read_excel("./donnee/races_canines_acm.xls",header=0,sheet_name=2,index_col=0)
# Variables quantitatives supplémentaires
D = pd.read_excel("./donnee/races_canines_acm.xls",header=0,sheet_name=3,index_col=0)
C.index = D.index = A.index
# Concaténation
Data = pd.concat([pd.concat([A,B],axis=0),C,D],axis=1)
Data.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 33 entries, Beauceron to Wisky
## Data columns (total 8 columns):
```

```
## #   Column        Non-Null Count  Dtype
## --- ------        --------------  -----
## 0   Taille        33 non-null     object
## 1   Poids         33 non-null     object
## 2   Velocite      33 non-null     object
## 3   Intelligence  33 non-null     object
## 4   Affection     33 non-null     object
## 5   Agressivite   33 non-null     object
## 6   Fonction      27 non-null     object
## 7   Cote          27 non-null     float64
## dtypes: float64(1), object(7)
## memory usage: 3.4+ KB
```

Ces données décrivent les caractéristiques de 27 races de chiens au moyen de variables qualitatives.

La première colonne du tableau 3.1 correspond à l'identifiant des observations. Les 6 premières variables sont considérés comme actives : Taille, Poids, Vélocité, Intelligence, Affection, Agressivité. La $7^{\text{ème}}$ variable « Fonction » est considérée comme variable illustrative qualitative tandis que la $8^{\text{ème}}$ comme variable illustrative quantitative. Les modalités des différentes variables sont les suivantes :

— Taille, Poids, vélocité, intelligence : faible (-), moyenn (+), fort (++)
— Affection, agressivité : faible (-), fort(+)
— fonction : compagnie, chasse, utilité.

La variable cote est une variable que nous avons pris soins de créer afin d'illustrer le concept de variable illustrative quantitative en ACM.

Les principales questions auxquelles nous nous posons sont les suivantes :

— Quels sont les chiens qui se ressemblent ? Quels sont les chiens qui sont dissemblables ? (proximité entre les individus)
— Sur quels caractères sont fondées ces ressemblances/dissemblances ?
— Quelles sont les associations entre les modalités ? Par exemple, un animal de grande taille est - il plus agressif ou moins agressif ?
— Quelles sont les relations entre les variables ? Par exemple y a-t-il une relation entre la taille et l'agressivité ou bien sont - ce des caractères orthogonaux.

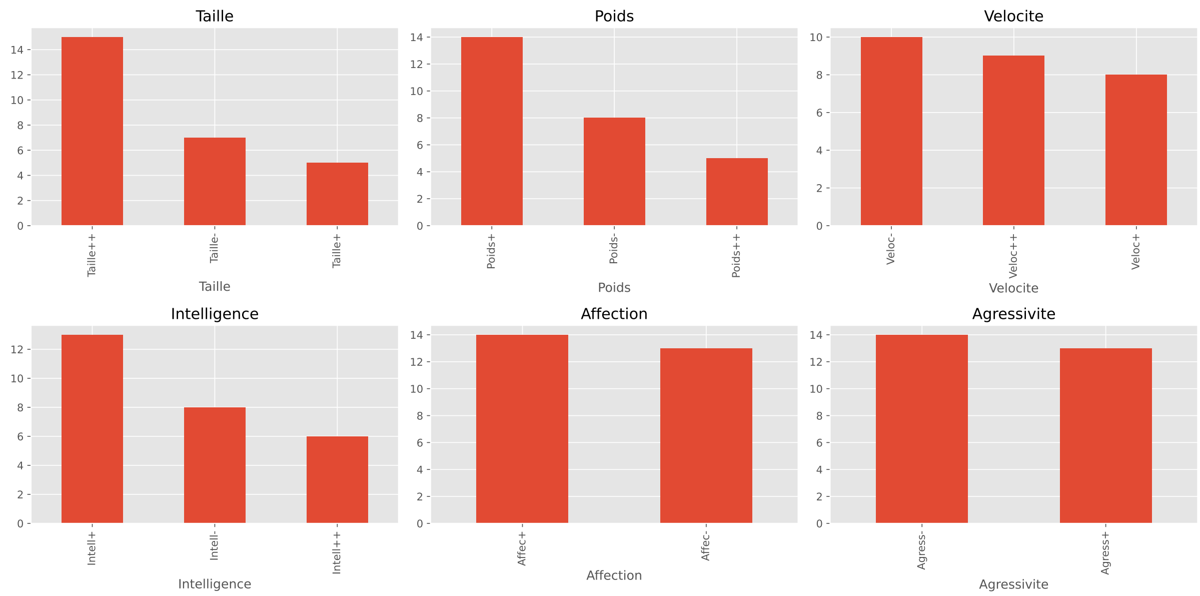
A partir du tableau 3.1, on remarque que les paires de chiens (Bull - Dog, Teckel), (Chihuahua, Pékinois) et (Dalmatien, Labrador) sont des valeurs identiques pour les 7 variables, il y aura donc des observations confondues.

A l'aide d'un diagramme à barres, nous visualisons nos différentes variables :

```python
# Diagramme à barres
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(16,8))
for i, name in enumerate(A.columns):
  ax = fig.add_subplot(2,3,i+1)
  A[name].value_counts().plot.bar(ax=ax)
  ax.set(title=name)
  ax.grid(visible=True)
  plt.tight_layout()
```

Table 3.1 – Données Races Canines

	Taille	Poids	Velocite	Intelligence	Affection	Agressivite	Fonction	Cote
Beauceron	Taille++	Poids+	Veloc++	Intell+	Affec+	Agress+	utilite	2.5
Basset	Taille-	Poids-	Veloc-	Intell-	Affec-	Agress+	chasse	4.5
Berger All	Taille++	Poids+	Veloc++	Intell++	Affec+	Agress+	utilite	3.0
Boxer	Taille+	Poids+	Veloc+	Intell+	Affec+	Agress+	compagnie	2.0
Bull-Dog	Taille-	Poids-	Veloc-	Intell+	Affec+	Agress-	compagnie	4.5
Bull-Mastif	Taille++	Poids++	Veloc-	Intell++	Affec-	Agress+	utilite	4.0
Caniche	Taille-	Poids-	Veloc+	Intell++	Affec+	Agress-	compagnie	2.0
Chihuahua	Taille-	Poids-	Veloc-	Intell-	Affec+	Agress-	compagnie	3.5
Cocker	Taille+	Poids-	Veloc-	Intell+	Affec+	Agress+	compagnie	4.5
Colley	Taille++	Poids+	Veloc++	Intell+	Affec+	Agress-	compagnie	2.0
Dalmatien	Taille+	Poids+	Veloc+	Intell+	Affec+	Agress-	compagnie	2.5
Doberman	Taille++	Poids+	Veloc++	Intell++	Affec-	Agress+	utilite	3.0
Dogue All	Taille++	Poids++	Veloc++	Intell-	Affec-	Agress+	utilite	4.0
Epag. Breton	Taille+	Poids+	Veloc+	Intell++	Affec+	Agress-	chasse	3.5
Epag. Français	Taille++	Poids+	Veloc+	Intell+	Affec-	Agress-	chasse	2.5
Fox-Hound	Taille++	Poids+	Veloc++	Intell-	Affec-	Agress+	chasse	3.0
Fox-Terrier	Taille-	Poids-	Veloc+	Intell+	Affec+	Agress+	compagnie	4.5
Gd Bleu Gasc	Taille++	Poids+	Veloc+	Intell-	Affec-	Agress+	chasse	3.5
Labrador	Taille+	Poids+	Veloc+	Intell+	Affec+	Agress-	chasse	2.0
Levrier	Taille++	Poids+	Veloc++	Intell-	Affec-	Agress-	chasse	2.5
Mastiff	Taille++	Poids++	Veloc-	Intell-	Affec-	Agress+	utilite	4.0
Pekinois	Taille-	Poids-	Veloc-	Intell-	Affec+	Agress-	compagnie	3.0
Pointer	Taille++	Poids+	Veloc++	Intell++	Affec-	Agress-	chasse	3.5
St-Bernard	Taille++	Poids++	Veloc-	Intell+	Affec-	Agress+	utilite	4.5
Setter	Taille++	Poids+	Veloc++	Intell+	Affec-	Agress-	chasse	2.0
Teckel	Taille-	Poids-	Veloc-	Intell+	Affec+	Agress-	compagnie	2.5
Terre-Neuve	Taille++	Poids++	Veloc-	Intell+	Affec-	Agress-	utilite	3.0
Medor	Taille+	Poids-	Veloc-	Intell++	Affec-	Agress+	NA	NaN
Djeck	Taille++	Poids++	Veloc+	Intell+	Affec+	Agress-	NA	NaN
Taico	Taille-	Poids+	Veloc++	Intell++	Affec+	Agress+	NA	NaN
Rocky	Taille+	Poids+	Veloc+	Intell-	Affec+	Agress-	NA	NaN
Boudog	Taille-	Poids-	Veloc++	Intell+	Affec-	Agress+	NA	NaN
Wisky	Taille+	Poids++	Veloc-	Intell-	Affec+	Agress+	NA	NaN



3.2 ACM

3.2.1 Objectifs

L'objectif est de trouver un système de représentation (répère factoriel) qui préserve au mieux les distances entre les individus, qui permet de discerner le mieux possible les individus entre eux, qui maximise les (le carré des) écarts à l'origine.

3.2.2 Chargement de scientisttools

```
from scientisttools import MCA
```

3.2.3 Individus et variables actifs

On crée une instance de la classe MCA, en lui passant ici des étiquettes pour les lignes et les variables. Ces paramètres sont facultatifs ; en leur absence, le programme détermine automatiquement des étiquettes.

```
# Instanciation
my_mca = MCA()
```

On estime le modèle en appliquant la méthode `fit` de la classe MCA sur le jeu de données.

```
# Estimation du modèle
my_mca.fit(A)
```

```
## MCA()
```

3.2.3.1 Les valeurs propres

L'exécution de la méthode `my_mca.fit(A)` provoque le calcul des attributs parmi lesquels `my_mca.eig_` pour les valeurs propres.

```
# Valeurs propres
print(my_mca.eig_)
```

```
##          eigenvalue  difference  proportion  cumulative
## Dim.1      0.481606    0.096869   28.896370   28.896370
## Dim.2      0.384737    0.173783   23.084237   51.980607
## Dim.3      0.210954    0.053400   12.657243   64.637850
## Dim.4      0.157554    0.007421    9.453242   74.091092
## Dim.5      0.150133    0.026837    9.007960   83.099052
## Dim.6      0.123295    0.041833    7.397718   90.496770
## Dim.7      0.081462    0.035793    4.887748   95.384518
## Dim.8      0.045670    0.022128    2.740185   98.124703
## Dim.9      0.023542    0.015829    1.412515   99.537218
## Dim.10     0.007713         NaN    0.462782  100.000000
```

L'attribut `my_mca.eig_` contient :

- en 1ère ligne : les valeurs propres en valeur absolue
- en 2ème ligne : les différences des valeurs propres
- en 3ème ligne : les valeurs propres en pourcentage de la variance totale (proportions)
- en 4ème ligne : les valeurs propres en pourcentage cumulé de la variance totale.

La fonction `get_eig` retourne les valeurs propres sous forme de tableau de données.

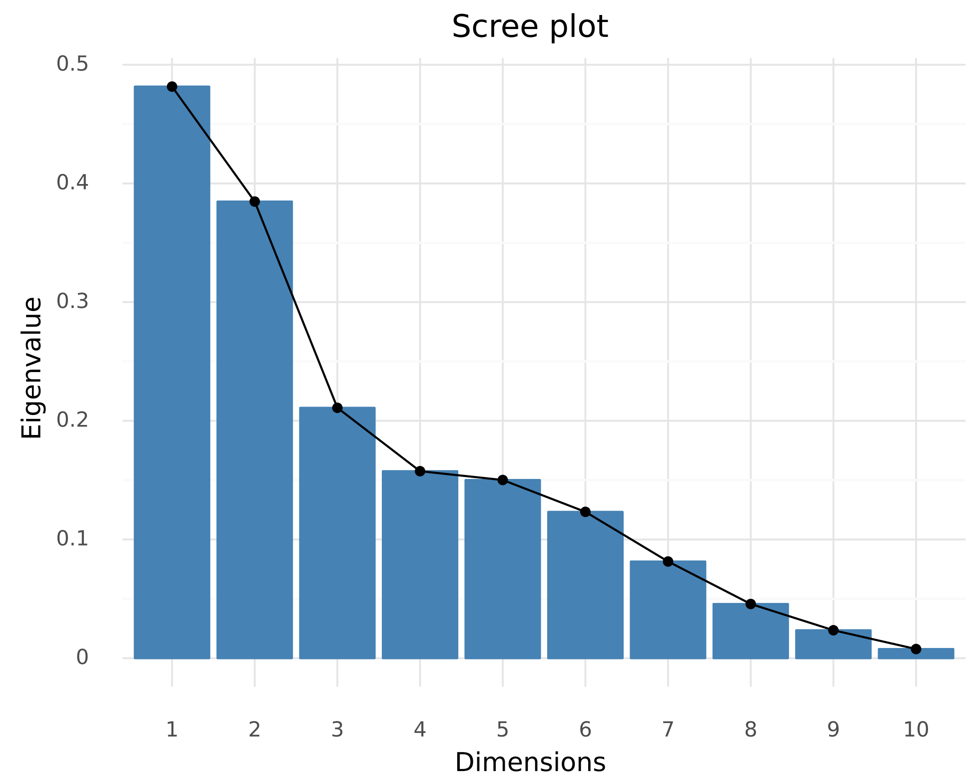
```
# Valeurs propres
from scientisttools import get_eig
print(get_eig(my_mca))
```

```
##          eigenvalue  difference  proportion  cumulative
## Dim.1      0.481606    0.096869   28.896370   28.896370
## Dim.2      0.384737    0.173783   23.084237   51.980607
## Dim.3      0.210954    0.053400   12.657243   64.637850
## Dim.4      0.157554    0.007421    9.453242   74.091092
## Dim.5      0.150133    0.026837    9.007960   83.099052
## Dim.6      0.123295    0.041833    7.397718   90.496770
## Dim.7      0.081462    0.035793    4.887748   95.384518
## Dim.8      0.045670    0.022128    2.740185   98.124703
## Dim.9      0.023542    0.015829    1.412515   99.537218
## Dim.10     0.007713         NaN    0.462782  100.000000
```

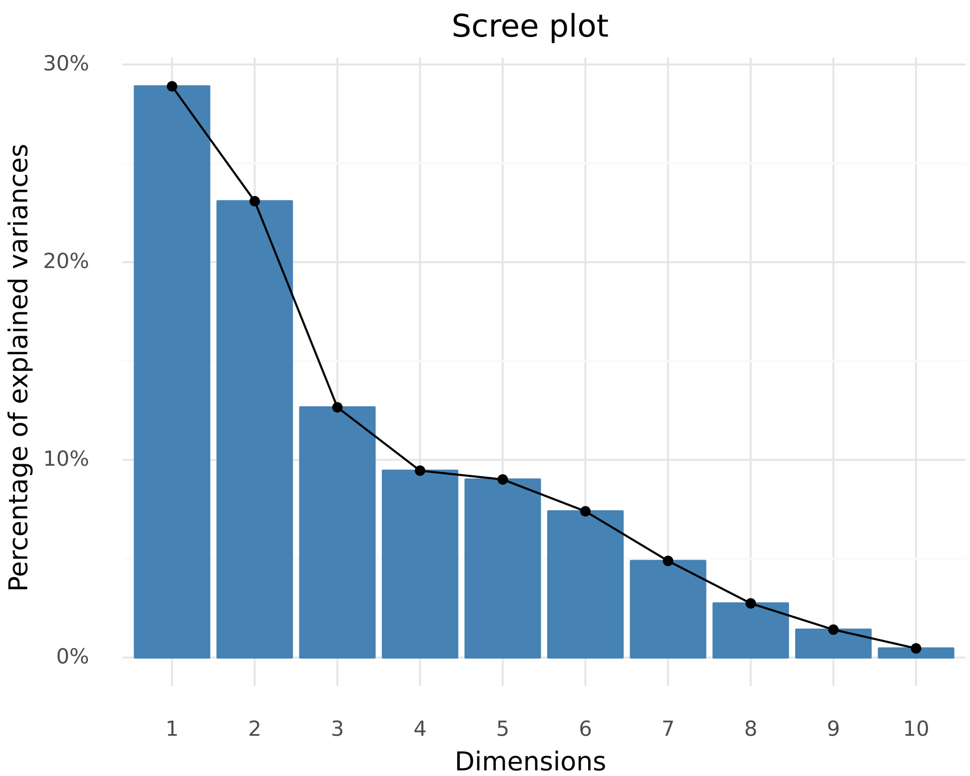
Le nombre de modalités actives est $16(3 \times 4 + 2 \times 2)$, ce qui conduit à 10 facteurs et à une inertie totale de $\dfrac{16}{6} - 1 = \dfrac{5}{3} = 1.667$.

Les valeurs propres peuvent être représentées graphiquement

```
from scientisttools import fviz_eig
print(fviz_eig(my_mca,choice="eigenvalue"))
```




```
print(fviz_eig(my_mca,choice="proportion"))
```



Le critère de Kaiser conduit à ne retenir que trois axes, le diagramme des valeurs propres montre cependant une chute après λ_2. On interprètera donc uniquement les deux premiers axes.

3.2.3.2 Correction de Benzécri

La correction de Benzécri s'appuie sur l'idée qu'une partie de l'information est rédondante dans les données présentées à l'algorithme de l'ACM.

```
# Correction de Benzécri
my_mca.benzecri_correction_
```

```
##         eigenvalue  proportion  cumulative
## Dim.1     0.142829   66.701311   66.701311
## Dim.2     0.068479   31.979703   98.681014
## Dim.3     0.002824    1.318986  100.000000
```

3.2.3.3 Correction de Greenacre

La correction de Greenacre s'appuie sur la correction de Benzécri mais reconsidère la proportion d'inertie portée par les facteurs. Une partie de l'information est triviale dans le tableau de Burt, il s'agit du croisement endogène de chaque variable.

```
# Correction de Greenacre
my_mca.greenacre_correction_
```

```
##         eigenvalue  proportion  cumulative
## Dim.1     0.142829   54.450544   54.450544
## Dim.2     0.068479   26.106117   80.556662
## Dim.3     0.002824    1.076733   81.633395
```

On peut obtenir un résumé des principaux résultats en utilisant la fonction `summaryMCA`.

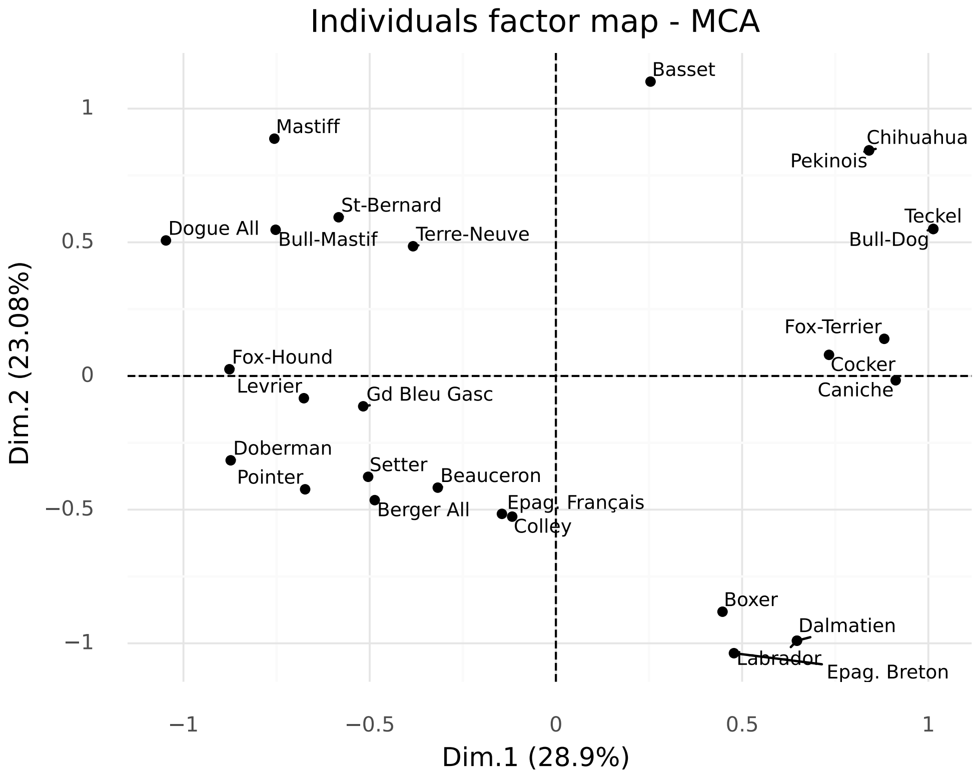
```
from scientisttools import summaryMCA
summaryMCA(my_mca)
```

```
##                      Multiple Correspondance Analysis - Results
##
## Importance of components
##                              Dim.1   Dim.2   Dim.3  ...   Dim.8   Dim.9   Dim.10
## Variance                     0.482   0.385   0.211  ...   0.046   0.024    0.008
## Difference                   0.097   0.174   0.053  ...   0.022   0.016      NaN
## % of var.                   28.896  23.084  12.657  ...   2.740   1.413    0.463
## Cumulative of % of var.     28.896  51.981  64.638  ...  98.125  99.537  100.000
##
## [4 rows x 10 columns]
##
## Individuals (the 10 first)
```

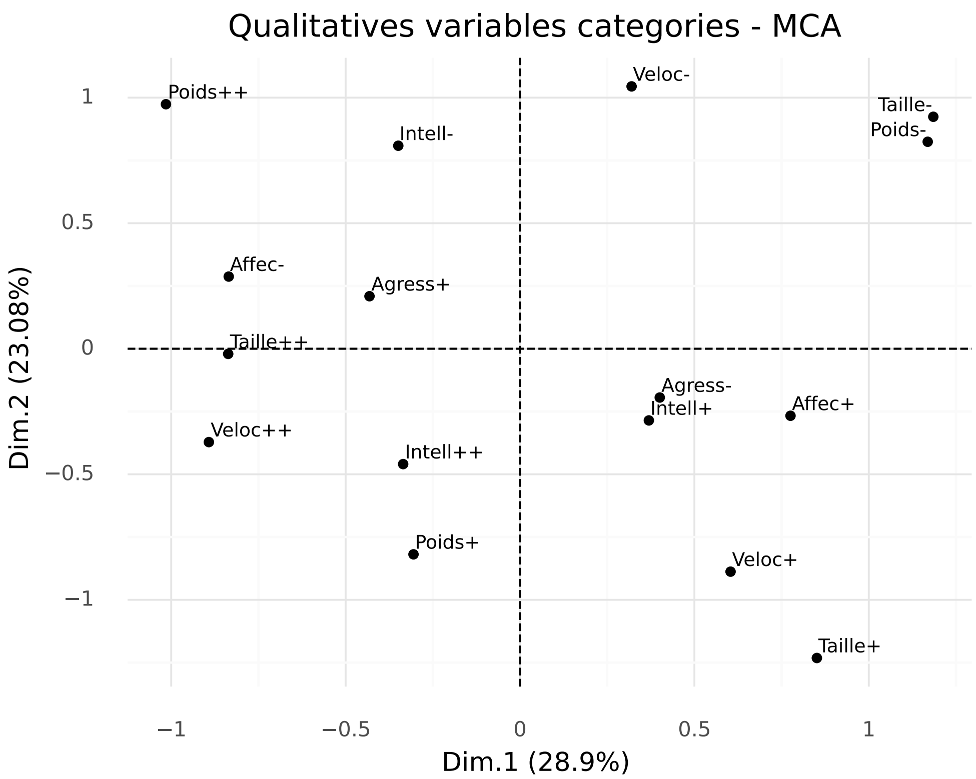
```
##
##                dist  weight  inertia  Dim.1  ...   cos2  Dim.3    ctr   cos2
## Beauceron     1.065   0.037    0.042 -0.317  ...  0.154 -0.101  0.181  0.009
## Basset        1.382   0.037    0.071  0.254  ...  0.635 -0.191  0.638  0.019
## Berger All    1.241   0.037    0.057 -0.486  ...  0.140 -0.498  4.357  0.161
## Boxer         1.341   0.037    0.067  0.447  ...  0.433  0.692  8.408  0.266
## Bull-Dog      1.282   0.037    0.061  1.013  ...  0.184 -0.163  0.469  0.016
## Bull-Mastif   1.446   0.037    0.077 -0.753  ...  0.143  0.498  4.347  0.118
## Caniche       1.470   0.037    0.080  0.912  ...  0.000 -0.577  5.836  0.154
## Chihuahua     1.364   0.037    0.069  0.841  ...  0.383 -0.470  3.877  0.119
## Cocker        1.388   0.037    0.071  0.733  ...  0.003  0.662  7.700  0.228
## Colley        1.054   0.037    0.041 -0.117  ...  0.249 -0.335  1.969  0.101
##
## [10 rows x 12 columns]
##
## Categories (the 10 first)
##
##             dist  weight  inertia  Dim.1  ...  Dim.3     ctr   cos2  vtest
## Taille+    2.098   0.031    0.136  0.851  ...  1.016  15.104  0.235  2.470
## Taille++   0.894   0.093    0.074 -0.837  ... -0.051   0.115  0.003 -0.292
## Taille-    1.690   0.043    0.123  1.185  ... -0.616   7.772  0.133 -1.858
## Poids+     0.964   0.086    0.080 -0.305  ... -0.231   2.191  0.058 -1.224
## Poids++    2.098   0.031    0.136 -1.015  ...  1.222  21.833  0.339  2.970
## Poids-     1.541   0.049    0.117  1.169  ... -0.359   3.013  0.054 -1.187
## Veloc+     1.541   0.049    0.117  0.604  ...  0.356   2.972  0.053  1.179
## Veloc++    1.414   0.056    0.111 -0.892  ... -0.763  15.335  0.291 -2.751
## Veloc-     1.304   0.062    0.105  0.320  ...  0.402   4.722  0.095  1.571
## Intell+    1.038   0.080    0.086  0.369  ...  0.493   9.253  0.226  2.423
##
## [10 rows x 15 columns]
##
## Categorical variables (eta2)
##
##                inertia  Dim.1     ctr  Dim.2     ctr  Dim.3     ctr
## Taille           0.074  0.887  30.698  0.502  21.767  0.291  22.992
## Poids            0.074  0.644  22.288  0.725  31.393  0.342  27.038
## Velocite         0.074  0.411  14.229  0.684  29.631  0.291  23.030
## Intelligence     0.074  0.127   4.387  0.280  12.124  0.234  18.465
## Affection        0.037  0.648  22.413  0.077   3.324  0.004   0.314
## Agressivite      0.037  0.173   5.984  0.041   1.760  0.103   8.162
```

3.2.3.4 Représentation graphique

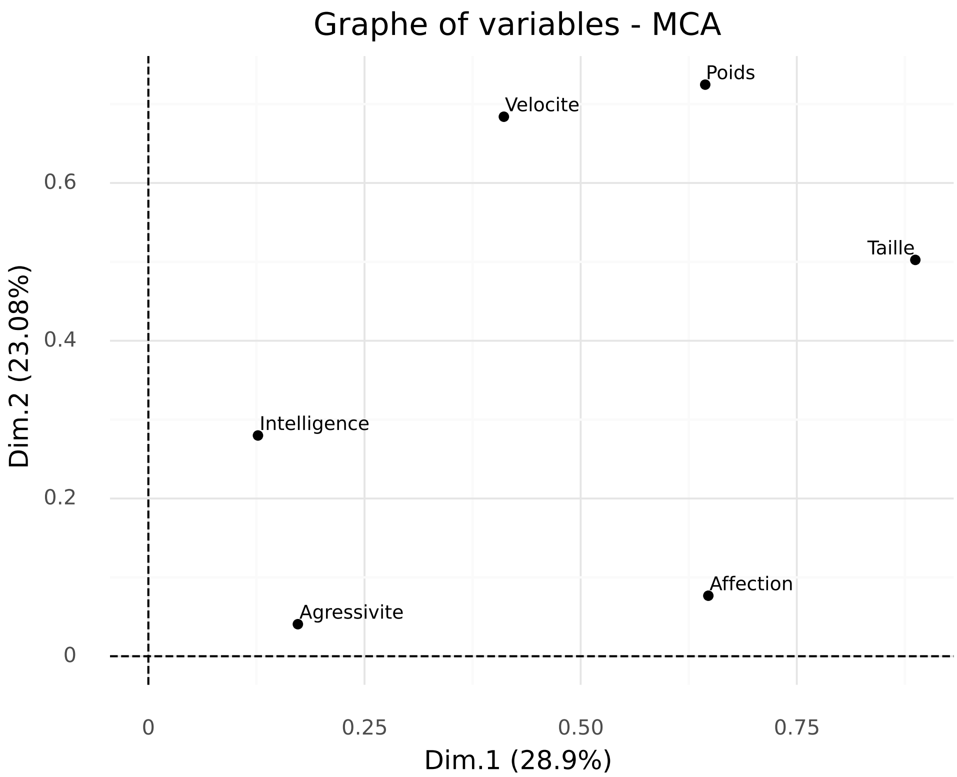
```python
# Carte des individus
from scientisttools import fviz_mca_ind
print(fviz_mca_ind(my_mca,repel=True))
```




```python
# Carte des modalités
from scientisttools import fviz_mca_mod
print(fviz_mca_mod(my_mca,repel=True))
```



```python
# Carte des variables
from scientisttools import fviz_mca_var
print(fviz_mca_var(my_mca,repel=True))
```



3.2.4 ACM avec les éléments supplémentaires

Les individus illustratifs et les variables illustratives n'influencent pas la construction des composantes principales de l'analyse. Ils/Elles aident à l'interprétation des dimensions de variabilité.

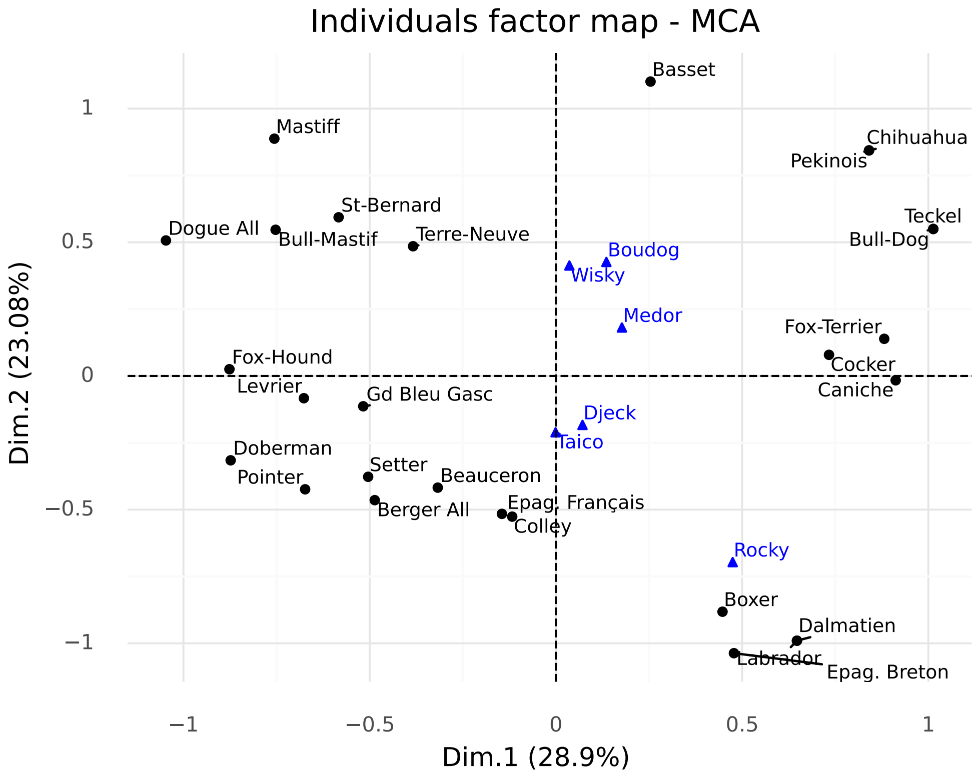
On peut ajouter deux types de variables : continues et qualitatives.

On ajoute la variable « Cote » comme variable continue illustrative quantitative et « Fonction » comme variable qualitative. Tapez la ligne de code suivante :

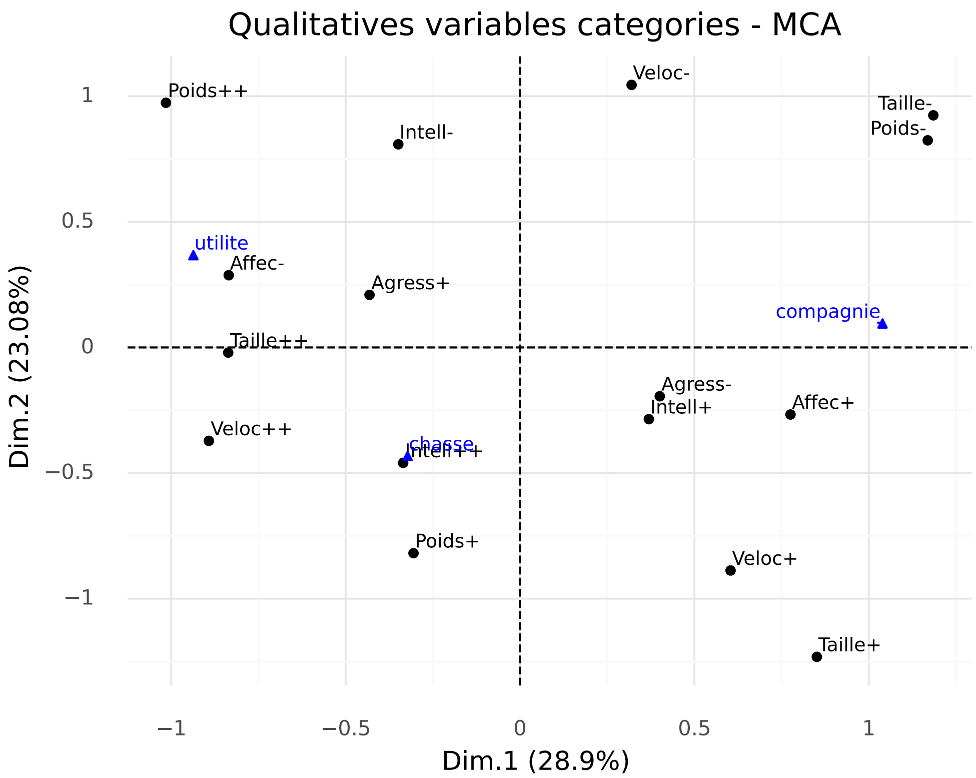
```python
# ACM avec les éléments supplémentaires
my_mca2 = MCA(ind_sup=list(range(27,33)),quali_sup=6,quanti_sup=7)
# Estimation
my_mca2.fit(Data)
```

```
## MCA(ind_sup=[27, 28, 29, 30, 31, 32], quali_sup=6, quanti_sup=7)
```

```python
# Carte des individus
print(fviz_mca_ind(my_mca2,repel=True))
```



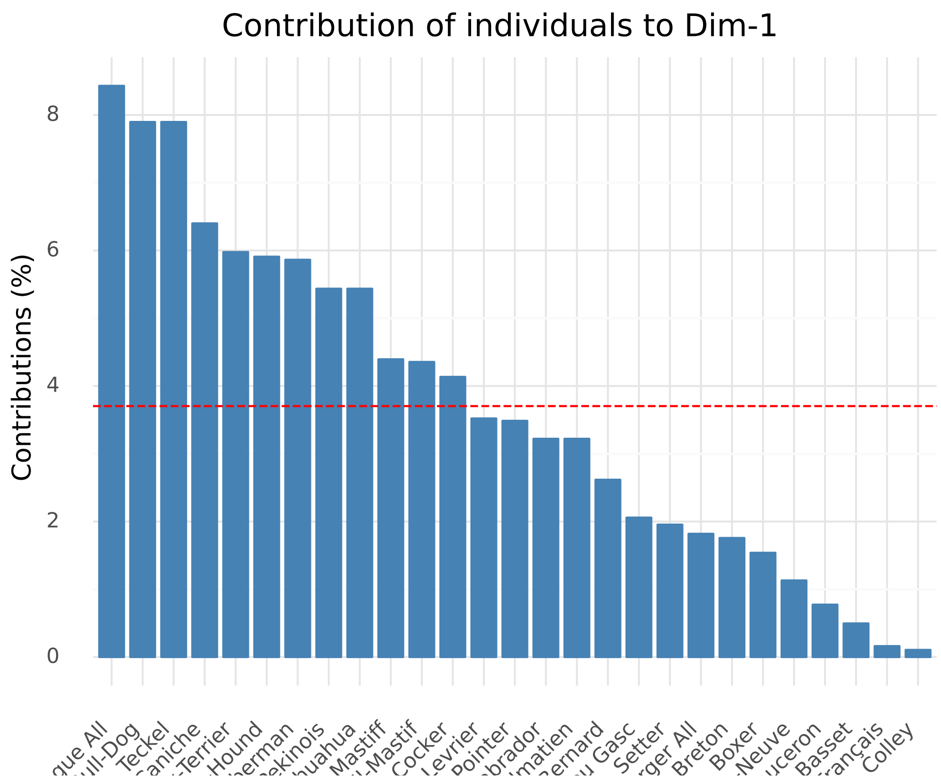

```
# Carte des modalités
print(fviz_mca_mod(my_mca2,repel=True))
```



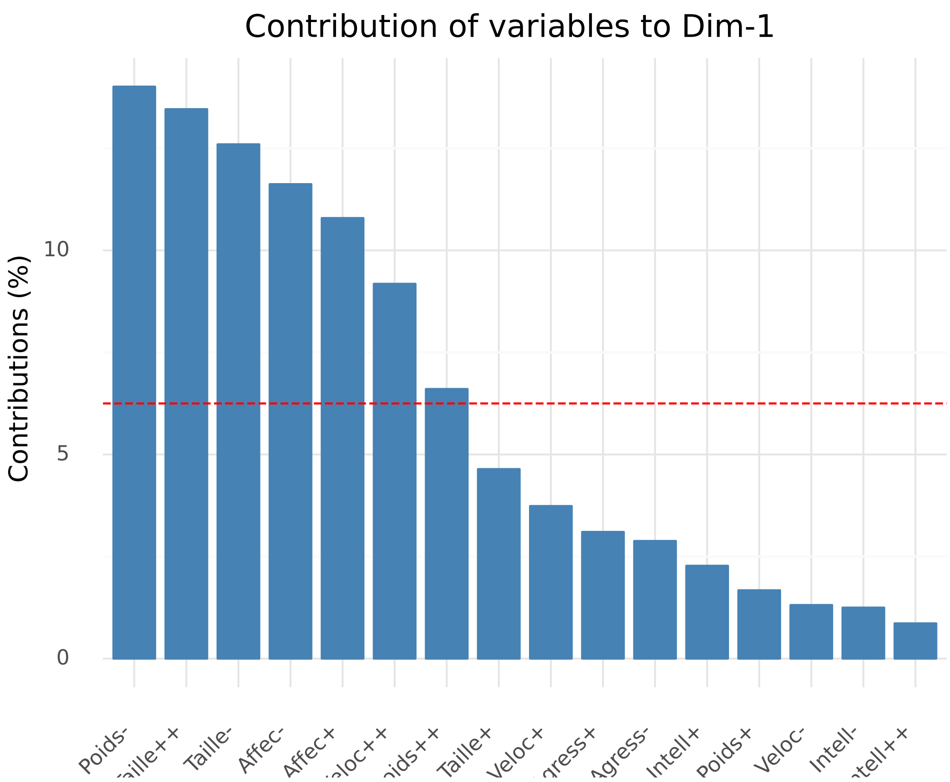
3.3 Interprétation des axes

Des graphiques qui permettent d'interpréter rapidement les axes : on choisit un axe factoriel (le 1er axe dans notre exemple) et on observe quels sont les points lignes et colonnes qui présentent les plus fortes contributions et cos2 pour cet axe.

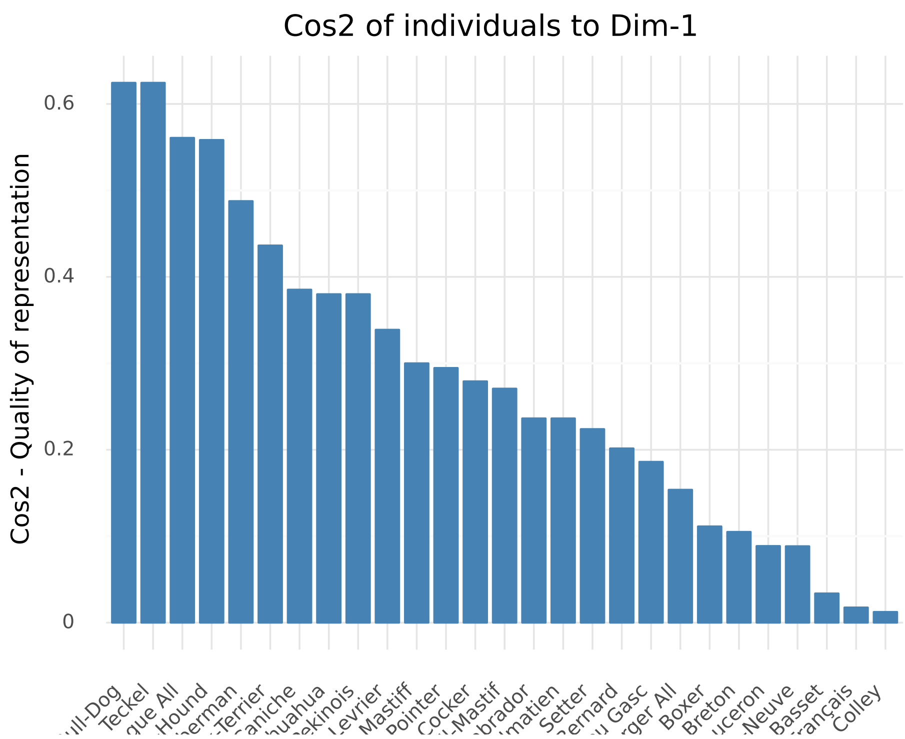
```
# Classement des points lignes en fonction de leur contribution au 1er axe
from scientisttools import fviz_contrib, fviz_cos2
p = fviz_contrib(my_mca2,choice="ind")
print(p)
```



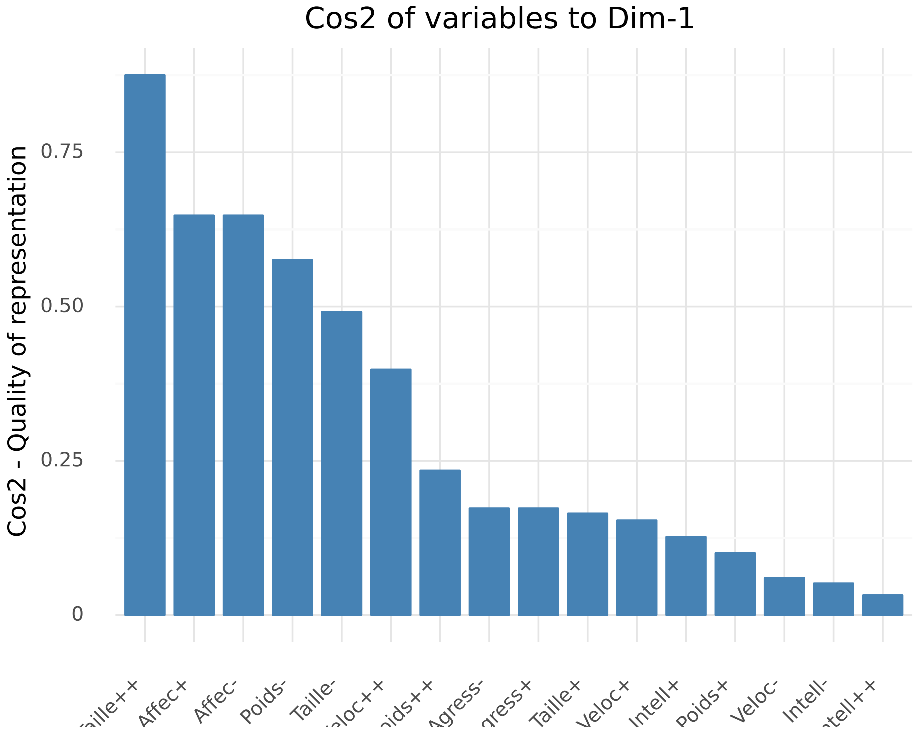

```
# Classement des modalités en fonction de leur contribution au 1er axe
p = fviz_contrib(my_mca2,choice="var")
print(p)
```




```
# Classement des individus en fonction de leur cos2 sur le 1er axe
p = fviz_cos2(my_mca2)
print(p)
```



```
# Classement des modalités en fonction de leur cos2 sur le 1er axe
p = fviz_cos2(my_mca2,choice = "var")
print(p)
```



3.4 Description des axes

On peut décrire les dimensions données par les variables en calculant le ratio de corrélation entre une variable et une dimension et en réalisant un test de significativité.

```
from scientisttools import dimdesc
dim_desc = dimdesc(my_mca2)
dim_desc.keys()
```

```
## dict_keys(['Dim.1', 'Dim.2', 'Dim.3', 'Dim.4', 'Dim.5', 'Dim.6', 'Dim.7', 'Dim.8', 'Dim.9
```

```
dim_desc["Dim.1"]
```

```
##              Sum. Intra  Sum. Inter        R2    F-stats        pvalue
## Taille         1.468427   11.534939  0.887073  94.263625  0.000000e+00
## Fonction       3.971435    9.031931  0.694584  27.290680  6.600000e-07
## Affection      4.581660    8.421706  0.647656  45.953357  4.200000e-07
## Poids          4.628594    8.374772  0.644046  21.712265  4.140000e-06
## Velocite       7.656719    5.346647  0.411174   8.379538  1.737170e-03
## Agressivite   10.754775    2.248591  0.172924   5.226960  3.098130e-02
```

```
dim_desc["Dim.2"]["quali"]
```

```
##              Sum. Intra  Sum. Inter        R2    F-stats        pvalue
## Poids          2.859918    7.527989  0.724688  31.586870  1.900000e-07
## Velocite       3.282502    7.105405  0.684007  25.975569  9.900000e-07
## Taille         5.168133    5.219774  0.502486  12.119907  2.299700e-04
## Intelligence   7.480643    2.907264  0.279870   4.663660  1.945048e-02
```

```
dim_desc["Dim.2"]["quanti"]
```

```
##       statistic    pvalue
## Cote   0.601291  0.000909
```

3.5 Approche Machine Learning

Ici, l'objectif est d'utiliser l'Analyse des Correspondances Multiples en tant que méthode de prétraitement.

La classe MCA implémente les méthodes `fit`, `transform` et `fit_transform` bien connues des utilisateurs de scikit-learn.

```
my_mca.transform(A).iloc[:5,:]
```

```
##                 Dim.1     Dim.2     Dim.3  ...     Dim.8     Dim.9    Dim.10
## Beauceron   -0.317200 -0.417701 -0.101468  ...  0.201986 -0.167019  0.022807
## Basset       0.254110  1.101227 -0.190701  ... -0.447363  0.100738  0.147102
## Berger All  -0.486396 -0.464450 -0.498134  ...  0.187330 -0.234185 -0.008920
## Boxer        0.447365 -0.881778  0.692016  ... -0.019819 -0.002446  0.140901
## Bull-Dog     1.013352  0.549879 -0.163423  ... -0.079036 -0.035602  0.066543
##
## [5 rows x 10 columns]
```

```
my_mca.fit_transform(A).iloc[:5,:]
```

```
##                 Dim.1     Dim.2     Dim.3  ...     Dim.8     Dim.9    Dim.10
## Beauceron   -0.317200 -0.417701 -0.101468  ...  0.201986 -0.167019  0.022807
## Basset       0.254110  1.101227 -0.190701  ... -0.447363  0.100738  0.147102
## Berger All  -0.486396 -0.464450 -0.498134  ...  0.187330 -0.234185 -0.008920
## Boxer        0.447365 -0.881778  0.692016  ... -0.019819 -0.002446  0.140901
## Bull-Dog     1.013352  0.549879 -0.163423  ... -0.079036 -0.035602  0.066543
##
## [5 rows x 10 columns]
```

3.5.1 Intégration dans une Pipeline de scikit-learn

La class MCA peut être intégrée dans une Pipeline de scikit-learn. Dans le cadre de notre exemple, nous cherchons à prédire la 7ème variable (variable "Fonction") à partir des 6 premières variables du jeu de données.

“Fonction” est une variable catégorielle comprenant 3 catégories : “chasse”, “compagnie” et “utilité”. Pour la prédire, nous allons utiliser un modèle de régression logistique qui prendra en input des axes issus d’une Analyse des Correspondances Multiples pratiquée sur les données brutes.

Dans un premier temps, et de façon tout à fait arbitraire, nous fixons le nombre de composantes extraites à 4.

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
import numpy as np

# X = features
X = A
# y = labels
y = C

# Construction de la Pipeline
# On enchaine une Analyse des Correspondances Multiples (4 axes retenus)
# puis une régression logistique
pipe = Pipeline([("mca", MCA(n_components=4)),
                 ("logistic_regression",
                  LogisticRegression(multi_class="multinomial",
                                     solver="lbfgs",penalty=None))])
# Estimation du modèle
pipe.fit(X, y)
```

```
## Pipeline(steps=[('mca', MCA(n_components=4)),
##                 ('logistic_regression',
##                  LogisticRegression(multi_class='multinomial', penalty=None))])
```

On prédit

```
# Prédiction sur l'échantillon de test
print(pipe.predict(B))
```

```
## ['utilite' 'chasse' 'chasse' 'compagnie' 'chasse' 'utilite']
```

Le paramètre `n_components` peut faire l’objet d’une optimisation via GridSearchCV de scikit-learn.

Nous reconstruisons donc une Pipeline, sans spécifier de valeur a priori pour n_components.

```
# Reconstruction d'une Pipeline, sans spécifier de valeur
# a priori pour n_components
pipe2 = Pipeline([("mca", MCA()),
                  ("logistic_regression", LogisticRegression(penalty=None))])

# Paramétrage de la grille de paramètres
```

```
# Attention à l'étendue des valeurs possibles pour pca__n_components !!!
param = [{"mca__n_components": [x + 1 for x in range(10)]}]

# Construction de l'obet GridSearchCV
grid_search = GridSearchCV(pipe2,
                           param_grid=param,
                           scoring="accuracy",
                           cv=5,
                           verbose=0)
# Estimation du modèle
grid_search.fit(X, y)
```

```
## GridSearchCV(cv=5,
##              estimator=Pipeline(steps=[('mca', MCA()),
##                                        ('logistic_regression',
##                                         LogisticRegression(penalty=None))]),
##              param_grid=[{'mca__n_components': [1, 2, 3, 4, 5, 6, 7, 8, 9,
##                                                 10]}],
##              scoring='accuracy')
```

```
# Affichage du score optimal
grid_search.best_score_
```

```
## 0.82
```

```
# Affichage du paramètre optimal
grid_search.best_params_
```

```
## {'mca__n_components': 7}
```

```
# Prédiction sur l'échantillon de test
grid_search.predict(B)
```

```
## array(['utilite', 'chasse', 'utilite', 'chasse', 'compagnie', 'utilite'],
##       dtype=object)
```

4

Analyse Factorielle des Données Mixtes

Sommaire

Ce chapitre a pour objectif de présenter rapidement les principales fonctionnalités offertes par le package « scientisttools » pour réaliser une Analyse Factorielle des Données Mixtes.

4.1 Présentation des données

L'analyse factorielle des données mixtes traite les tableaux individus-variables, lesquelles sont composées d'un mix de quantitatives et qualitatives. Nous utilisons la base des données « Autos 2005 » accessible sur la page de cours de Pierre-Louis Gonzalez (https ://maths.cnam.fr/spip.php ?article50) au CNAM. Notre base comporte $(I = 38)$ modèles de véhicules décrits par $(K_1 = 9)$ variables quantitatives (puissance, cylindrée, vitesse, longueur, largeur, hauteur, poids, CO2 et prix) et $(K_2 = 3)$ variables qualitatives (origie avec 3 modalités : France, Europe, Autres ; carburant avec 2 modalités : diesel, essence ; type4X4 avec 2 modalités : oui, non).

4.1.1 Importation des données

```
# Chargement des données
import pandas as pd
# Données actives
A = pd.read_excel("./donnee/autos2005.xlsx",sheet_name=0,index_col=0)
# Individus actifs
B = pd.read_excel("./donnee/autos2005.xlsx",sheet_name=1,index_col=0)
# Variables illustratives quantitatives
C = pd.read_excel("./donnee/autos2005.xlsx",sheet_name=2,index_col=0)
# Variables illustratives qualitatives
```

```
D = pd.read_excel("./donnee/autos2005.xlsx",sheet_name=3,index_col=0)
C.index = D.index = A.index
# Concaténation
Data = pd.concat([pd.concat([A,B],axis=0),C,D],axis=1)
# Affichage des caractéristiques
Data.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 45 entries, ALFA 156     to MERCEDES
## Data columns (total 16 columns):
##  #   Column        Non-Null Count  Dtype
## ---  ------        --------------  -----
##  0   puissance     45 non-null     int64
##  1   cylindree     45 non-null     int64
##  2   vitesse       45 non-null     int64
##  3   longueur      45 non-null     int64
##  4   largeur       45 non-null     int64
##  5   hauteur       45 non-null     int64
##  6   poids         45 non-null     int64
##  7   CO2           45 non-null     int64
##  8   prix          45 non-null     int64
##  9   origine       45 non-null     object
##  10  carburant     45 non-null     object
##  11  type4X4       45 non-null     object
##  12  coffre        38 non-null     float64
##  13  reservoir     38 non-null     float64
##  14  consommation  38 non-null     float64
##  15  surtaxe       38 non-null     object
## dtypes: float64(3), int64(9), object(4)
## memory usage: 7.0+ KB
```

Les variables coffre, reservoir et consommation seront utilisées comme illustratives quantitatives et « surtaxe » comme illustrative qualitative. Certaines voitures ont été mises en illustratives.

Les questions usuelles que l'on se pose sont les suivantes :

1. Quelles sont les véhicules qui se ressemblent, c'est - à - dire qui présentent des caractéristiques similaires ? Il sera question d'étudier les proximités entre les individus.
2. Sur quelles caractéristiques sont basées les ressemblances et dissemblances, avec la difficulté ici de les comptabiliser de manière différenciée selon que les variables incriminées sont quantitatives ou qualitatives.
3. Quelles sont les relations entre les variables ? Entre quantitatives, l'idée de la corrélation s'impose ; entre qualitatives, le χ^2 de contingence. Mais comment faire entre quantitatives et qualitatives ? (rapport de corrélation, etc.)

Table 4.1 – Données Autos 2005

	puissance	cylindree	vitesse	longueur	largeur	hauteur	poids	CO2	prix	origine	carburant	type4X4	coffre	reservoir	consommation	surtaxe
ALFA 156	250	3179	250	443	175	141	1410	287	40800	Europe	Essence	type4X4_non	378	63	12.1	surtaxe_oui
AUDIA3	102	1595	185	421	177	143	1205	168	21630	Europe	Essence	type4X4_non	350	55	7.0	surtaxe_oui
AUDIA8	280	3697	250	506	203	145	1770	281	78340	Europe	Essence	type4X4_non	500	90	11.7	surtaxe_non
AVENSIS	115	1995	195	463	176	148	1400	155	26400	Autres	Diesel	type4X4_non	510	60	5.8	surtaxe_oui
BMW X5	218	2993	210	467	188	172	2095	229	52000	Europe	Diesel	type4X4_non	465	93	8.6	surtaxe_oui
BMW530	231	2979	250	485	185	147	1495	231	46400	Europe	Essence	type4X4_non	520	70	9.5	surtaxe_non
CHRYS300	340	5654	250	502	188	148	1835	291	54900	Autres	Essence	type4X4_non	442	72	12.2	surtaxe_non
CITRONC2	61	1124	158	367	166	147	932	141	10700	France	Essence	type4X4_non	224	41	5.9	surtaxe_non
CITRONC4	138	1997	207	426	178	146	1381	142	23400	France	Diesel	type4X4_non	314	60	5.4	surtaxe_oui
CITRONC5	210	2496	230	475	178	148	1589	238	33000	France	Essence	type4X4_non	471	65	10.0	surtaxe_non
CLIO	100	1461	185	382	164	142	980	113	17600	France	Diesel	type4X4_non	255	50	4.3	surtaxe_non
CORSA	70	1248	165	384	165	144	1035	127	13590	Europe	Diesel	type4X4_non	260	45	4.7	surtaxe_oui
FIESTA	68	1399	164	392	168	144	1138	117	14150	Europe	Diesel	type4X4_non	261	45	4.4	surtaxe_oui
GOLF	75	1968	163	421	176	149	1217	143	19140	Europe	Diesel	type4X4_non	350	55	5.4	surtaxe_oui
LAGUNA	165	1998	218	458	178	143	1320	196	25350	France	Essence	type4X4_non	430	70	8.2	surtaxe_non
LANDCRUI	204	4164	170	489	194	185	2495	292	67100	Autres	Diesel	type4X4_oui	403	96	11.1	surtaxe_oui
MAZDARX8	231	1308	235	443	177	134	1390	284	34000	Autres	Essence	type4X4_non	287	61	11.4	surtaxe_oui
MEGANECC	165	1998	225	436	178	141	1415	191	27800	France	Essence	type4X4_non	190	60	8.0	surtaxe_oui
MERC_A	140	1991	201	384	177	160	1340	141	24550	Europe	Diesel	type4X4_non	435	54	5.4	surtaxe_oui
MERC_E	204	3222	243	482	183	146	1735	183	46450	Europe	Diesel	type4X4_non	520	80	6.9	surtaxe_non
MODUS	113	1598	188	380	170	159	1170	163	16950	France	Essence	type4X4_non	198	49	6.8	surtaxe_oui
MONDEO	145	1999	215	474	194	143	1378	189	23100	Europe	Essence	type4X4_non	500	59	7.9	surtaxe_oui
MURANO	234	3498	200	477	188	171	1870	295	44000	Autres	Essence	type4X4_oui	438	82	12.3	surtaxe_oui
MUSA	100	1910	179	399	170	169	1275	146	17900	Europe	Diesel	type4X4_non	320	47	5.5	surtaxe_non
OUTLAND	202	1997	220	455	178	167	1595	237	29990	Autres	Diesel	type4X4_oui	402	60	10.0	surtaxe_oui
P1007	75	1360	165	374	169	161	1181	153	13600	France	Essence	type4X4_non	178	50	6.4	surtaxe_oui
P307CC	180	1997	225	435	176	143	1490	210	28850	France	Essence	type4X4_non	204	50	8.8	surtaxe_non
P407	136	1997	212	468	182	145	1415	194	23400	France	Essence	type4X4_non	407	66	8.2	surtaxe_non
P607	204	2721	230	491	184	145	1723	223	40550	France	Diesel	type4X4_non	468	80	8.4	surtaxe_oui
PANDA	54	1108	150	354	159	154	860	135	8070	Europe	Essence	type4X4_non	206	35	5.7	surtaxe_non
PASSAT	150	1781	221	471	175	147	1360	197	27740	Europe	Essence	type4X4_non	475	62	8.2	surtaxe_non
PTCRUISER	223	2429	200	429	171	154	1595	235	27400	Autres	Essence	type4X4_non	210	57	9.9	surtaxe_oui
SANTA_FE	125	1991	172	450	185	173	1757	197	27990	Autres	Diesel	type4X4_oui	833	65	7.5	surtaxe_oui
TWINGO	60	1149	151	344	163	143	840	143	8950	France	Essence	type4X4_non	168	40	6.0	surtaxe_non
VECTRA	150	1910	217	460	180	146	1428	159	26550	Europe	Diesel	type4X4_non	500	61	5.9	surtaxe_oui
VELSATIS	150	2188	200	486	186	158	1735	188	38250	France	Diesel	type4X4_non	460	80	7.1	surtaxe_oui
X-TRAIL	136	2184	180	446	177	168	1520	190	29700	Autres	Diesel	type4X4_oui	350	60	7.2	surtaxe_oui
YARIS	65	998	155	364	166	150	880	134	10450	Autres	Essence	type4X4_non	205	45	5.6	surtaxe_non
CORVETTE	404	5970	300	444	185	125	1517	310	63350	Autres	Essence	type4X4_non	NaN	NaN	NaN	NA
TAHOE	290	5327	170	506	223	196	2463	340	49600	Autres	Essence	type4X4_oui	NaN	NaN	NaN	NA
OPEL	339	4188	227	351	192	181	2235	166	77810	France	Diesel	type4X4_non	NaN	NaN	NaN	NA
RENAULT	183	4439	160	387	186	137	1177	228	31681	France	Essence	type4X4_oui	NaN	NaN	NaN	NA
CITROEN	88	4161	231	367	182	139	947	185	51161	Europe	Essence	type4X4_non	NaN	NaN	NaN	NA
TOYOTA	83	1880	232	373	195	149	1229	118	51233	Autres	Diesel	type4X4_non	NaN	NaN	NaN	NA
MERCEDES	311	5361	188	375	159	145	874	339	72876	Europe	Diesel	type4X4_oui	NaN	NaN	NaN	NA

4.2 AFDM

4.2.1 Objectifs

L'objectif est de trouver un système de représentation (répère factoriel) qui préserve au mieux les distances entre les individus, qui permet de discerner le mieux possible les individus entre eux, qui maximise les (le carré des) écarts à l'origine.

4.2.2 Chargement de scientisttools

```
from scientisttools import FAMD
```

4.2.2.1 Individus et variables actifs

On crée une instance de la classe FAMD, en lui passant ici des étiquettes pour les lignes et les variables. Ces paramètres sont facultatifs ; en leur absence, le programme détermine automatiquement des étiquettes.

```
# Instanciation
my_famd = FAMD()
```

On estime le modèle en appliquant la méthode `fit` de la classe FAMD sur le jeu de données.

```python
# Estimation du modèle
my_famd.fit(A)
```

```
## FAMD()
```

4.2.2.2 Les valeurs propres

L'exécution de la méthode `my_famd.fit(A)` provoque le calcul des attributs parmi lesquels `my_famd.eig_` pour les valeurs propres.

```python
# Valeurs propres
print(my_famd.eig_)
```

```
##          eigenvalue  difference  proportion  cumulative
## Dim.1      6.602293    4.092257   50.786869   50.786869
## Dim.2      2.510036    1.205140   19.307967   70.094835
## Dim.3      1.304896    0.438129   10.037661   80.132497
## Dim.4      0.866767    0.309234    6.667435   86.799932
## Dim.5      0.557532    0.167951    4.288710   91.088642
## Dim.6      0.389581    0.121886    2.996776   94.085418
## Dim.7      0.267694    0.095605    2.059188   96.144606
## Dim.8      0.172089    0.032077    1.323764   97.468370
## Dim.9      0.140012    0.043340    1.077018   98.545388
## Dim.10     0.096673    0.045782    0.743635   99.289023
## Dim.11     0.050890    0.019811    0.391465   99.680488
## Dim.12     0.031080    0.020623    0.239075   99.919563
## Dim.13     0.010457         NaN    0.080437  100.000000
```

L'attribut `my_famd.eig_` contient :

- en 1ère ligne : les valeurs propres en valeur absolue
- en 2ème ligne : les différences des valeurs propres
- en 3ème ligne : les valeurs propres en pourcentage de la variance totale (proportions)
- en 4ème ligne : les valeurs propres en pourcentage cumulé de la variance totale.

La fonction `get_eig` retourne les valeurs propres sous forme de tableau de données.

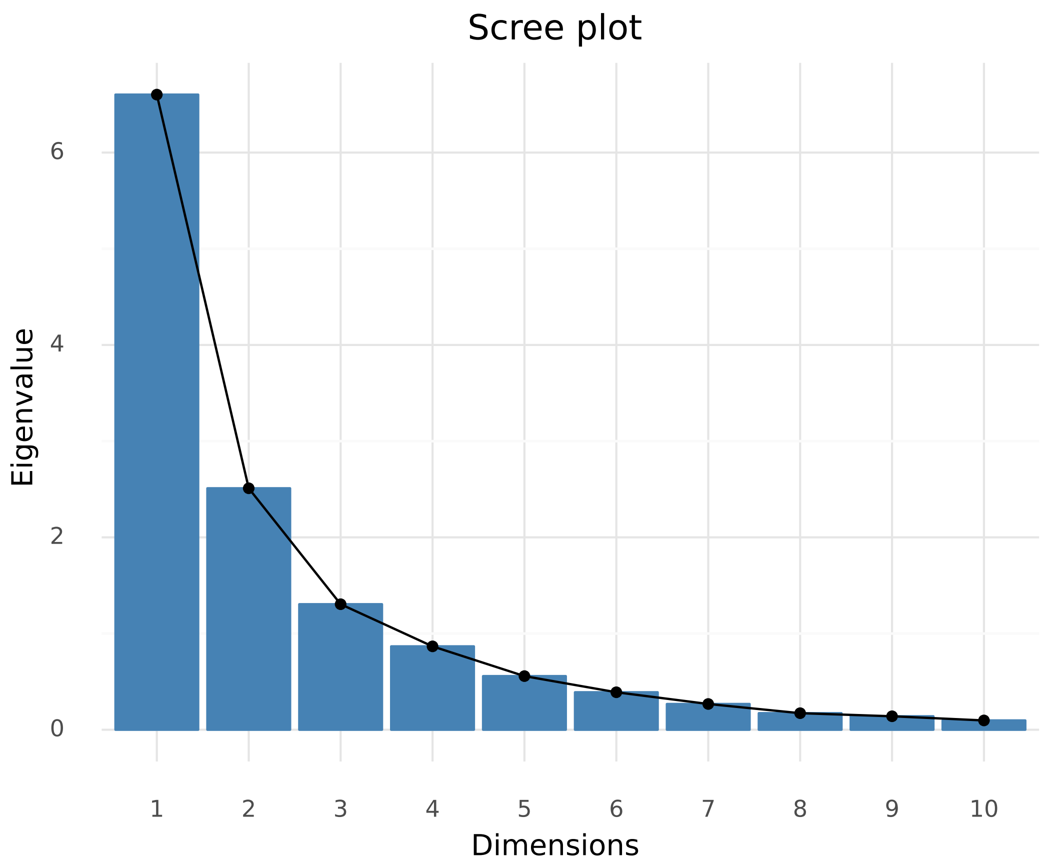
```python
# Valeurs propres
from scientisttools import get_eig
print(get_eig(my_famd))
```

```
##          eigenvalue  difference  proportion  cumulative
## Dim.1      6.602293    4.092257   50.786869   50.786869
## Dim.2      2.510036    1.205140   19.307967   70.094835
## Dim.3      1.304896    0.438129   10.037661   80.132497
## Dim.4      0.866767    0.309234    6.667435   86.799932
## Dim.5      0.557532    0.167951    4.288710   91.088642
## Dim.6      0.389581    0.121886    2.996776   94.085418
## Dim.7      0.267694    0.095605    2.059188   96.144606
```

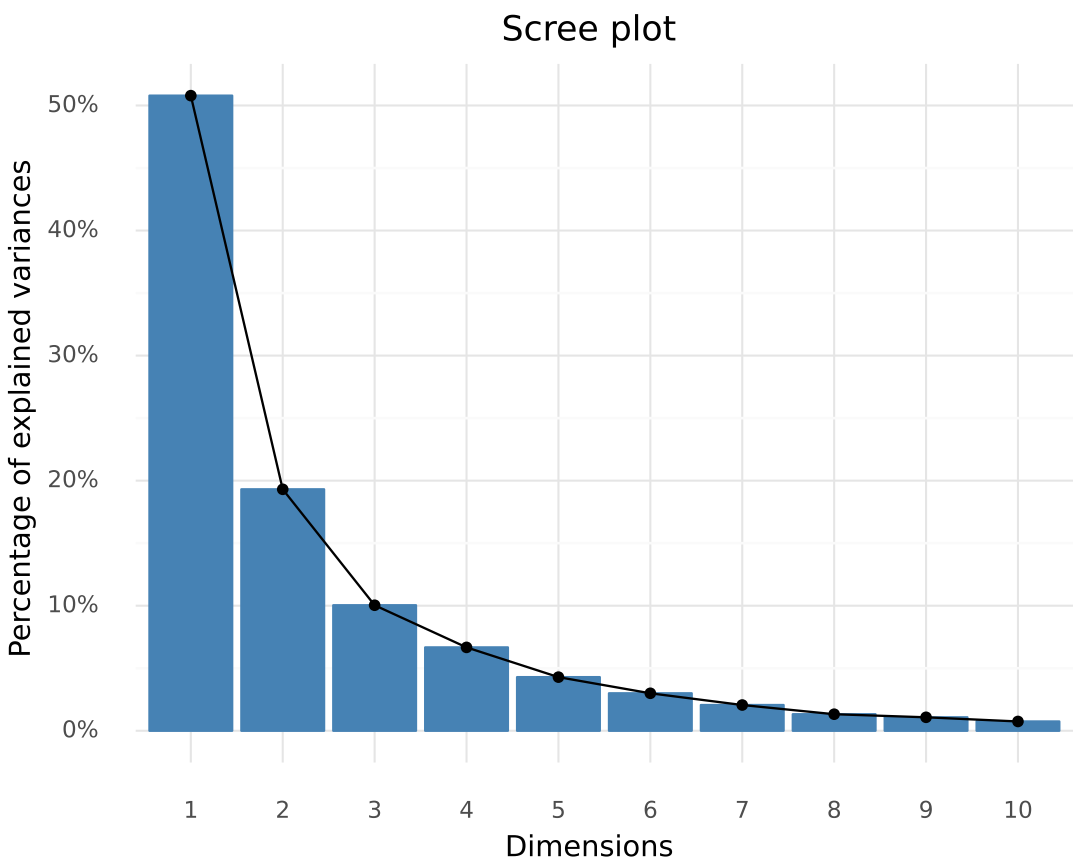
```
## Dim.8     0.172089    0.032077    1.323764   97.468370
## Dim.9     0.140012    0.043340    1.077018   98.545388
## Dim.10    0.096673    0.045782    0.743635   99.289023
## Dim.11    0.050890    0.019811    0.391465   99.680488
## Dim.12    0.031080    0.020623    0.239075   99.919563
## Dim.13    0.010457         NaN    0.080437  100.000000
```

Les valeurs propres peuvent être représentées graphiquement :

```
from scientisttools import fviz_screeplot
print(fviz_screeplot(my_famd,choice="eigenvalue"))
```




```
print(fviz_screeplot(my_famd,choice="proportion"))
```



Même si le mécanisme sous - jacent de l'AFDM repose sur une ACP, nous ne pouvons pas vraiment utiliser les stratégies usuelles.

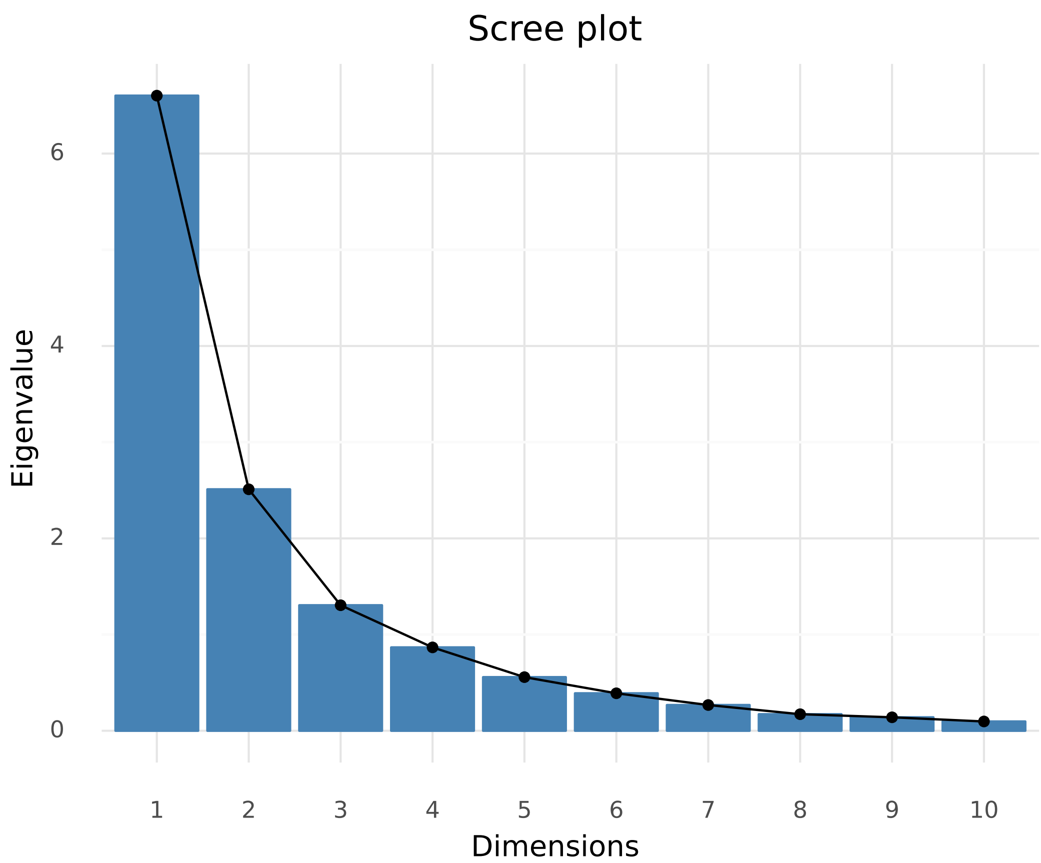
— Avec la règle de Kaiser, nous sélectionnerons les facteurs tels que $\lambda_\alpha \geq 1$, c'est - à - dire $H = 3$. On se rend compte en pratique que ce critère est trop permissif, nous retenons un nombre excessif de facteurs parce qu'une partie de l'information est rédondante comme en ACM.

— Avec la règle de Karlis - Saporta - Spinaki, la valeur seuil est surestimé parce que le nombre de colonnes K des données présentées à l'ACP est « surévalué », des redondances ont été artificiellement introduites.

De fait, les critères de l'ACP ne s'appliquent pas ici parce que les données ne sont pas composées de variables nativement quantitatives, certaines colonnes sont liées entre elles avec le codage disjonctif complet des variables qualitatives.

Finalement, on en revient au diagramme des valeurs propres et la recherche de « coudes », annonciateurs de changement significatif de structure dans les données.

```
print(fviz_screeplot(my_famd,choice="eigenvalue"))
```



Nous avons un « coude » au niveau de $h = 2$. Nous choisissons de retenir $H = 2$.

On peut obtenir un résumé des principaux résultats en utilisant la fonction `summaryFAMD`.

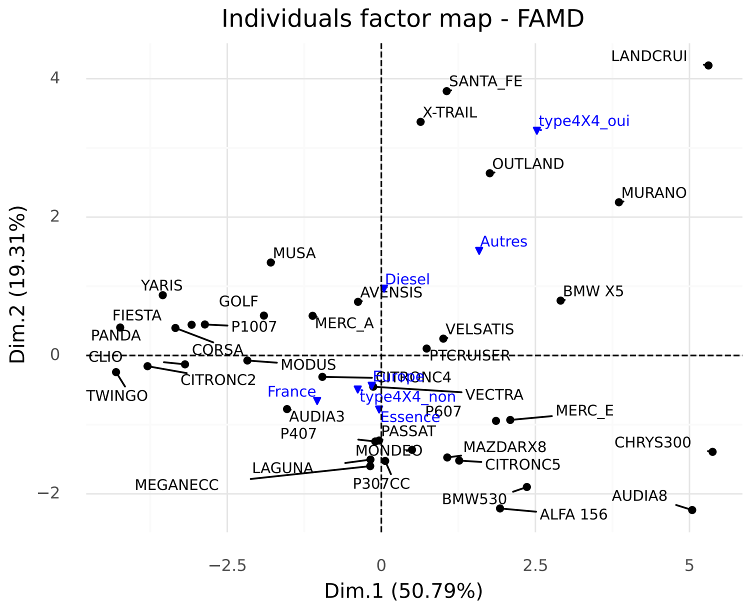
```
from scientisttools import summaryFAMD
summaryFAMD(my_famd)
```

```
##                        Factor Analysis of Mixed Data - Results
##
## Importance of components
##                           Dim.1   Dim.2   Dim.3  ...  Dim.11  Dim.12  Dim.13
## Variance                  6.602   2.510   1.305  ...   0.051   0.031    0.01
## Difference                4.092   1.205   0.438  ...   0.020   0.021     NaN
## % of var.                50.787  19.308  10.038  ...   0.391   0.239    0.08
## Cumulative of % of var.  50.787  70.095  80.132  ...  99.680  99.920  100.00
##
## [4 rows x 13 columns]
##
## Individuals (the 10 first)
##
##                dist  weight  inertia  Dim.1  ...   cos2  Dim.3    ctr   cos2
## ALFA 156      3.577   0.026    0.337  1.926  ...  0.382 -0.137  0.038  0.001
## AUDIA3        2.321   0.026    0.142 -1.530  ...  0.111  0.396  0.316  0.029
## AUDIA8        5.900   0.026    0.916  5.044  ...  0.143  0.887  1.586  0.023
```

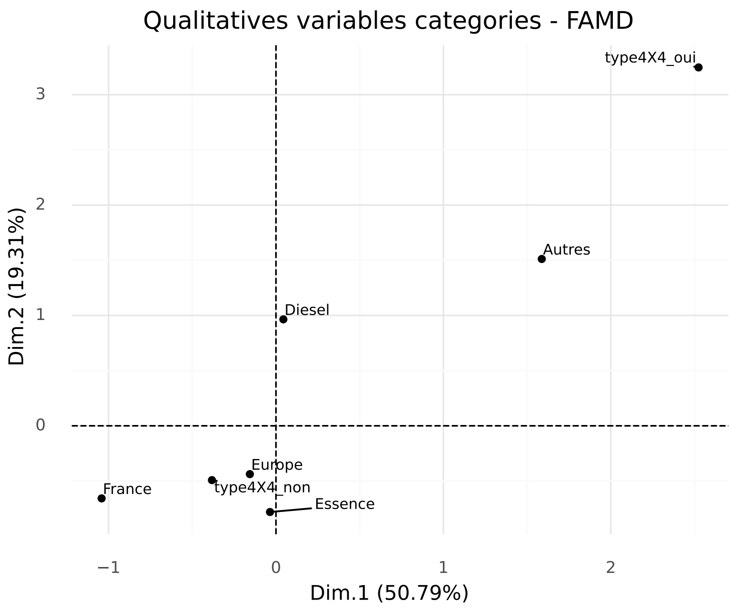
```
## AVENSIS       2.376   0.026    0.149 -0.379  ...  0.106  0.289  0.169  0.015
## BMW X5        3.977   0.026    0.416  2.909  ...  0.040  1.942  7.607  0.239
## BMW530        3.283   0.026    0.284  2.362  ...  0.335  0.541  0.591  0.027
## CHRYS300      6.191   0.026    1.009  5.375  ...  0.051 -1.124  2.549  0.033
## CITRONC2      4.079   0.026    0.438 -3.793  ...  0.001 -1.192  2.866  0.085
## CITRONC4      2.203   0.026    0.128 -0.956  ...  0.020  0.492  0.489  0.050
## CITRONC5      2.530   0.026    0.168  1.264  ...  0.361 -1.003  2.028  0.157
##
## [10 rows x 12 columns]
##
## Continuous variables
##
##            Dim.1    ctr   cos2  Dim.2    ctr   cos2  Dim.3    ctr   cos2
## puissance  0.916 12.704  0.839 -0.271  2.936  0.074 -0.126  1.220  0.016
## cylindree  0.888 11.951  0.789 -0.039  0.061  0.002  0.046  0.160  0.002
## vitesse    0.703  7.495  0.495 -0.598 14.263  0.358  0.039  0.115  0.001
## longueur   0.899 12.229  0.807 -0.142  0.804  0.020  0.130  1.292  0.017
## largeur    0.876 11.621  0.767 -0.039  0.060  0.001  0.187  2.693  0.035
## hauteur    0.288  1.260  0.083  0.846 28.519  0.716  0.028  0.059  0.001
## poids      0.915 12.693  0.838  0.249  2.469  0.062  0.099  0.757  0.010
## CO2        0.891 12.037  0.795 -0.090  0.323  0.008 -0.327  8.202  0.107
## prix       0.940 13.397  0.884 -0.062  0.154  0.004  0.126  1.218  0.016
##
## Categories
##
##               dist  weight  inertia  Dim.1  ...  Dim.3    ctr   cos2  vtest
## Autres       1.673   0.088    0.246  1.588  ... -0.896 12.409  0.287 -2.851
## Europe       1.238   0.132    0.202 -0.156  ...  1.096 27.832  0.783  4.712
## France       1.387   0.114    0.219 -1.041  ... -0.575  6.643  0.172 -2.208
## Diesel       1.111   0.149    0.184  0.044  ...  0.837 18.405  0.567  4.010
## Essence      0.900   0.184    0.149 -0.036  ... -0.678 14.899  0.567 -4.010
## type4X4_non  0.389   0.289    0.044 -0.382  ...  0.103  0.539  0.070  1.406
## type4X4_oui  2.569   0.044    0.289  2.524  ... -0.679  3.558  0.070 -1.406
##
## [7 rows x 15 columns]
##
## Categorical variables (eta2)
##
##            Dim.1  Dim.2  Dim.3
## origine    0.158  0.329  0.612
## carburant  0.000  0.300  0.435
## type4X4    0.146  0.637  0.053
```

4.2.3 Représentation graphique

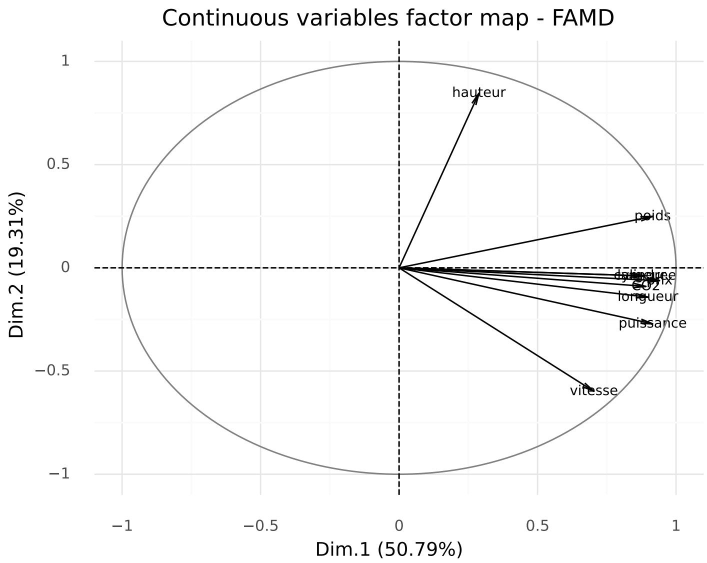
```
# Carte des individus
from scientisttools import fviz_famd_ind
print(fviz_famd_ind(my_famd,repel=True))
```



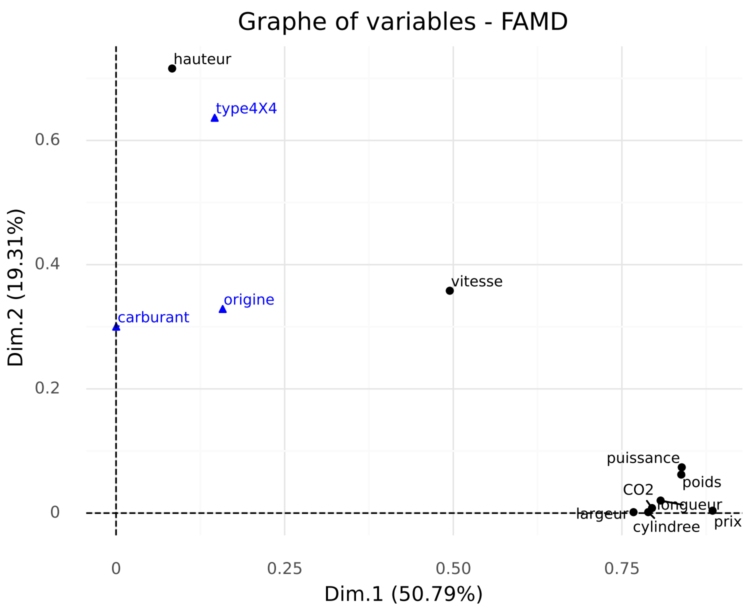

```python
# Carte des modalités - variables qualitatives
from scientisttools import fviz_famd_mod
print(fviz_famd_mod(my_famd,repel=True))
```




```python
# Cercle des corrélations - variables quantitatives
from scientisttools import fviz_famd_col
print(fviz_famd_col(my_famd))
```



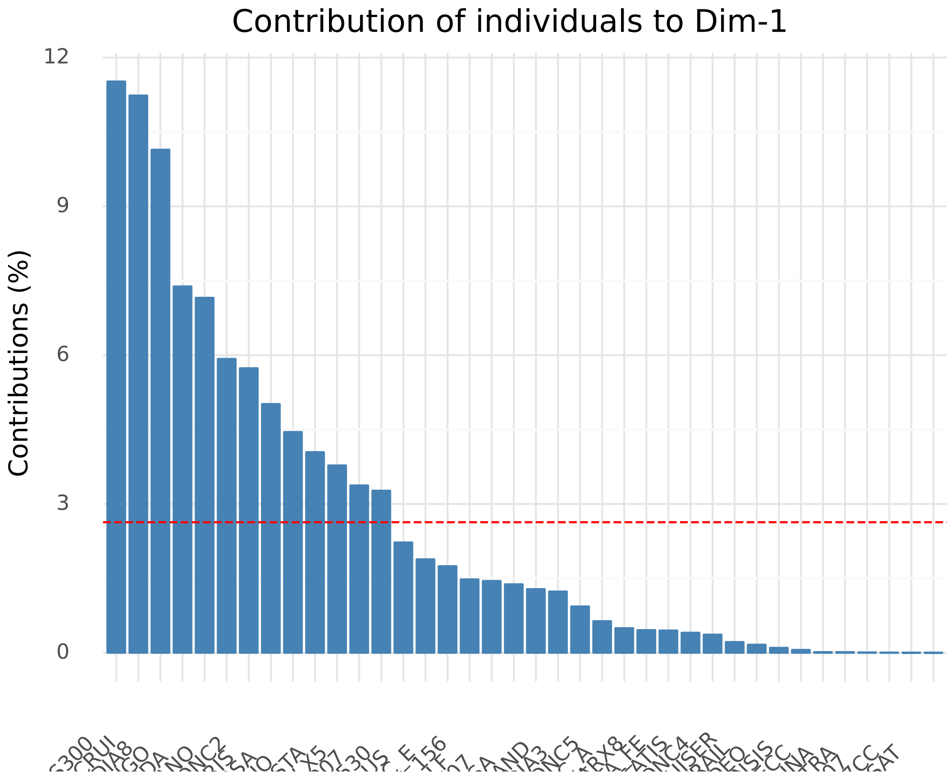

```
# Carte des variables - rapport de corrélation et cosinus carré
from scientisttools import fviz_famd_var
p = fviz_famd_var(my_famd,repel=True)
print(p)
```



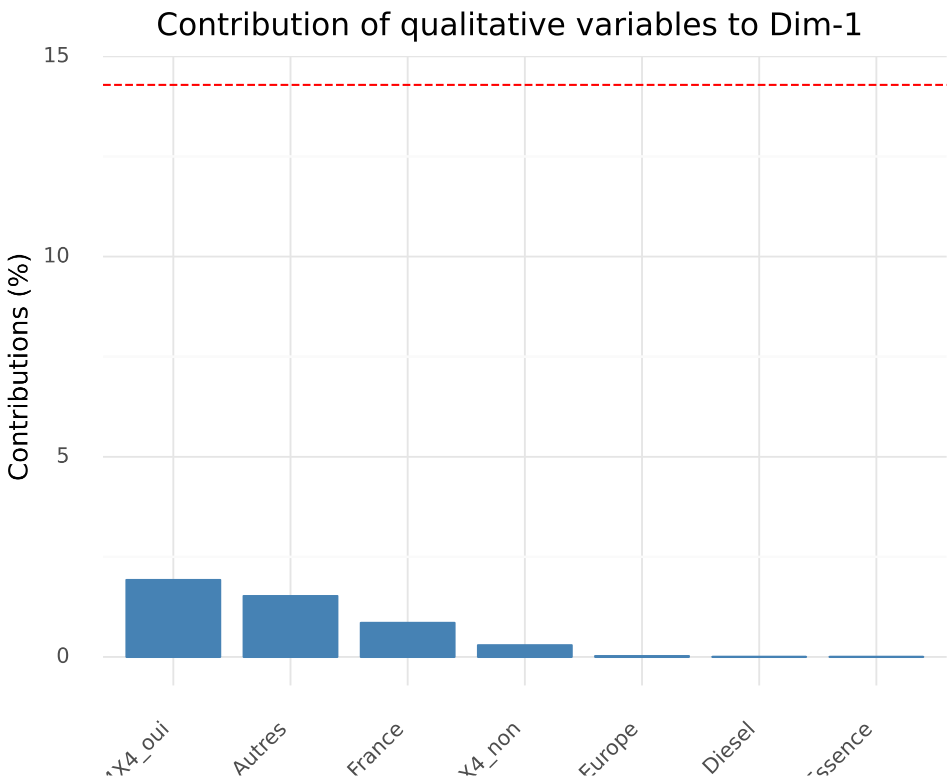
4.3 Interprétation des axes

Des graphiques qui permettent d'interpréter rapidement les axes : on choisit un axe factoriel (le 1er axe dans notre exemple) et on observe quels sont les points lignes et colonnes qui présentent les plus fortes contributions et cos2 pour cet axe.

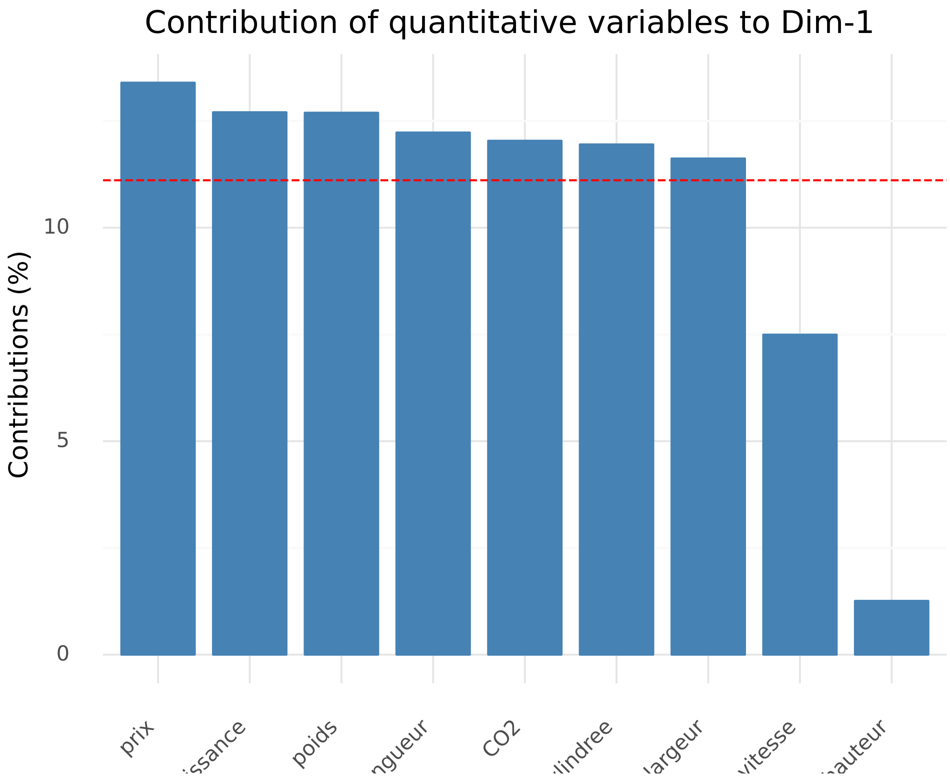
```
# Classement des points lignes en fonction de leur contribution au 1er axe
from scientisttools import fviz_contrib, fviz_cos2
print(fviz_contrib(my_famd,choice="ind"))
```



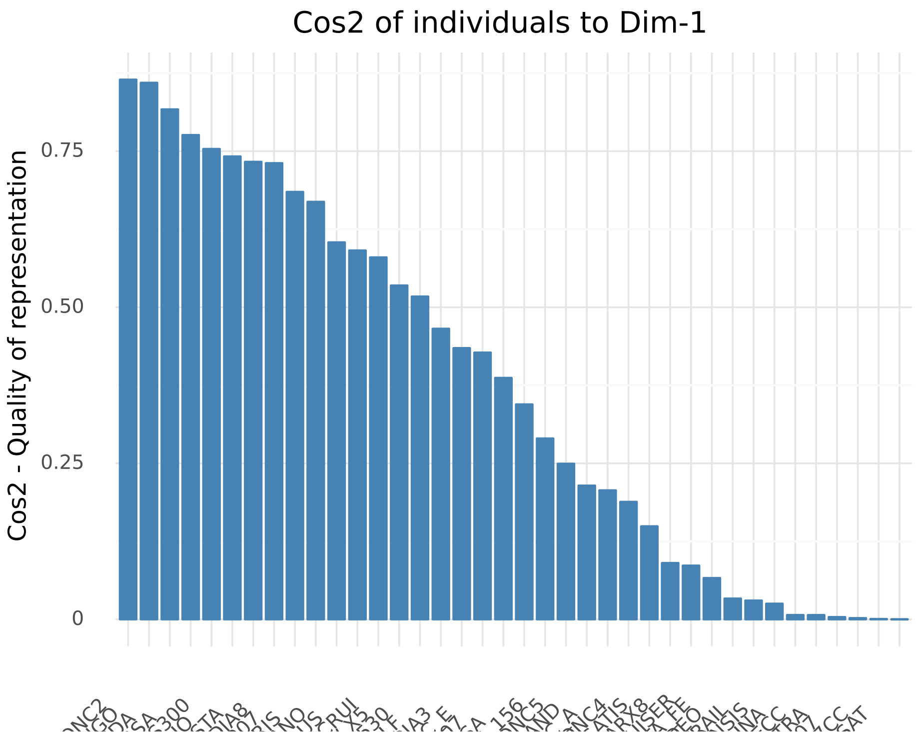

```
# Classement des modalités en fonction de leur contribution au 1er axe
print(fviz_contrib(my_famd,choice="quali_var"))
```



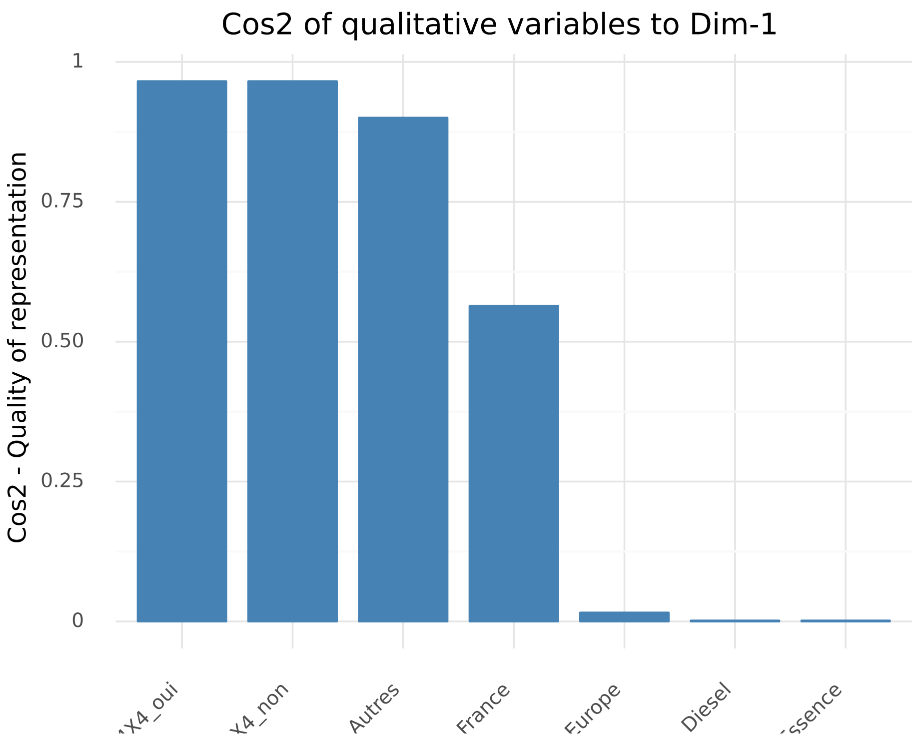
```
# Classement des variables quantitatives
# en fonction de leur contribution au 1er axe
print(fviz_contrib(my_famd,choice="quanti_var"))
```



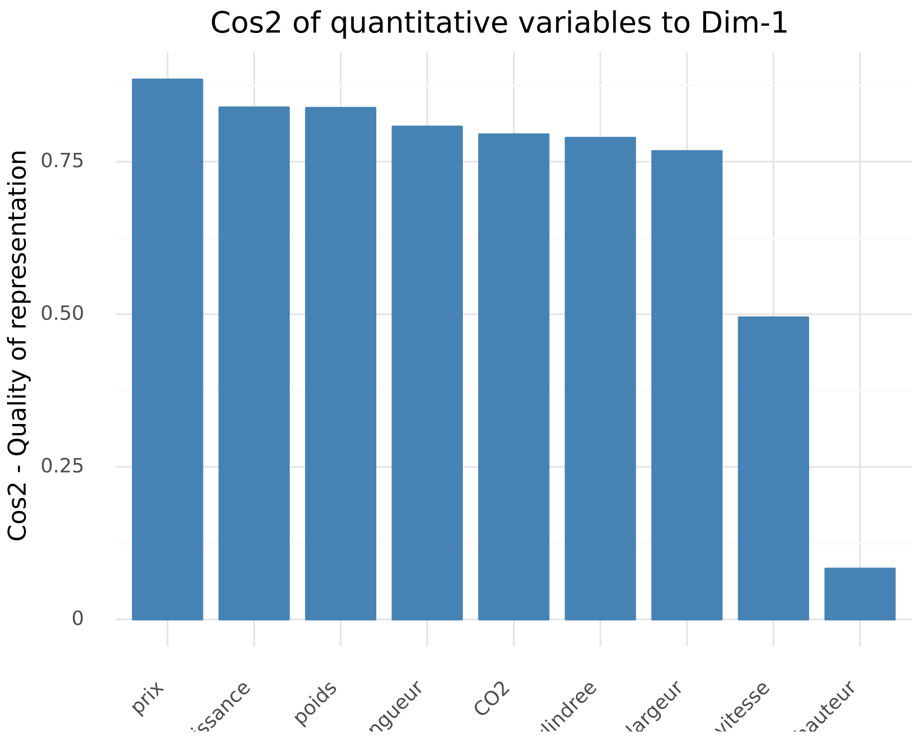

```
# Classement des individus en fonction de leur cos2 sur le 1er axe
print(fviz_cos2(my_famd,choice="ind"))
```




```r
# Classement des modalités en fonction de leur cos2 sur le 1er axe
print(fviz_cos2(my_famd,choice="quali_var"))
```




```r
# Classement des variables quantitatives
# en fonction de leur cos2 sur le 1er axe
print(fviz_cos2(my_famd,choice="quanti_var"))
```



4.4 Approche Machine Learning

Ici, l'objectif est d'utiliser l'Analyse Factorielle des Données Mixtes en tant que méthode de prétraitement.

La classe FAMD implémente les méthodes `fit`, `transform` et `fit_transform` bien connues des utilisateurs de scikit-learn.

```
my_famd.transform(A).iloc[:5,:2]
```

```
##                  Dim.1     Dim.2
## ALFA 156      1.926355 -2.210697
## AUDIA3       -1.530127 -0.774674
## AUDIA8        5.044196 -2.233793
## AVENSIS      -0.378548  0.775136
## BMW X5        2.908971  0.792592
```

```
my_famd.fit_transform(A).iloc[:5,:2]
```

```
##                  Dim.1     Dim.2
## ALFA 156      1.926355 -2.210697
## AUDIA3       -1.530127 -0.774674
## AUDIA8        5.044196 -2.233793
## AVENSIS      -0.378548  0.775136
## BMW X5        2.908971  0.792592
```

4.4.1 Intégration dans une Pipeline de scikit-learn

La class `FAMD` peut être intégrée dans une Pipeline de scikit-learn. Dans le cadre de notre exemple, nous cherchons à prédire la variable (variable “Prix”) à partir des 11 autres variables du jeu de données (données actives).

“prix” est une variable quantitative. Pour la prédire, nous allons utiliser un modèle de régression linéaire qui prendra en input des axes issus d’une Analyse Factorielle des Données Mixtes pratiquée sur les données brutes.

Dans un premier temps, et de façon tout à fait arbitraire, nous fixons le nombre de composantes extraites à 4.

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV

# X = features
X = A.drop(columns=["prix"])
# y = target
y = A.prix

# Construction de la Pipeline
# On enchaine une Analyse Factorielle des Données Mixtes (4 axes retenus)
# puis une régression linéaire
pipe = Pipeline([("famd", FAMD(n_components=4)),
                 ("ols", LinearRegression())])
# Estimation du modèle
pipe.fit(X, y)
```

```
## Pipeline(steps=[('famd', FAMD(n_components=4)), ('ols', LinearRegression())])
```

On prédit

```
# Prédiction sur l'échantillon de test
print(pipe.predict(B))
```

```
## [58779.0497945  69412.02123729 46817.29617417 29828.26194279
##  27688.40210294 26546.10674283 36394.992253  ]
```

Le paramètre `n_components` peut faire l’objet d’une optimisation via GridSearchCV de scikit-learn.

Nous reconstruisons donc une Pipeline, sans spécifier de valeur a priori pour n_components.

```
# Reconstruction d'une Pipeline, sans spécifier de valeur
# a priori pour n_components
pipe2 = Pipeline([("famd", FAMD()),
                  ("ols", LinearRegression())])

# Paramétrage de la grille de paramètres
```

```python
# Attention à l'étendue des valeurs possibles pour famd__n_components !!!
param = [{"famd__n_components": [x + 1 for x in range(12)]}]

# Construction de l'objet GridSearchCV
grid_search = GridSearchCV(pipe2,
                           param_grid=param,
                           scoring="neg_mean_squared_error",
                           cv=5,
                           verbose=0)
# Estimation du modèle
grid_search.fit(X, y)
```

```
## GridSearchCV(cv=5,
##              estimator=Pipeline(steps=[('famd', FAMD()),
##                                        ('ols', LinearRegression())]),
##              param_grid=[{'famd__n_components': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
##                                                  11, 12]}],
##              scoring='neg_mean_squared_error')
```

```python
# Affichage du score optimal
grid_search.best_score_
```

```
## -49681609.97640531
```

```python
# Affichage du RMSE optimal
import numpy as np
print(np.sqrt(-grid_search.best_score_))
```

```
## 7048.518282334615
```

```python
# Affichage du paramètre optimal
grid_search.best_params_
```

```
## {'famd__n_components': 6}
```

```python
# Prédiction sur l'échantillon de test
grid_search.predict(B)
```

```
## array([59166.75977397, 75330.61691695, 55918.63883543, 31786.01147151,
##        30135.88354508, 22455.79889542, 41006.95089521])
```

5 Classification Hiérarchique sur Composantes Principales

Sommaire

Ce chapitre a pour objectif de présenter rapidement les principales fonctionnalités offertes par le package « scientisttools » pour réaliser une classification hiérarchique combinée avec une analyse factorielle (HCPC, *Hierarchical Clustering on Principal Components*).

5.1 Présentation des données

On va réaliser une classification hiérarchique sur les composantes principales d'une analyse factorielle. Nous allons prendre un exemple sur les données météorologiques. Les données sur lesquelles nous allons travailler proviennent d'un jeu de données d'étudiants français qui avaient pris cela comme sujet d'examen (Tableau 5.1). En lignes, les individus statistiques sont représentés par les 15 villes de France sélectionnées et en colonnes les températures mensuelles moyennes. Ces températures mensuelles moyennes ont été calculées sur 30 ans. Donc, par exemple, à Bordeaux en Janvier, il fait en moyenne 5.6 degrés. Cette valeur de 5.6 degrés est la moyenne sur tous les jours de Janvier pendant 30 ans. On a ainsi 12 variables correspondants au 12 mois de l'année. On retrouve également en colonnes deux variables liées à la position géographique des villes (Latitude et Longitude).

```python
# Chargement des données
import pandas as pd
Data = pd.read_excel("./donnee/temperature_acp.xlsx",sheet_name=0,index_col=0)
```

Table 5.1 – Données - Température des villes françaises

	Jan	Fev	Mars	Avril	Mai	Juin	Juil	Août	Sept	Oct	Nov	Dec	moy	amp	Lati	Long	groupe
Bordeaux	5.6	6.6	10.3	12.8	15.8	19.3	20.9	21.0	18.6	13.8	9.1	6.2	13.333333	15.4	44.50	-0.34	C
Brest	6.1	5.8	7.8	9.2	11.6	14.4	15.6	16.0	14.7	12.0	9.0	7.0	10.766667	10.2	48.24	-4.29	A
Clermont	2.6	3.7	7.5	10.3	13.8	17.3	19.4	19.1	16.2	11.2	6.6	3.6	10.941667	16.8	45.47	3.05	B
Grenoble	1.5	3.2	7.7	10.6	14.5	17.8	20.1	19.5	16.7	11.4	6.5	2.3	10.983333	18.6	45.10	5.43	B
Lille	2.4	2.9	6.0	8.9	12.4	15.3	17.1	17.1	14.7	10.4	6.1	3.5	9.733333	14.7	50.38	3.04	B
Lyon	2.1	3.3	7.7	10.9	14.9	18.5	20.7	20.1	16.9	11.4	6.7	3.1	11.358333	18.6	45.45	4.51	B
Marseille	5.5	6.6	10.0	13.0	16.8	20.8	23.3	22.8	19.9	15.0	10.2	6.9	14.233333	17.8	43.18	5.24	C
Montpellier	5.6	6.7	9.9	12.8	16.2	20.1	22.7	22.3	19.3	14.6	10.0	6.5	13.891667	17.1	43.36	3.53	C
Nantes	5.0	5.3	8.4	10.8	13.9	17.2	18.8	18.6	16.4	12.2	8.2	5.5	11.691667	13.8	47.13	-1.33	A
Nice	7.5	8.5	10.8	13.3	16.7	20.1	22.7	22.5	20.3	16.0	11.5	8.2	14.841667	15.2	43.42	7.15	C
Paris	3.4	4.1	7.6	10.7	14.3	17.5	19.1	18.7	16.0	11.4	7.1	4.3	11.183333	15.7	48.52	2.20	B
Rennes	4.8	5.3	7.9	10.1	13.1	16.2	17.9	17.8	15.7	11.6	7.8	5.4	11.133333	13.1	48.05	-1.41	A
Strasbourg	0.4	1.5	5.6	9.8	14.0	17.2	19.0	18.3	15.1	9.5	4.9	1.3	9.716667	18.6	48.35	7.45	B
Toulouse	4.7	5.6	9.2	11.6	14.9	18.7	20.9	20.9	18.3	13.3	8.6	5.5	12.683333	16.2	43.36	1.26	C
Vichy	2.4	3.4	7.1	9.9	13.6	17.1	19.3	18.8	16.0	11.0	6.6	3.4	10.716667	16.9	46.08	3.26	B
Amsterdam	2.9	2.5	5.7	8.2	12.5	14.8	17.1	17.1	14.5	11.4	7.0	4.4	NaN	NaN	NaN	NaN	NA
Anvers	3.1	2.9	6.2	8.9	12.9	15.5	17.9	17.6	14.7	11.5	6.8	4.7	NaN	NaN	NaN	NaN	NA
Athènes	9.1	9.7	11.7	15.4	20.1	24.5	27.4	27.2	23.8	19.2	14.6	11.0	NaN	NaN	NaN	NaN	NA
Barcelone	9.1	10.3	11.8	14.1	17.4	21.2	24.2	24.1	21.7	17.5	13.1	10.0	NaN	NaN	NaN	NaN	NA
Berlin	-0.2	0.1	4.4	8.2	13.8	16.0	18.3	18.0	14.4	10.0	4.2	1.2	NaN	NaN	NaN	NaN	NA
Bruxelles	3.3	3.3	6.7	8.9	12.8	15.6	17.8	17.8	15.0	11.1	6.7	4.4	NaN	NaN	NaN	NaN	NA
Budapest	-1.1	0.8	5.5	11.6	17.0	20.2	22.0	21.3	16.9	11.3	5.1	0.7	NaN	NaN	NaN	NaN	NA
Copenhague	-0.4	-0.4	1.3	5.8	11.1	15.4	17.1	16.6	13.3	8.8	4.1	1.3	NaN	NaN	NaN	NaN	NA
Cracovie	-3.7	-2.0	1.9	7.9	13.2	16.9	18.4	17.6	13.7	8.6	2.6	-1.7	NaN	NaN	NaN	NaN	NA
Dublin	4.8	5.0	5.9	7.8	10.4	13.3	15.0	14.6	12.7	9.7	6.7	5.4	NaN	NaN	NaN	NaN	NA

5.2 ACP

Le but général de l'étude est de comparer les températures mensuelles des différentes villes. D'une part du point de vue des villes, on pose les questions suivantes : Quelles sont les villes qui se ressemblent vis-à-vis de l'ensemble des variables (les mois). Quelles sont celles qui diffèrent. Plus généralement, peut-on faire une typologie des villes mettant en évidence l'ensemble des ressemblances ainsi définies ? D'autre part, du point de vue des mois : Quels mois sont corrélés entre eux ? Quels sont ceux qui le sont peu ? Plus généralement, peut-on faire un bilan de corrélation entre les 12 mois ? Les températures mensuelles sont-elles liées à la position géographique (variables supplémentaires) ?

5.2.1 Chargement de scientisttools

```python
from scientisttools import PCA
```

On crée une instance de la classe PCA, en lui passant ici des étiquettes pour les lignes et les variables. Ces paramètres sont facultatifs ; en leur absence, le programme détermine automatiquement des étiquettes.

Le constructeur de la classe PCA possède un paramètre `normalize` qui indique si l'ACP est réalisée :

- à partir de données centrées et réduites -> PCA(normalize=True)
- à partir de données centrées mais non réduites -> PCA(normalize=False)

Par défaut, la valeur du paramètre `normalize` est fixée à `True`, car c'est le cas le plus courant.

Réalisez l'ACP sur tous les individus (actifs et supplémentaires) et les variables (actives et supplémentaires)en tapant la ligne de code suivante :

```
# ACP - Instanciation
res_pca = PCA(ind_sup=list(range(15,Data.shape[0])),
              quanti_sup=list(range(12,16)),quali_sup=16,parallelize=True)
res_pca.fit(Data)
```

```
## Missing values are imputed by the mean of the variable.
## PCA(ind_sup=[15, 16, 17, 18, 19, 20, 21, 22, 23, 24], parallelize=True,
##     quali_sup=16, quanti_sup=[12, 13, 14, 15])
```

5.3 HCPC

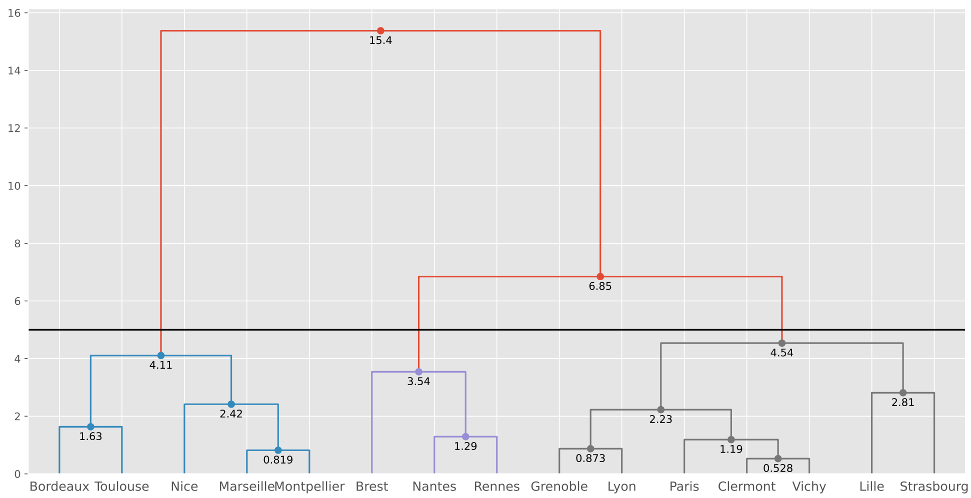
La première étape consistait à réaliser une ACP du tableau de données. On réalise ensuite la classification hiérarchique. Nous demandons une partition en 3 classes.

```
from scientisttools import HCPC
res_hcpc = HCPC(res_pca,n_clusters=3)
```

5.3.1 Dendrogram

L'arbre hiérarchique nous montre notre partition en trois classes.

```
# Plot dendodgram
from scientisttools import plot_dendrogram
import matplotlib.pyplot as plt
fig,axe = plt.subplots(figsize=(16,8))
plot_dendrogram(res_hcpc,ax=axe,max_d=5)
plt.show()
```



5.3.2 Plan factoriel

Le plan factoriel où les individus sont coloriés en fonction de la classe à laquelle ils appartiennent est le suivant :

```
# Plan factoriel
from plotnine import *
from scientisttools import fviz_hcpc_cluster
p = (fviz_hcpc_cluster(res_hcpc,add_ellipse=False,repel=False,
                       show_clust_cent=False,center_marker_size=5)+theme_gray()+
    theme(legend_direction="vertical",legend_position=(0.8,0.8)))
print(p)
```

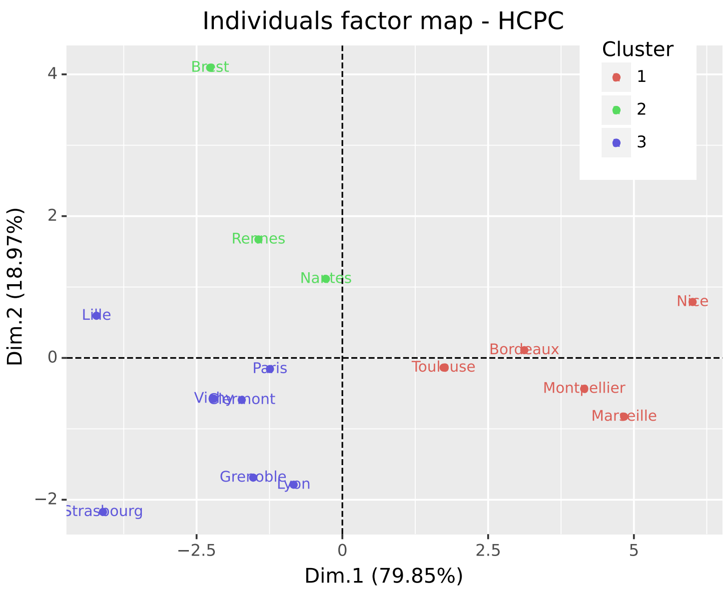


Figure 5.1 – Plan factoriel

```
# Statistiques sur les classes
cinfos =  res_hcpc.cluster_["effectif"].to_frame().reset_index()
```

Table 5.2 – Statistiques sur les classes

clust	effectif
1	5
2	3
3	7

En creusant plus en profondeur, on a la composition suivante :

— La classe 1 (Les 5 villes méridionales) : Bordeaux, Marseille, Montpellier, Nice et Toulouse.
— La classe 2 (les 3 villes les plus occidentales - à faible amplitude thermique) : Brest, Nantes et Rennes
— La classe 3 (les 7 villes à forte amplitude thermique) : Clermont, Grenoble, Lille, Lyon, Paris, Strasbourg et Vichy.

5.4 Description des classes

Les classes peuvent être décrites par :

— les variables et/ou des modalités
— les axes factoriels
— les individus

5.4.1 Coordonnées des classes

```
# Centre de gravité des classes
gclasse = res_hcpc.cluster_["coord"].reset_index()
```

Table 5.3 – Centre de gravité

clust	Dim.1	Dim.2	Dim.3	Dim.4	Dim.5
1	3.968786	-0.1003350	0.0134384	-0.0144852	0.0095143
2	-1.329316	2.2929976	-0.0885413	0.0346947	0.0168230
3	-2.265141	-0.9110454	0.0283474	-0.0045226	-0.0140058

5.4.2 Corrélation entre classe et variables

```
# Rapport de correlation
eta2 = res_hcpc.desc_var_["quanti_var"]
```

Table 5.4 – Rapport de corrélation

	Eta2	pvalue
moy	0.8365869	0.0000190
Oct	0.8362199	0.0000193
Sept	0.8300795	0.0000241
Fev	0.8227293	0.0000310
Mars	0.8126389	0.0000433
Jan	0.8117941	0.0000444
Nov	0.8082999	0.0000496
Avril	0.7928986	0.0000789
Dec	0.7870853	0.0000932
Août	0.7863781	0.0000950
Juin	0.7241197	0.0004409
Mai	0.7163772	0.0005205
Juil	0.7156365	0.0005287
amp	0.6464252	0.0019538
Lati	0.6395987	0.0021914
Long	0.6016129	0.0039979

5.4.3 Description par variables

```
# Description par les variables quantitatives
vardesc = res_hcpc.desc_var_["quanti"]
#Cluster 1
vardesc["1"]
```

```
##             vtest  Mean in category  ...  Overall sd    pvalue
## Sept    3.398358         19.280000  ...    1.785447  0.000678
## moy     3.387913         13.796667  ...    1.548427  0.000704
## Avril   3.329339         12.700000  ...    1.366846  0.000871
## Oct     3.322273         14.540000  ...    1.767937  0.000893
## Mars    3.235769         10.040000  ...    1.477235  0.001213
## Août    3.176011         21.900000  ...    1.943765  0.001493
## Juin    3.003021         19.800000  ...    1.732692  0.002673
## Mai     2.997205         16.080000  ...    1.453578  0.002725
## Nov     2.965719          9.880000  ...    1.742591  0.003020
## Juil    2.915782         22.100000  ...    2.056750  0.003548
## Fev     2.882633          6.800000  ...    1.805055  0.003944
## Dec     2.535689          6.660000  ...    1.892042  0.011223
## Jan     2.464888          5.780000  ...    1.939232  0.013706
## Lati   -2.953993         43.564000  ...    2.217038  0.003137
##
## [14 rows x 6 columns]
```

```
#Cluster 2
vardesc["2"]
```

```
##            vtest  Mean in category  ...  Overall sd    pvalue
## Mai   -2.016380         12.866667  ...    1.453578  0.043760
## Août -2.021201         17.466667  ...    1.943765  0.043259
## Juin -2.051475         15.933333  ...    1.732692  0.040221
## Juil -2.183050         17.433333  ...    2.056750  0.029032
## Long -2.875244         -2.343333  ...    3.205624  0.004037
## amp   -2.952095         12.366667  ...    2.247626  0.003156
##
## [6 rows x 6 columns]
```

```
#Cluster 3
vardesc["3"]
```

```
##             vtest  Mean in category  ...  Overall sd    pvalue
## Sept   -2.046173         15.942857  ...    1.785447  0.040739
## Avril -2.107041         10.157143  ...    1.366846  0.035114
## moy    -2.603897         10.661905  ...    1.548427  0.009217
## Oct    -2.811187         10.900000  ...    1.767937  0.004936
## Mars   -2.854433          7.028571  ...    1.477235  0.004311
## Nov    -3.152395          6.357143  ...    1.742591  0.001619
## Fev    -3.250132          3.157143  ...    1.805055  0.001154
## Dec    -3.283930          3.071429  ...    1.892042  0.001024
## Jan    -3.355280          2.114286  ...    1.939232  0.000793
##
## [9 rows x 6 columns]
```

Nous les résumons en 3 points :

1. Les individus de la classe 1 sont caractérisés par une température élevée toute l'année, particulièrement en demi - saison. Ces villes sont méridionaales (faible latitude).

2. « A l'opposé », les individus de la classe 3 sont caractérisés par une température faible toute l'année, particulièrement pendant les mois les plus froids.
3. La classe 2 comporte des villes présentant une faible amplitude thermique ; elles sont situées à l'ouest (faible longitude).

5.4.4 Description par les axes factoriels

```
# Description par les axes factoriels
res_hcpc.desc_axes_["quanti_var"]
```

```
##            Eta2    pvalue
## Dim.1  0.834734  0.000020
## Dim.2  0.633565  0.002421
```

```
# Description par les axes factoriels
axinfos = res_hcpc.desc_axes_["quanti"]
# Cluster 1
axinfos["1"]
```

```
##           vtest  Mean in category  ...  Overall sd    pvalue
## Dim.1  3.392217          3.968786  ...    3.095445  0.000693
##
## [1 rows x 6 columns]
```

```
# Cluster 1
axinfos['2']
```

```
##           vtest  Mean in category  ...  Overall sd    pvalue
## Dim.2  2.843227          2.292998  ...     1.50878  0.004466
##
## [1 rows x 6 columns]
```

```
# Cluster 3
axinfos["3"]
```

```
##           vtest  Mean in category  ...  Overall sd    pvalue
## Dim.2 -2.113402         -0.911045  ...    1.508780  0.034566
## Dim.1 -2.561180         -2.265141  ...    3.095445  0.010432
##
## [2 rows x 6 columns]
```

Les individus de la classe 1 possèdent de faibles coordonnées sur le premier axe. Ceux de la classe 2 possèdent des coordonnées faibles sur le deuxième axe et les individus de la classe 2 possèdent des coordonnées élevées sur les deux premiers axes.

5.4.5 Description par les individus

Il existe deux types d'individus spécifiques pour décrire les classes :

- — Les individus les plus proches du centre de classe (le parangons)
- — Les individus les plus éloignés des centres des autres classes.

```
# Individu proches
para = res_hcpc.desc_ind_["para"]
para
```

```
## {'Cluster : 1': Montpellier    0.175490
## Bordeaux       1.302643
## Marseille      1.423228
## Nice           5.027258
## Toulouse       5.088993
## Name: distance, dtype: float64, 'Cluster : 2': Rennes    0.410280
## Nantes    2.514486
## Brest     4.182422
## Name: distance, dtype: float64, 'Cluster : 3': Vichy       0.183210
## Clermont    0.446900
## Grenoble    1.401182
## Paris       1.793753
## Lyon        2.822719
## Name: distance, dtype: float64}
```

```
### Individus loins
dist = res_hcpc.desc_ind_["dist"]
dist
```

```
## {'Cluster : 1': Toulouse       5.088993
## Nice           5.027258
## Marseille      1.423228
## Bordeaux       1.302643
## Montpellier    0.175490
## Name: distance, dtype: float64, 'Cluster : 2': Brest     4.182422
## Nantes    2.514486
## Rennes    0.410280
## Name: distance, dtype: float64, 'Cluster : 3': Lille         6.185746
## Strasbourg    5.125671
## Lyon          2.822719
## Paris         1.793753
## Grenoble      1.401182
## Name: distance, dtype: float64}
```

6 Analyse Factorielle Multiple

Sommaire

Ce chapitre a pour objectif de présenter rapidement les principales fonctionnalités offertes par le package « scientisttools » pour réaliser une Analyse Factorielle Multiple.

6.1 AFM Sur variables quantitatives

6.1.1 Importation des données

```
#importation des données
import pandas as pd
url = "http://factominer.free.fr/factomethods/datasets/wine.txt"
wine = pd.read_table(url,sep="\t")

group_name = ["origin","odor","visual","odor.after.shaking","taste","overall"]
group = [2,5,3,10,9,2]
num_group_sup = [0,5]

from scientisttools import MFA

res_mfa = MFA(n_components=5,group=group,group_type=["n"]+["s"]*5,var_weights_mfa=None,
              name_group = group_name,num_group_sup=[0,5],parallelize=True)
res_mfa.fit(wine)
```

```
## MFA(group=[2, 5, 3, 10, 9, 2], group_type=['n', 's', 's', 's', 's', 's'],
##     name_group=['origin', 'odor', 'visual', 'odor.after.shaking', 'taste',
##                 'overall'],
##     num_group_sup=[0, 5], parallelize=True)
```

6.1.2 Valeurs propres

```
# Valeurs propres
res_mfa.eig_
```

```
##          eigenvalue  difference  proportion  cumulative
## Dim.1      3.461950    2.095182   49.378382   49.378382
## Dim.2      1.366768    0.751339   19.494446   68.872829
## Dim.3      0.615429    0.243229    8.777969   77.650797
## Dim.4      0.372200    0.101817    5.308747   82.959544
## Dim.5      0.270382    0.067979    3.856511   86.816055
## Dim.6      0.202403    0.026690    2.886912   89.702967
## Dim.7      0.175713    0.049815    2.506230   92.209197
## Dim.8      0.125899    0.020623    1.795714   94.004911
## Dim.9      0.105276    0.026484    1.501563   95.506474
## Dim.10     0.078791    0.004899    1.123812   96.630286
## Dim.11     0.073892    0.013554    1.053940   97.684226
## Dim.12     0.060338    0.031633    0.860617   98.544843
## Dim.13     0.028705    0.006742    0.409424   98.954268
## Dim.14     0.021963    0.002799    0.313256   99.267523
## Dim.15     0.019164    0.008224    0.273339   99.540862
## Dim.16     0.010940    0.001784    0.156043   99.696906
## Dim.17     0.009156    0.002785    0.130594   99.827499
## Dim.18     0.006371    0.003068    0.090870   99.918369
## Dim.19     0.003303    0.000884    0.047117   99.965487
## Dim.20     0.002420         NaN    0.034513  100.000000
```

6.1.3 Information sur les individus

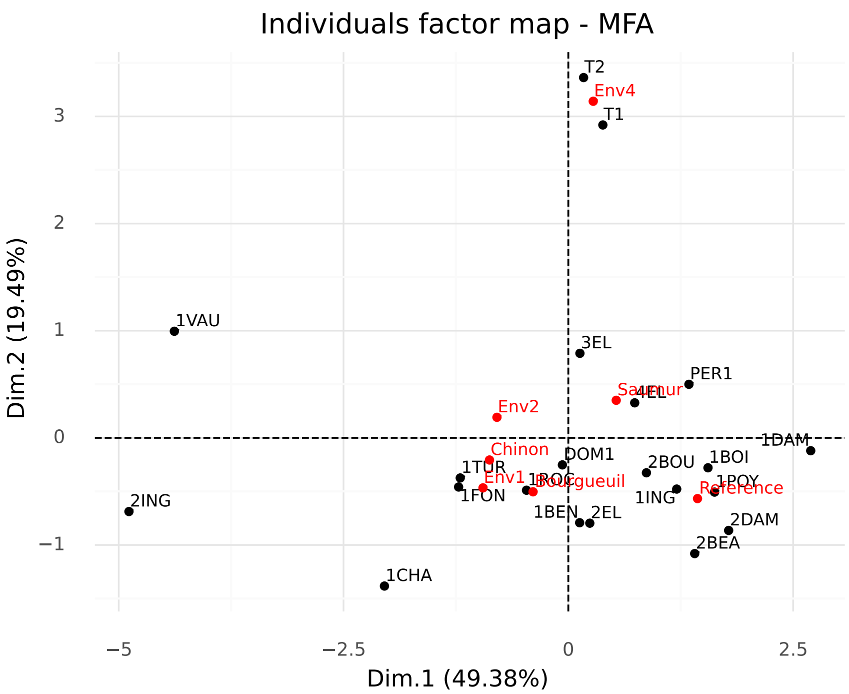
```
ind = res_mfa.ind_
```

6.1.3.1 Coordonnées factorielles

```
ind["coord"].head(6)
```

```
##           Dim.1     Dim.2     Dim.3     Dim.4     Dim.5
## 2EL    0.238874 -0.796677  0.935737  0.524407 -0.351492
## 1CHA  -2.044793 -1.383315  1.513530  0.729589  0.071290
## 1FON  -1.220141 -0.459020  0.062333 -1.036356  0.717976
## 1VAU  -4.381299  0.994551 -0.033460  0.310046  0.477621
## 1DAM   2.695771 -0.120330 -0.689965  0.830386  0.816247
## 2BOU   0.868637 -0.326270  0.391083 -1.274204  0.070273
```

```
from scientisttools import fviz_mfa_ind
print(fviz_mfa_ind(res_mfa,ind_sup=False,repel=True))
```

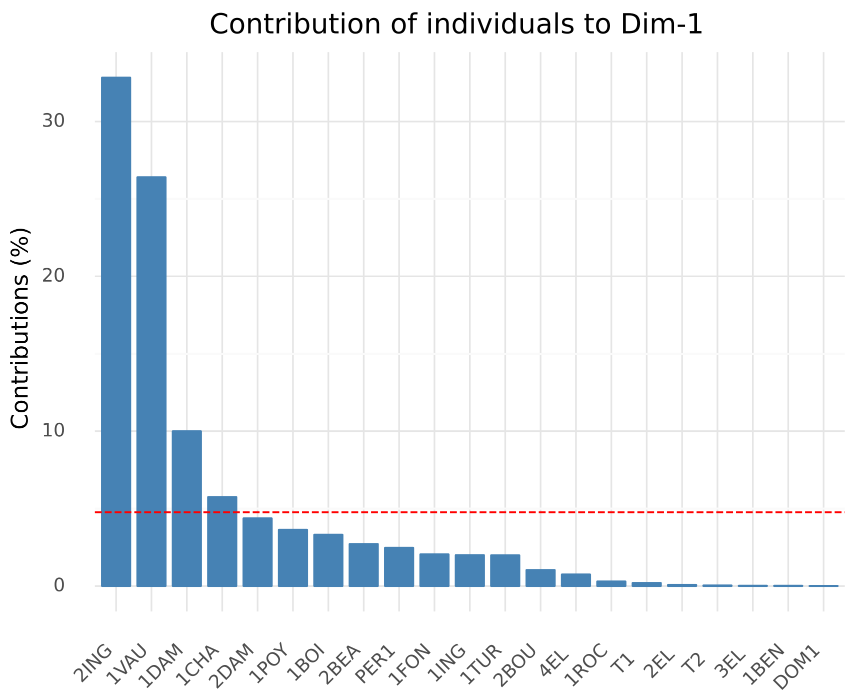


6.1.3.2 Contributions

```
ind["contrib"].head(6)
```

```
##            Dim.1     Dim.2      Dim.3      Dim.4      Dim.5
## 2EL     0.078487  2.211316   6.775010   3.518366   2.175870
## 1CHA    5.751202  6.666957  17.724946   6.810224   0.089507
## 1FON    2.047764  0.734090   0.030064  13.741141   9.078656
## 1VAU   26.403755  3.446198   0.008663   1.229865   4.017628
## 1DAM    9.995994  0.050446   3.683467   8.821961  11.733993
## 2BOU    1.037856  0.370886   1.183427  20.772205   0.086973
```

```
from scientisttools import fviz_contrib
print(fviz_contrib(res_mfa,choice="ind"))
```

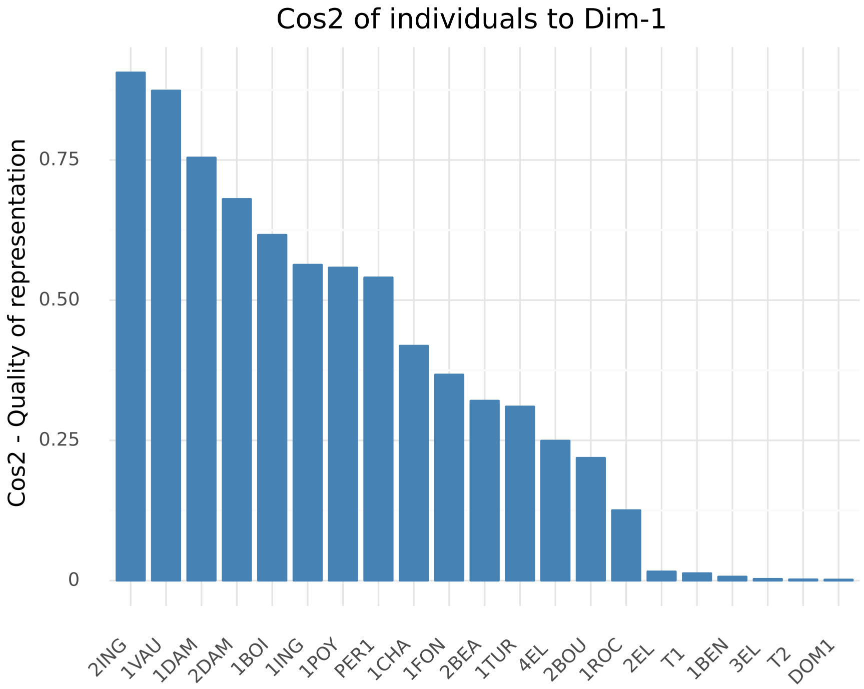


6.1.3.3 Cos2

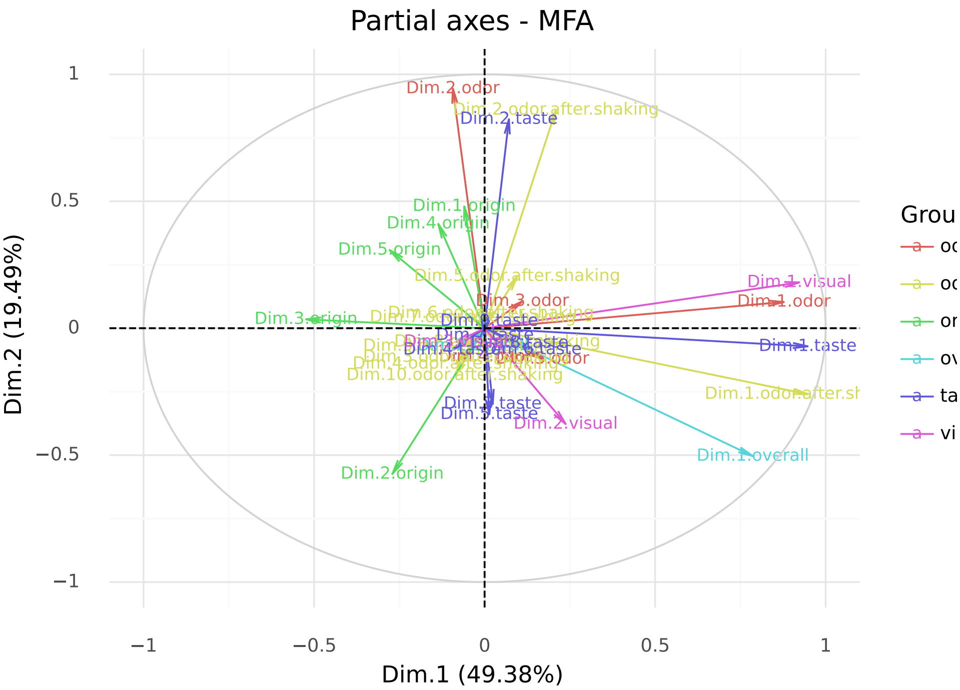
```
ind["cos2"].head(6)
```

```
##          Dim.1     Dim.2     Dim.3     Dim.4     Dim.5
## 2EL   0.016319  0.181524  0.250424  0.078651  0.035335
## 1CHA  0.418802  0.191669  0.229452  0.053317  0.000509
## 1FON  0.367423  0.052001  0.000959  0.265072  0.127223
## 1VAU  0.873760  0.045024  0.000051  0.004376  0.010384
## 1DAM  0.754361  0.001503  0.049416  0.071577  0.069160
## 2BOU  0.218894  0.030882  0.044371  0.471016  0.001433
```

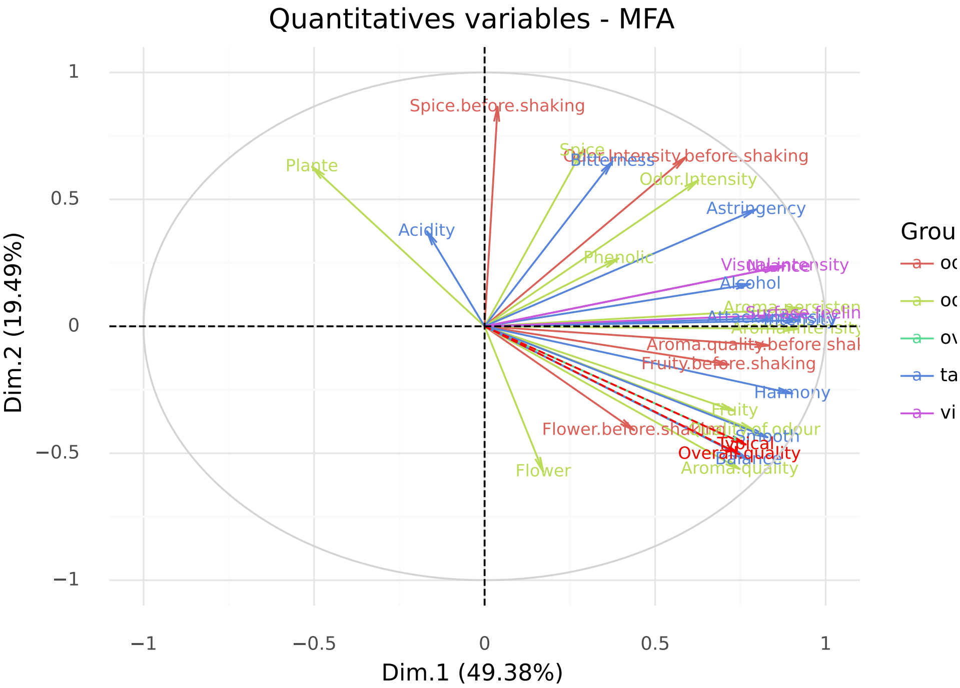
```
from scientisttools import fviz_cos2
print(fviz_cos2(res_mfa,choice="ind"))
```



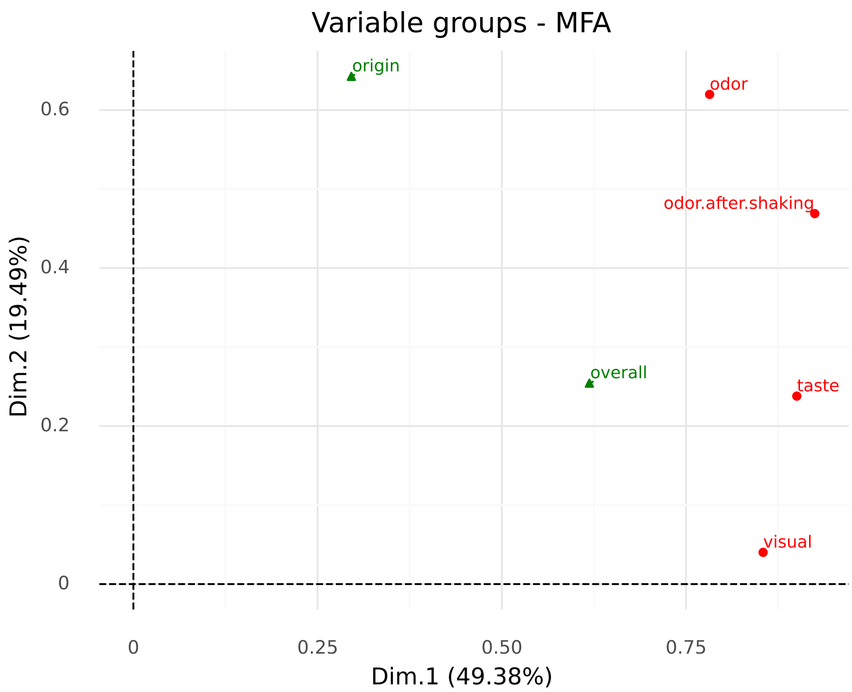

```
from scientisttools import fviz_mfa_axes
p = fviz_mfa_axes(res_mfa)
print(p)
```




```
from scientisttools import fviz_mfa_var
p = fviz_mfa_var(res_mfa)
print(p)
```




```python
from scientisttools import fviz_mfa_group
p = fviz_mfa_group(res_mfa,repel=True)
print(p)
```




```python
from scientisttools import summaryMFA
summaryMFA(res_mfa)
```

```
##                          Multiple Factor Analysis - Results
```

```
##
## Importance of components
##                          Dim.1   Dim.2   Dim.3   Dim.4   Dim.5
## Variance                 3.462   1.367   0.615   0.372   0.270
## Difference               2.095   0.751   0.243   0.102   0.068
## % of var.               49.378  19.494   8.778   5.309   3.857
## Cumulative of % of var. 49.378  68.873  77.651  82.960  86.816
##
## Groups
##
##                    dist2  Dim.1     ctr   cos2  ...   cos2  Dim.3     ctr   cos2
## odor               1.610  0.782  22.591  0.380  ...  0.239  0.374  60.695  0.087
## visual             1.003  0.855  24.688  0.728  ...  0.002  0.014   2.337  0.000
## odor.after.shaking 1.369  0.925  26.712  0.625  ...  0.161  0.180  29.263  0.024
## taste              1.123  0.900  26.009  0.722  ...  0.050  0.047   7.705  0.002
##
## [4 rows x 10 columns]
##
## Supplementary groups
##
##         dist2  Dim.1   cos2  Dim.2   cos2  Dim.3   cos2
## origin  2.645  0.296  0.033  0.643  0.156  0.196  0.015
## overall 1.007  0.619  0.380  0.254  0.064  0.010  0.000
##
## Individuals (the 10 first)
##
##       dist  weight  inertia  Dim.1     ctr  ...    ctr   cos2  Dim.3     ctr   cos2
## 2EL  1.870   0.048    0.166  0.239   0.078  ...  2.211  0.182  0.936   6.775  0.250
## 1CHA 3.160   0.048    0.475 -2.045   5.751  ...  6.667  0.192  1.514  17.725  0.229
## 1FON 2.013   0.048    0.193 -1.220   2.048  ...  0.734  0.052  0.062   0.030  0.001
## 1VAU 4.687   0.048    1.046 -4.381  26.404  ...  3.446  0.045 -0.033   0.009  0.000
## 1DAM 3.104   0.048    0.459  2.696   9.996  ...  0.050  0.002 -0.690   3.683  0.049
## 2BOU 1.857   0.048    0.164  0.869   1.038  ...  0.371  0.031  0.391   1.183  0.044
## 1BOI 1.978   0.048    0.186  1.553   3.318  ...  0.272  0.020 -0.414   1.324  0.044
## 3EL  2.330   0.048    0.259  0.129   0.023  ...  2.167  0.115  1.858  26.707  0.636
## DOM1 1.533   0.048    0.112 -0.066   0.006  ...  0.222  0.027 -0.459   1.629  0.090
## 1TUR 2.158   0.048    0.222 -1.202   1.987  ...  0.489  0.030 -0.716   3.964  0.110
##
## [10 rows x 12 columns]
##
## Continuous variables (the 10 first)
##
##                              Dim.1    ctr   cos2  ...  Dim.3     ctr   cos2
## Odor.Intensity.before.shaking  0.591  4.497  0.349  ... -0.023   0.039  0.001
## Aroma.quality.before.shaking   0.835  8.989  0.698  ... -0.354   9.092  0.125
## Fruity.before.shaking          0.716  6.606  0.513  ... -0.537  20.939  0.289
## Flower.before.shaking          0.439  2.480  0.192  ...  0.637  29.439  0.406
## Spice.before.shaking           0.038  0.019  0.001  ...  0.128   1.187  0.016
## Visual.intensity               0.881  7.912  0.776  ...  0.141   1.139  0.020
## Nuance                         0.862  7.577  0.744  ...  0.142   1.155  0.020
## Surface.feeling                0.950  9.198  0.903  ... -0.027   0.043  0.001
```

```
## Odor.Intensity                   0.627  2.416  0.393  ...  0.214   1.581  0.046
## Quality.of.odour                 0.791  3.844  0.626  ... -0.221   1.684  0.049
##
## [10 rows x 9 columns]
##
## Supplementary Continuous variables
##
##                  Dim.1   cos2  Dim.2   cos2  Dim.3   cos2
## Overall.quality  0.747  0.558 -0.504  0.254  0.130  0.017
## Typical          0.766  0.586 -0.466  0.217  0.039  0.001
##
## Supplementary categories
##
##              dist  Dim.1   cos2  Dim.2   cos2  Dim.3   cos2
## Bourgueuil  0.934 -0.392  0.176 -0.504  0.291 -0.216  0.054
## Chinon      1.196 -0.877  0.537 -0.207  0.030 -0.322  0.072
## Saumur      0.766  0.533  0.483  0.350  0.209  0.235  0.094
## Env1        1.211 -0.949  0.614 -0.467  0.149  0.455  0.141
## Env2        1.066 -0.794  0.554  0.191  0.032 -0.382  0.129
## Env4        3.188  0.277  0.008  3.141  0.971 -0.062  0.000
## Reference   1.584  1.437  0.823 -0.567  0.128 -0.164  0.011
##
## Supplementary categories (eta2)
##
##        Dim.1  Dim.2  Dim.3  Dim.4  Dim.5
## Label  0.098  0.106  0.101  0.190  0.292
## Soil   0.331  0.826  0.184  0.013  0.132
```