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 groupes 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 :

```
# 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.

```
# 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_`.

```
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.

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

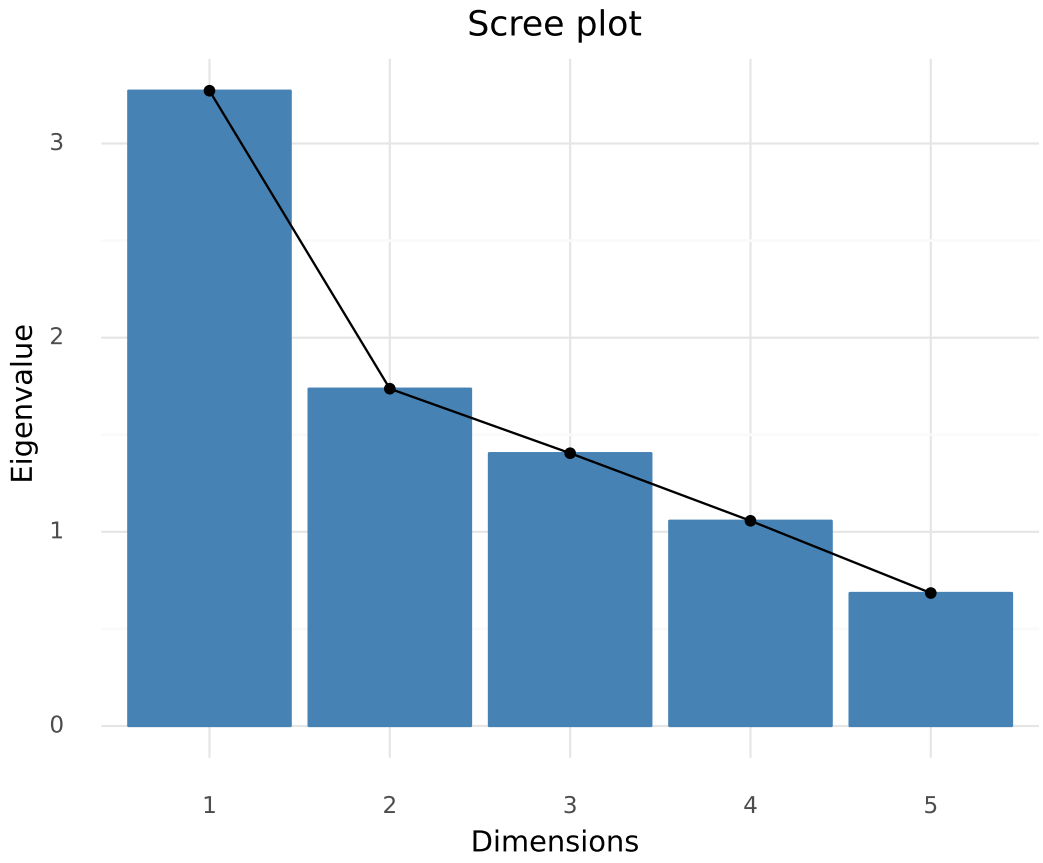
```
##         eigenvalue  difference  proportion  cumulative
## Dim.1     3.271906    1.534775   32.719055   32.719055
## Dim.2     1.737131    0.332214   17.371310   50.090366
## Dim.3     1.404917    0.348066   14.049167   64.139532
## Dim.4     1.056850    0.372077   10.568504   74.708036
## Dim.5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

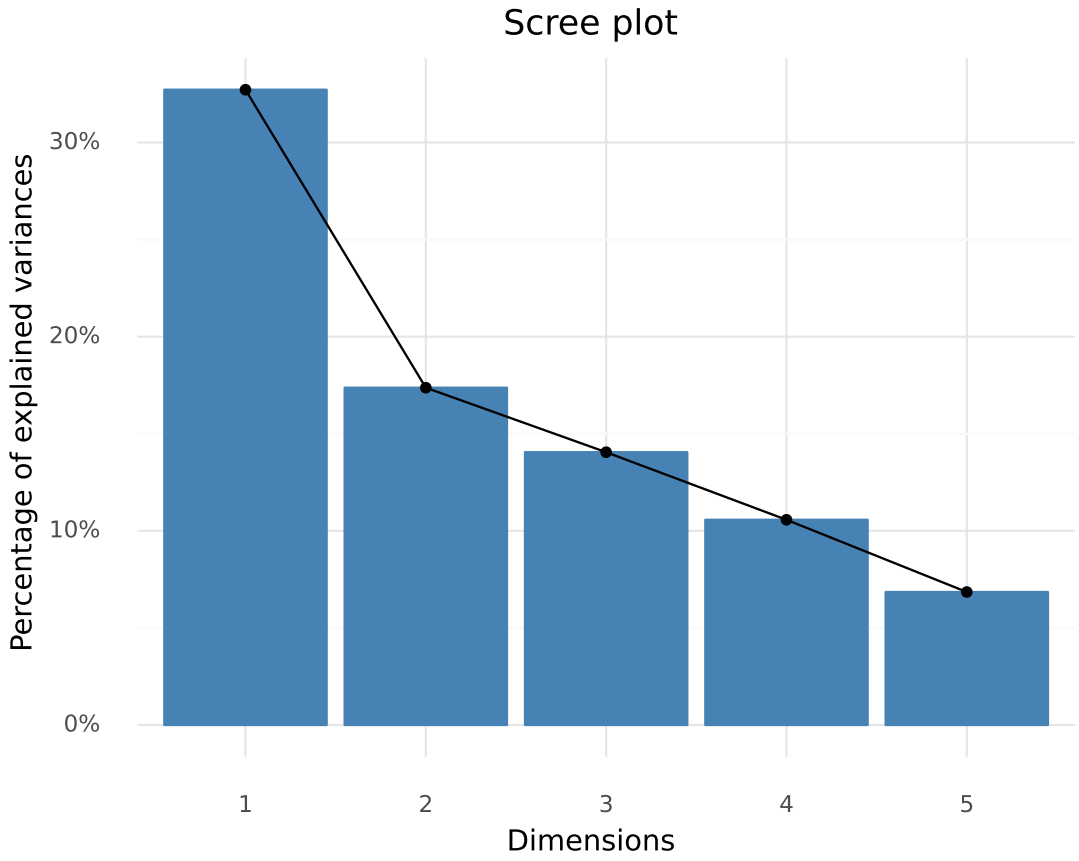
Les deux premières dimensions contiennent 50% de l'inertie totale (l'inertie est la variance totale du tableau de données, i.e. la trace de la matrice des corrélations).

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`.

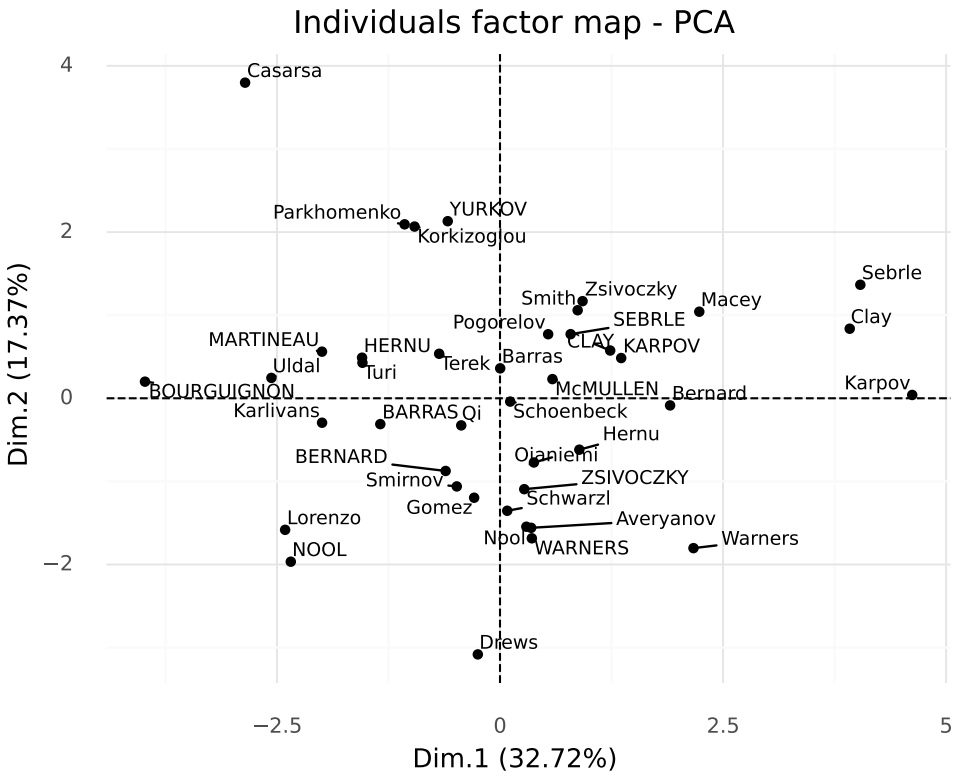
```
from scientisttools import summaryPCA
summaryPCA(res_pca)
```

```
##                      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
```

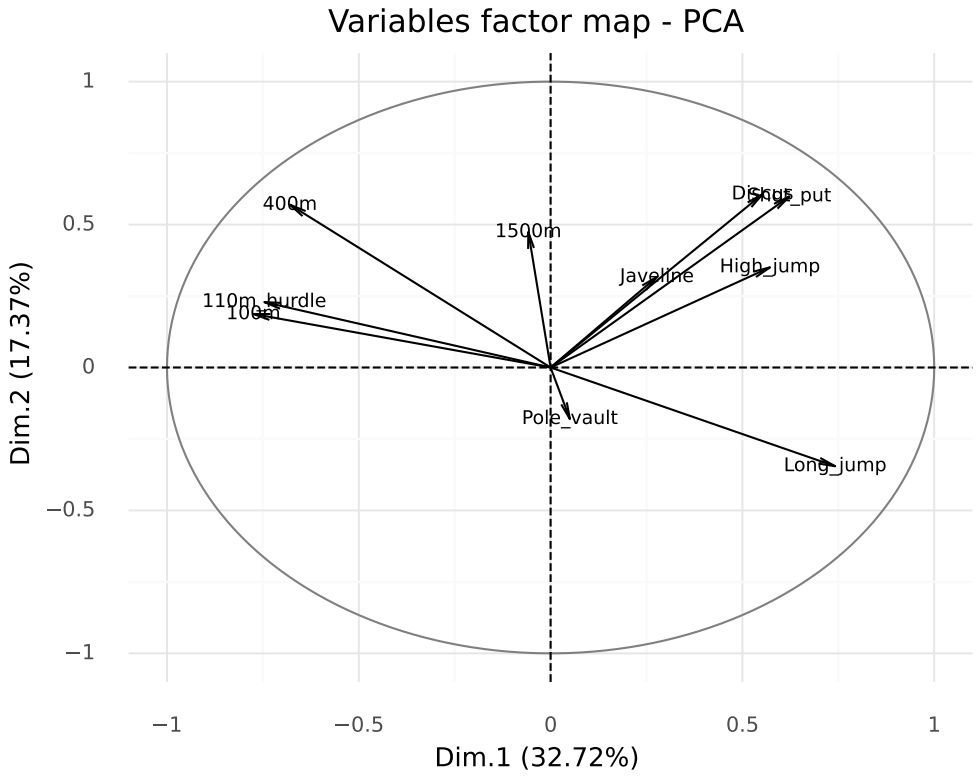
```
## WARNERS    2.428   0.024    0.144  0.357  ...  0.482  0.767  1.020  0.100
## ZSIVOCZKY  2.563   0.024    0.160  0.272  ...  0.182 -1.283  2.857  0.250
## McMULLEN   2.561   0.024    0.160  0.588  ...  0.008 -0.418  0.303  0.027
## MARTINEAU  3.742   0.024    0.342 -1.995  ...  0.022 -0.730  0.925  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

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




```
# 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.

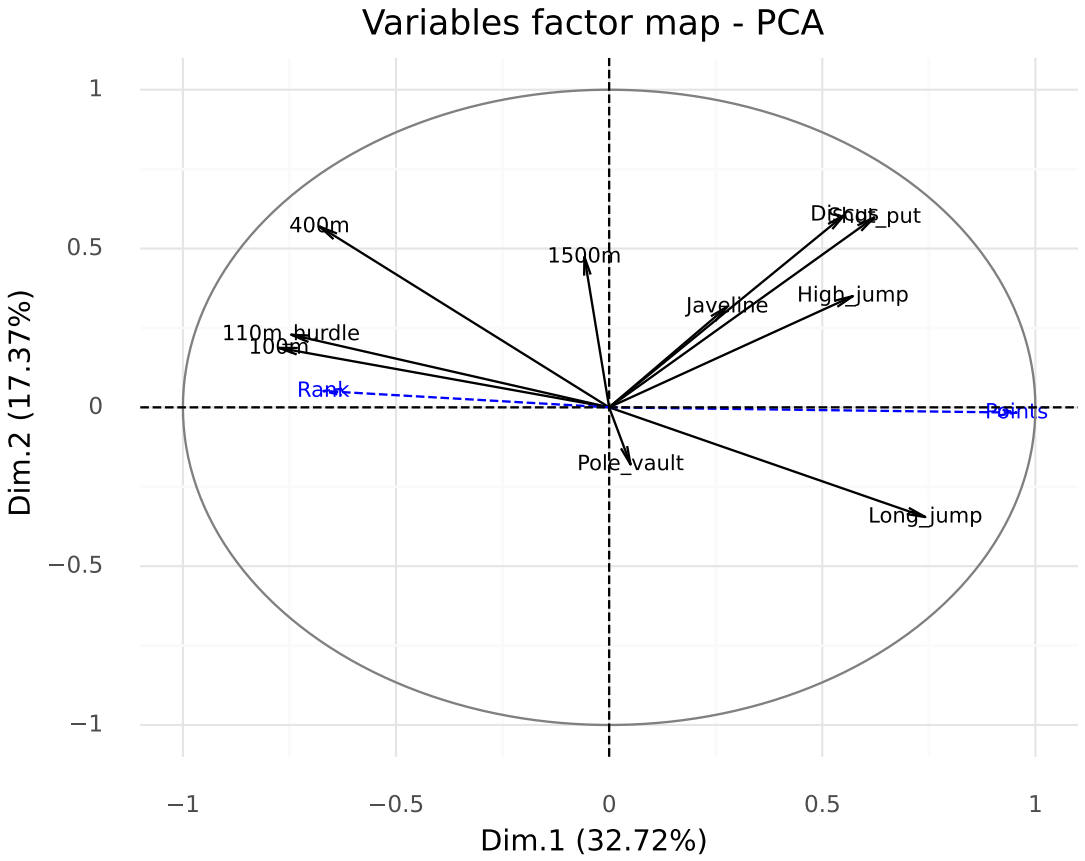
On peut ajouter deux types de variables : continues et qualitatives.

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])
```

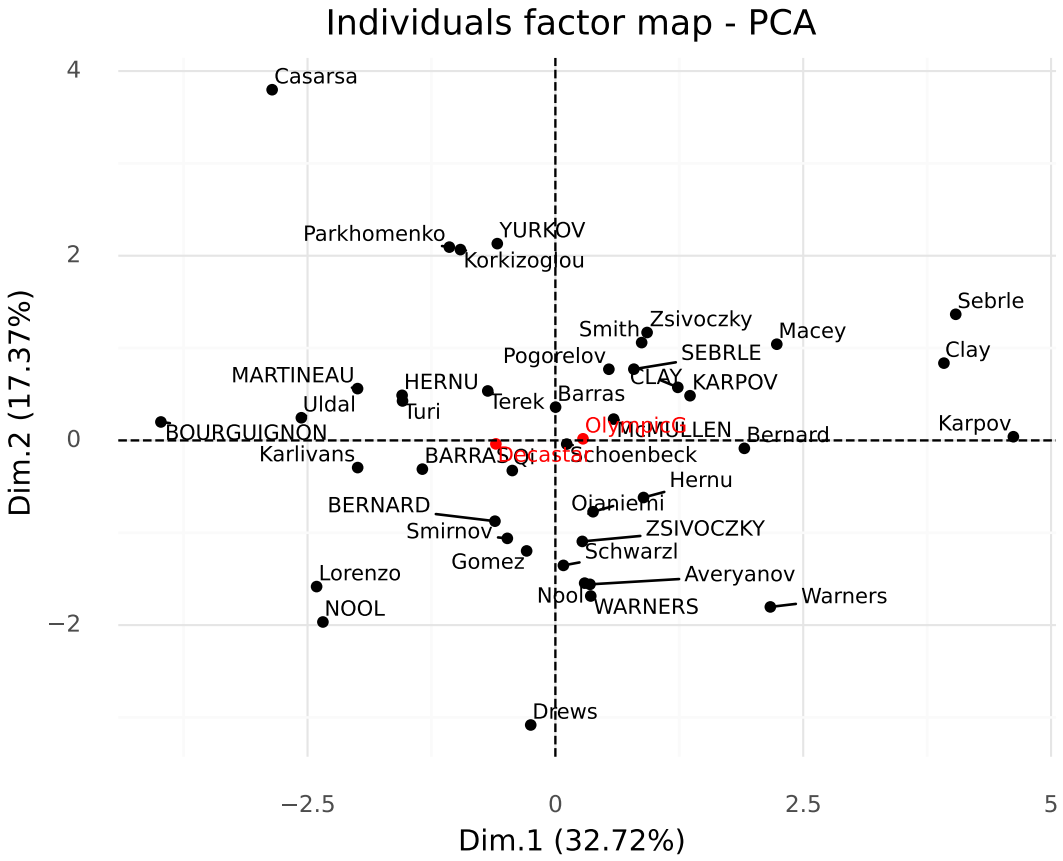
```
print(fviz_pca_var(res_pca))
```



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.

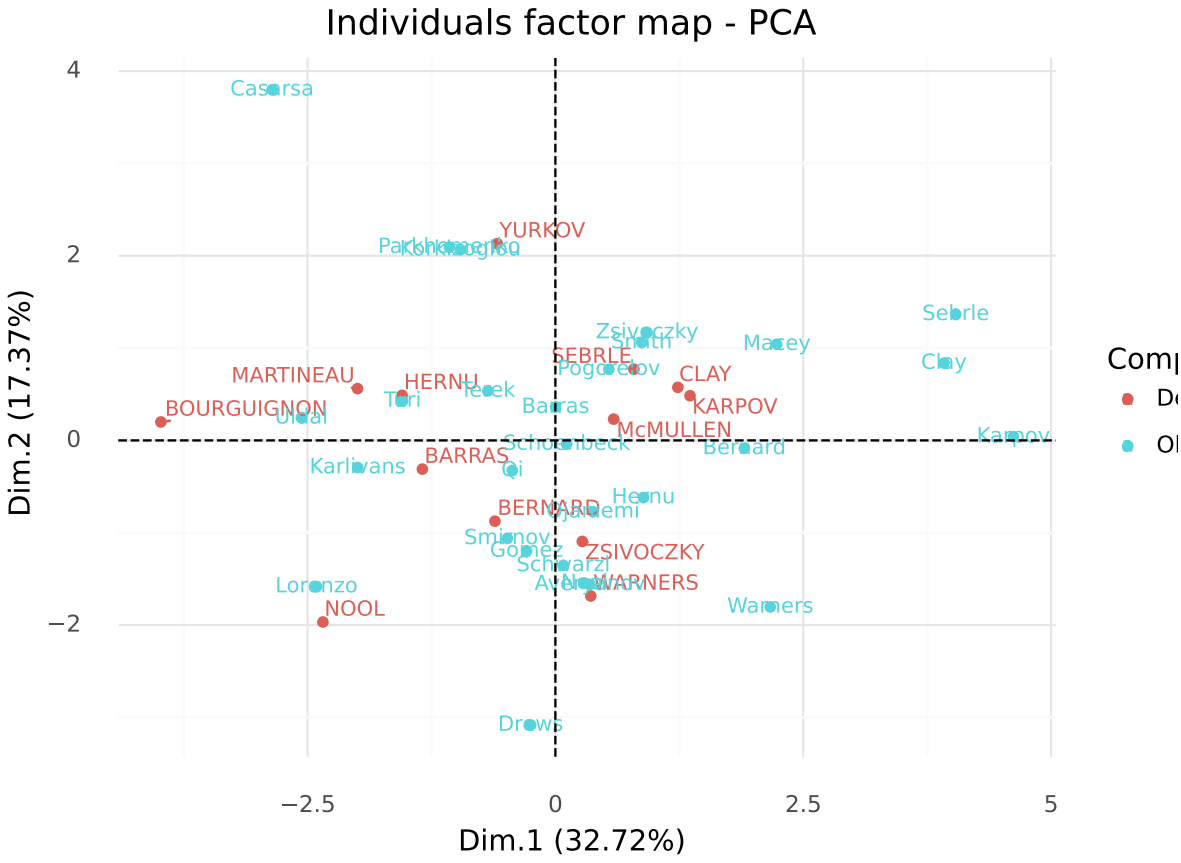
```
print(fviz_pca_ind(res_pca,repel=True))
```



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)
```

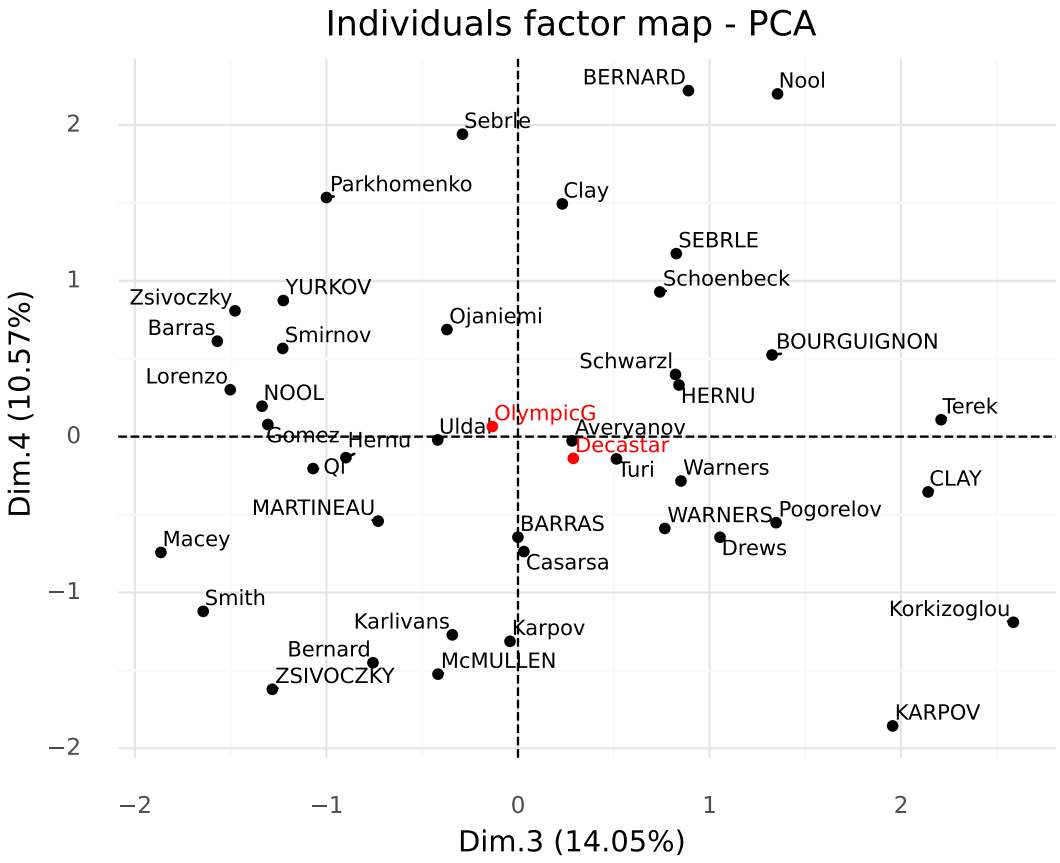
```
##                         Principal Component Analysis - Results
##
```

```
## 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'])
```

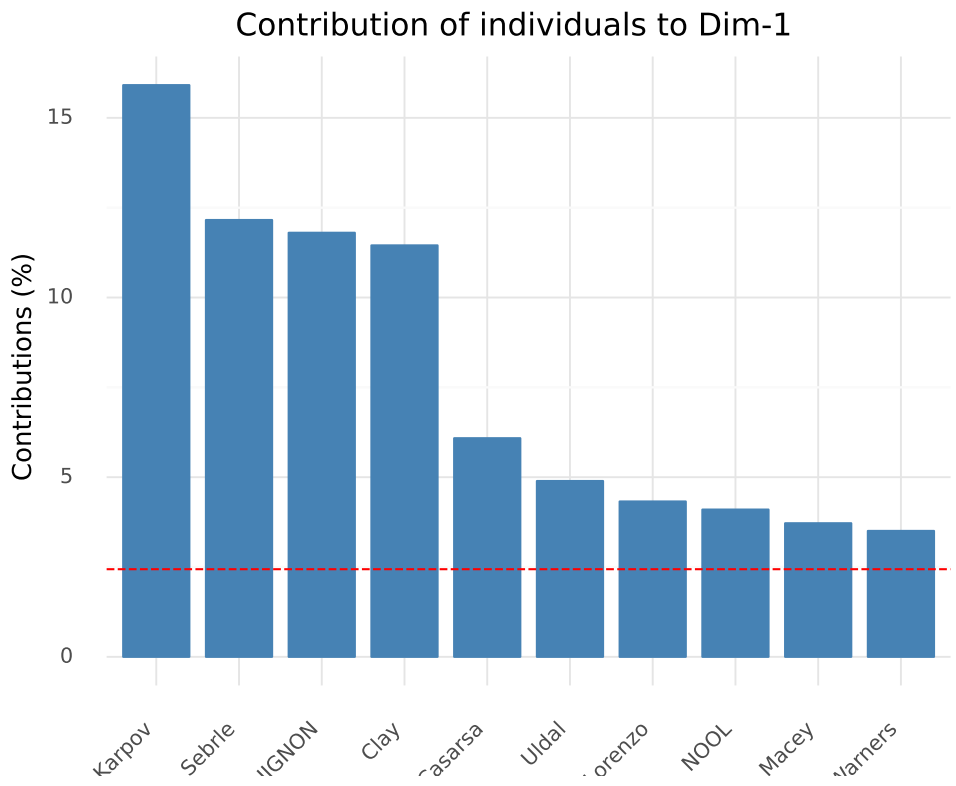
```
var = get_pca_var(res_pca)
print(var.keys())
```

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

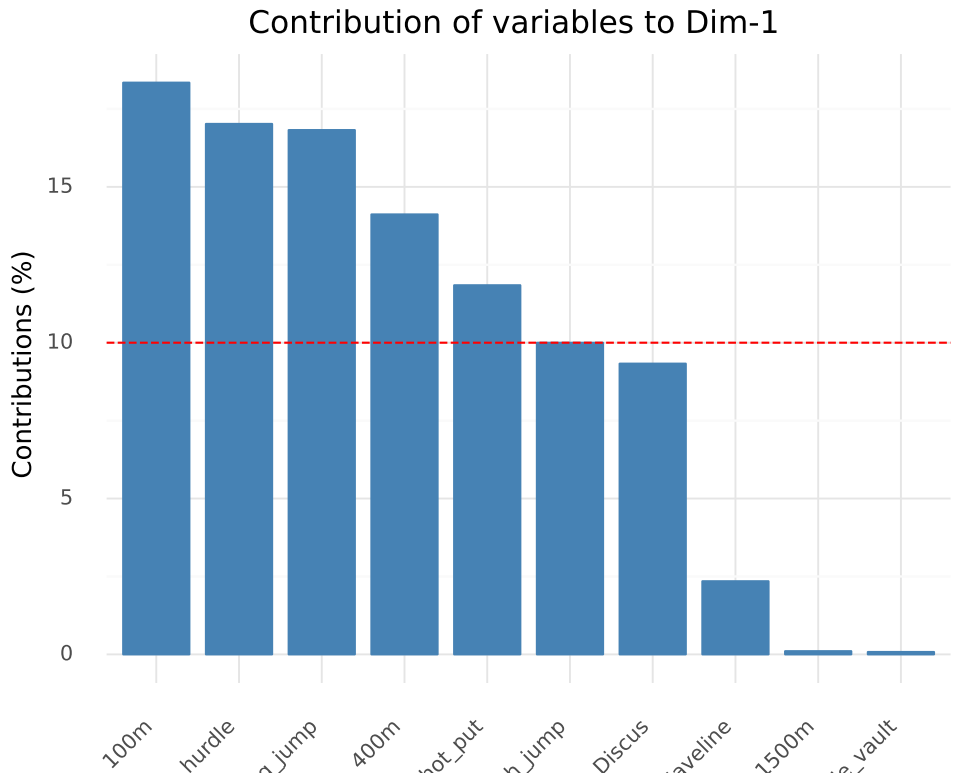
1.5 Interprétation des axes

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.

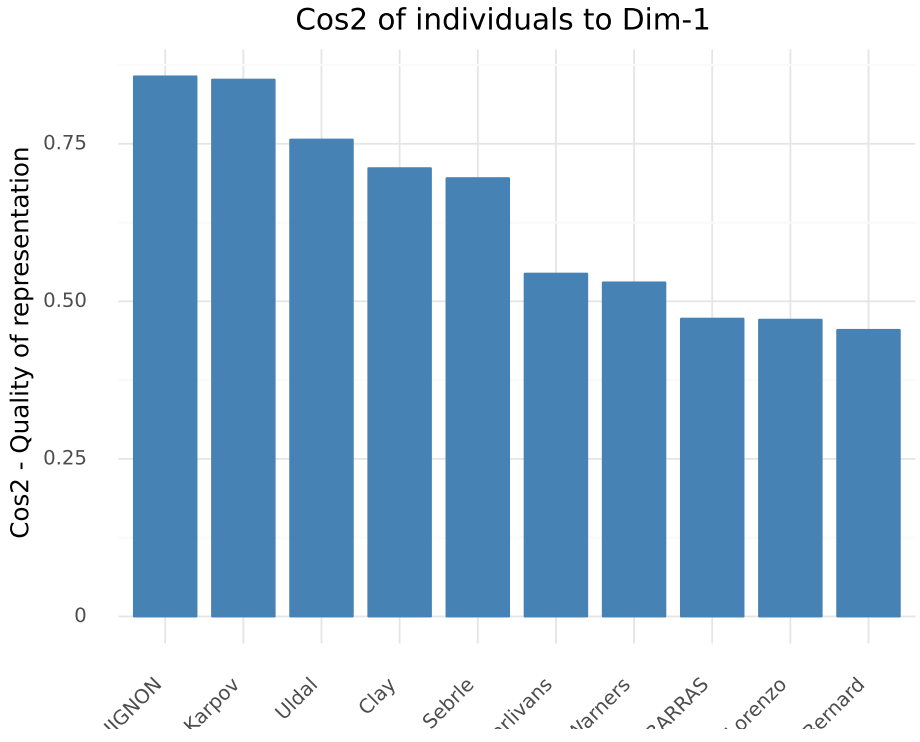
```
# Classement des points lignes en fonction de leur contribution au 1er axe
from scientisttools import fviz_contrib, fviz_cos2
print(fviz_contrib(res_pca,choice="ind",axis=0,top_contrib=10))
```



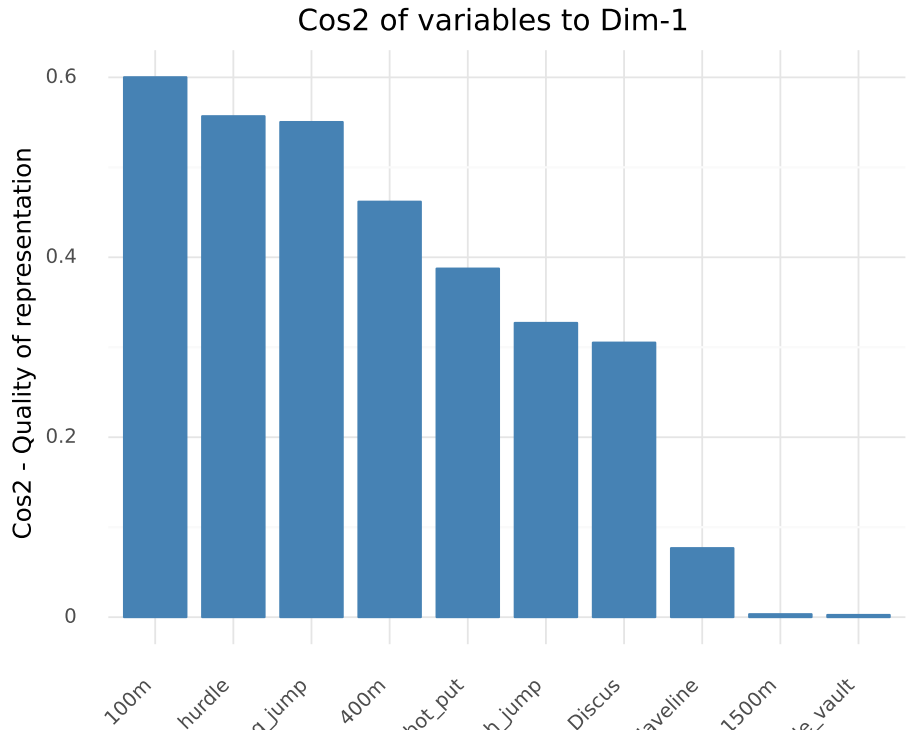

```
# Classement des points colonnes en fonction de leur contribution au 1er axe
print(fviz_contrib(res_pca,choice="var",axis=0))
```




```
# 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))
```



```
# 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.

```
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
```

```
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'], 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