ACM sous Python avec scientisttools

Duvérier DJIFACK ZEBAZE

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

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: 2.3+ KB
```

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

La première colonne du tableau 1 correspond à l'identifiant des observations. Les 6 premières variables sont considérés comme actives : Taille, Poids, Vélocité, Intelligence, Affection, Agressivité. La 7ème variable « Fonction » est considérée comme variable illustrative qualitative tandis que la 8ème 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.

Table 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

Les principales questions auxquelles nous nous posons sont les suivantes :

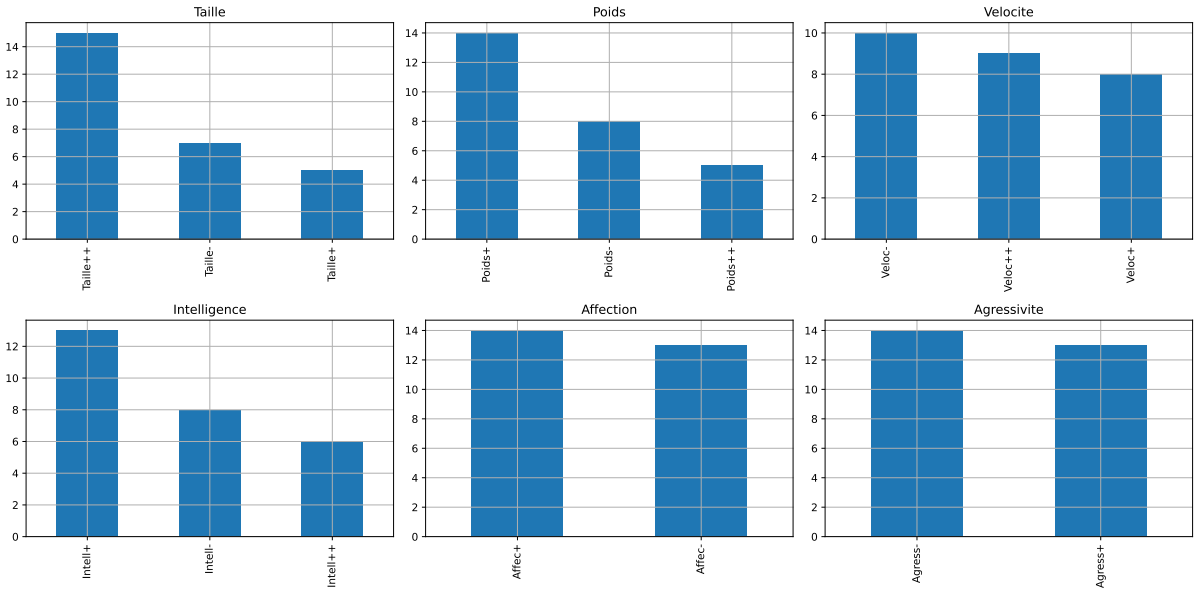
— Quels sont les chiens qui se ressemblent ? Quels sont les chiens qui sont dissemblables ? (proximité entre les individus)
— Sur quels caractères sont fondées ces ressemblances/dissemblances ?
— Quelles sont les associations entre les modalités ? Par exemple, un animal de grande taille est - il plus 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 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 :

```
# 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()
```



ACM

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.

Chargement de scientisttools

```
from scientisttools.decomposition import MCA
```

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(n_components=None,
             row_labels=A.index,
             var_labels=A.columns,
             mod_labels=None,
             matrix_type="completed",
             benzecri=True,
             greenacre=True,
             row_sup_labels=None,
             quali_sup_labels=None,
             quanti_sup_labels=None,
             parallelize=False)
```

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(row_labels=Index(['Beauceron', 'Basset', 'Berger All', 'Boxer', 'Bull-Dog', 'Bull-Mas
##         'Caniche', 'Chihuahua', 'Cocker', 'Colley', 'Dalmatien', 'Doberman',
##         'Dogue All', 'Epag. Breton', 'Epag. Français', 'Fox-Hound',
##         'Fox-Terrier', 'Gd Bleu Gasc', 'Labrador', 'Levrier', 'Mastiff',
##         'Pekinois', 'Pointer', 'St-Bernard', 'Setter', 'Teckel', 'Terre-Neuve'],
##       dtype='object', name='Chien'),
##     var_labels=Index(['Taille', 'Poids', 'Velocite', 'Intelligence', 'Affection',
##         'Agressivite'],
##       dtype='object'))
```

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_)
```

```
## [[4.81606165e-01 3.84737288e-01 2.10954049e-01 1.57554025e-01
##   1.50132670e-01 1.23295308e-01 8.14624601e-02 4.56697566e-02
##   2.35419107e-02 7.71303416e-03]
##  [9.68688769e-02 1.73783238e-01 5.34000239e-02 7.42135546e-03
##   2.68373618e-02 4.18328480e-02 3.57927036e-02 2.21278458e-02
##   1.58288765e-02 7.71303416e-03]
##  [2.88963699e+01 2.30842373e+01 1.26572430e+01 9.45324152e+00
##   9.00796020e+00 7.39771849e+00 4.88774761e+00 2.74018539e+00
```

```
##   1.41251464e+00 4.62782050e-01]
##  [2.88963699e+01 5.19806071e+01 6.46378501e+01 7.40910916e+01
##   8.30990518e+01 9.04967703e+01 9.53845179e+01 9.81247033e+01
##   9.95372180e+01 1.00000000e+02]]
```

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.

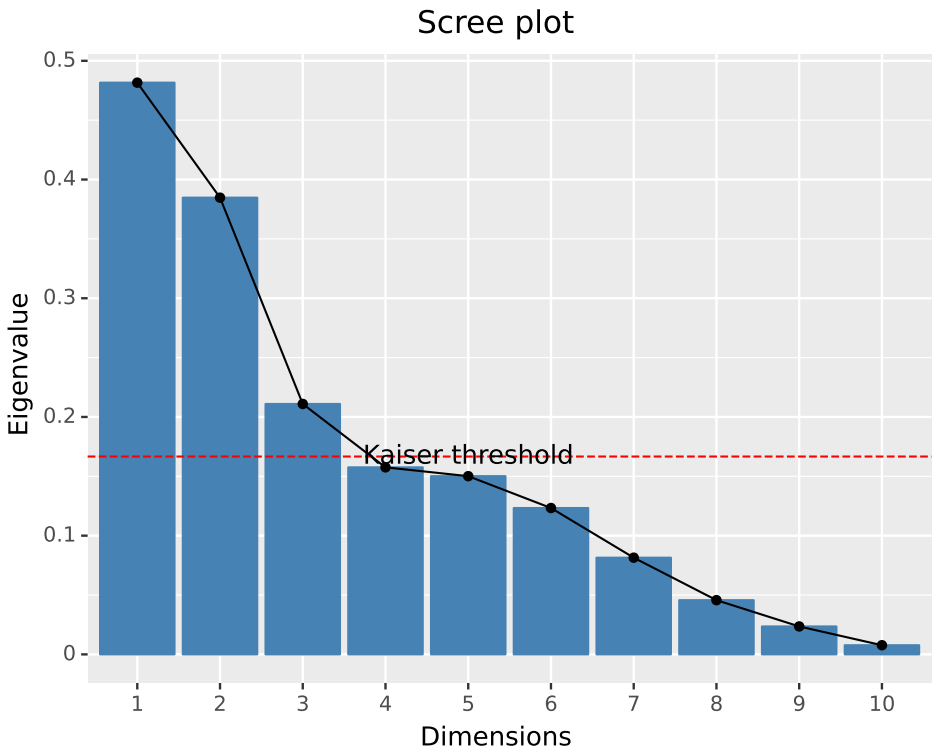
```
# Valeurs propres
from scientisttools.extractfactor import get_eig
print(get_eig(my_mca))
```

```
##          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    0.007713    0.462782  100.000000
```

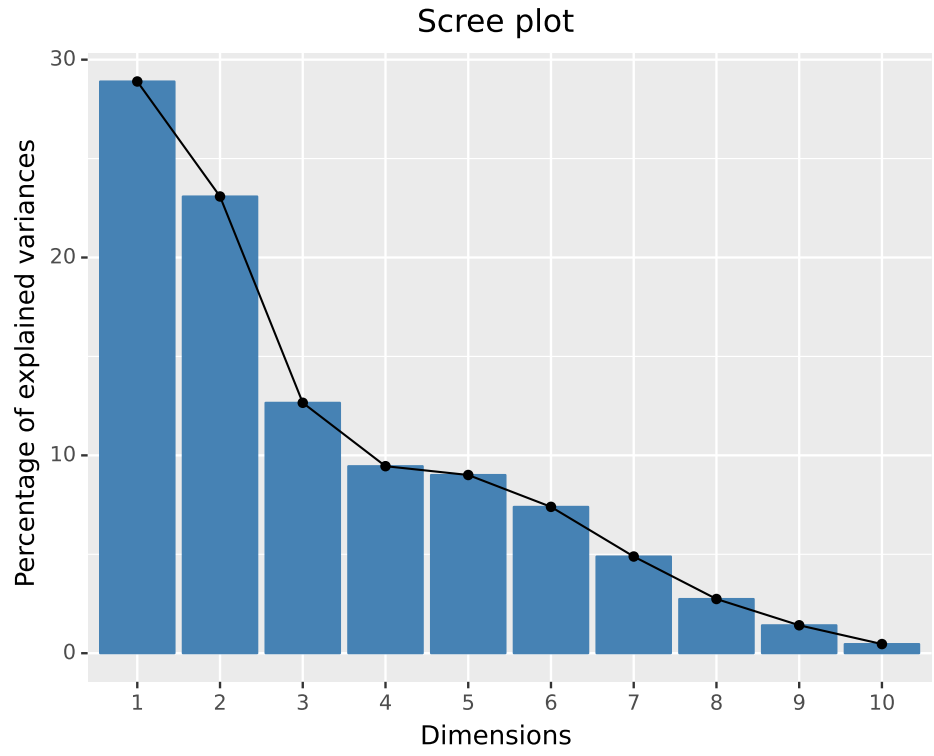
Le nombre de modalités actives est $16(3\times 4+2\times 2)$, ce qui conduit à 10 facteurs et à une inertie totale de $\frac{16}{6}-1=\frac{5}{3}=1.667$.

Les valeurs propres peuvent être représentées graphiquement

```
from scientisttools.ggplot import fviz_eigenvalue
print(fviz_eigenvalue(my_mca,choice="eigenvalue", add_kaiser=True))
```




```
print(fviz_eigenvalue(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.

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
```

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`.

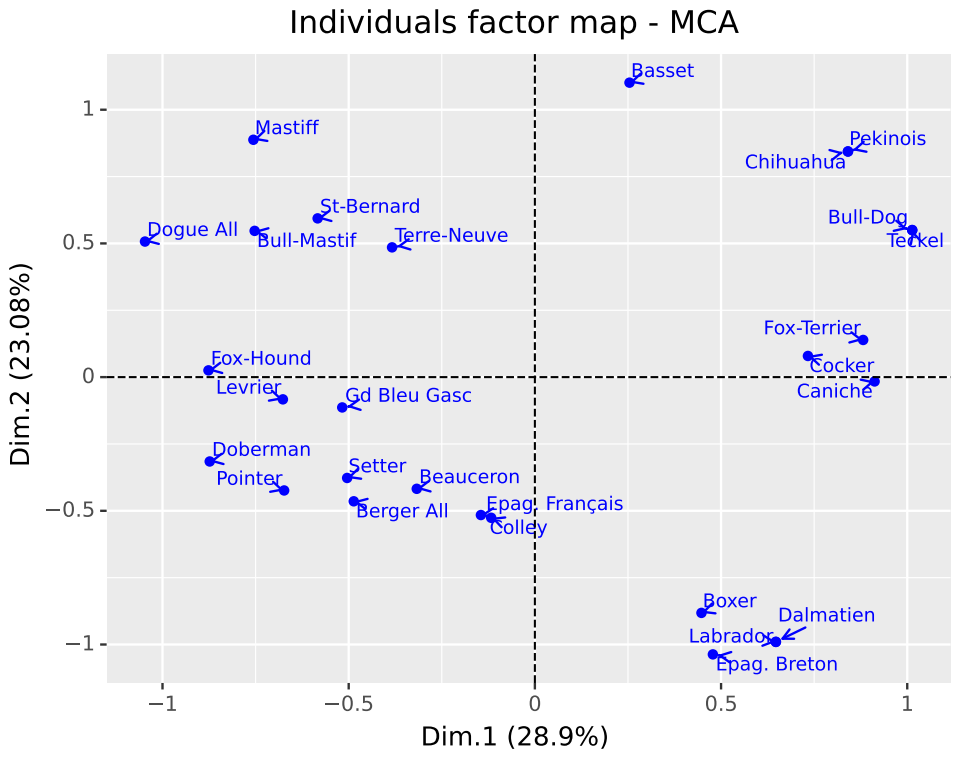
```
from scientisttools.extractfactor import summaryMCA
summaryMCA(my_mca)
```

```
##                      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    0.008
## % 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)
##
##              d(i,G)   p(i)  I(i,G)  Dim.1  ...   cos2  Dim.3    ctr   cos2
## Chien                                       ...
## 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
```

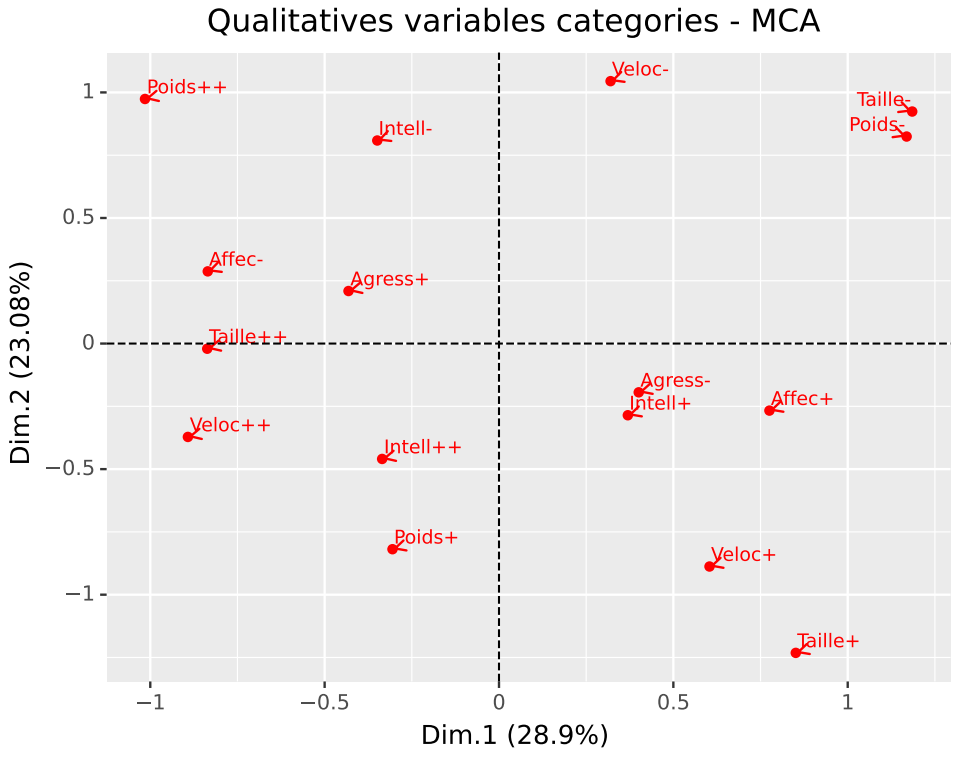
```
## Berger All    1.241  0.037   0.057 -0.486  ...  0.140 -0.498  4.357  0.161
## Boxer         1.341  0.037   0.067  0.447  ...  0.433  0.692  8.408  0.266
## Bull-Dog      1.282  0.037   0.061  1.013  ...  0.184 -0.163  0.469  0.016
## Bull-Mastif   1.446  0.037   0.077 -0.753  ...  0.143  0.498  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
##
##                       d(k,G)   p(k)  I(k,G)  Dim.1  ...  Dim.3     ctr   cos2  vtest
## Taille_Taille+         2.098  0.031   0.136  0.851  ...  1.016  15.104  0.235  2.470
## Taille_Taille++        0.894  0.093   0.074 -0.837  ... -0.051   0.115  0.003 -0.292
## Taille_Taille-         1.690  0.043   0.123  1.185  ... -0.616   7.772  0.133 -1.858
## Poids_Poids+           0.964  0.086   0.080 -0.305  ... -0.231   2.191  0.058 -1.224
## Poids_Poids++          2.098  0.031   0.136 -1.015  ...  1.222  21.833  0.339  2.970
## Poids_Poids-           1.541  0.049   0.117  1.169  ... -0.359   3.013  0.054 -1.187
## Velocite_Veloc+        1.541  0.049   0.117  0.604  ...  0.356   2.972  0.053  1.179
## Velocite_Veloc++       1.414  0.056   0.111 -0.892  ... -0.763  15.335  0.291 -2.751
## Velocite_Veloc-        1.304  0.062   0.105  0.320  ...  0.402   4.722  0.095  1.571
## Intelligence_Intell+   1.038  0.080   0.086  0.369  ...  0.493   9.253  0.226  2.423
##
## [10 rows x 15 columns]
##
## Categorical variables
##
##               I(j,G)  eta2.1    ctr.1  cos2.1  ...  cos2.2  eta2.3    ctr.3  cos2.3
## Taille         0.333   0.887  184.191   0.444  ...   0.251   0.291  137.951   0.146
## Poids          0.333   0.644  133.729   0.322  ...   0.362   0.342  162.226   0.171
## Velocite       0.333   0.411   85.376   0.206  ...   0.342   0.291  138.177   0.146
## Intelligence   0.333   0.127   26.321   0.063  ...   0.140   0.234  110.787   0.117
## Affection      0.167   0.648  134.478   0.648  ...   0.077   0.004    1.887   0.004
## Agressivite    0.167   0.173   35.906   0.173  ...   0.041   0.103   48.972   0.103
##
## [6 rows x 10 columns]
```

Représentation graphique

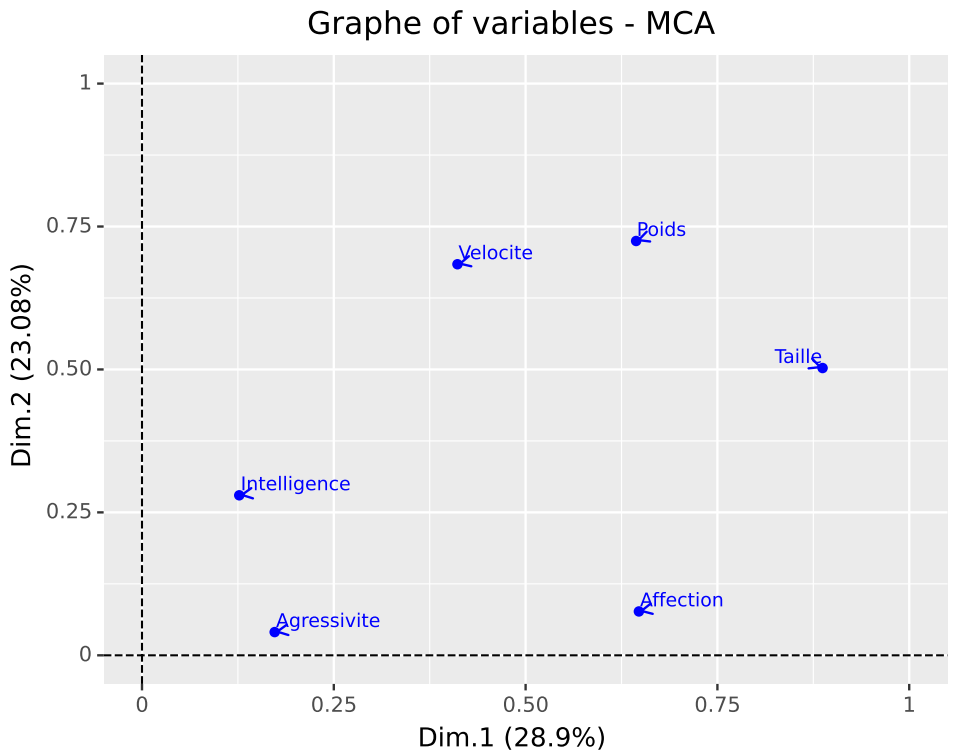
```
# Carte des individus
from scientisttools.ggplot import fviz_mca_ind
print(fviz_mca_ind(my_mca,color="blue",repel=True))
```




```
# Carte des modalités
from scientisttools.ggplot import fviz_mca_mod
print(fviz_mca_mod(my_mca,color="red",repel=True))
```



```
# Carte des variables
from scientisttools.ggplot import fviz_mca_var
print(fviz_mca_var(my_mca,color="blue",repel=True))
```



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.

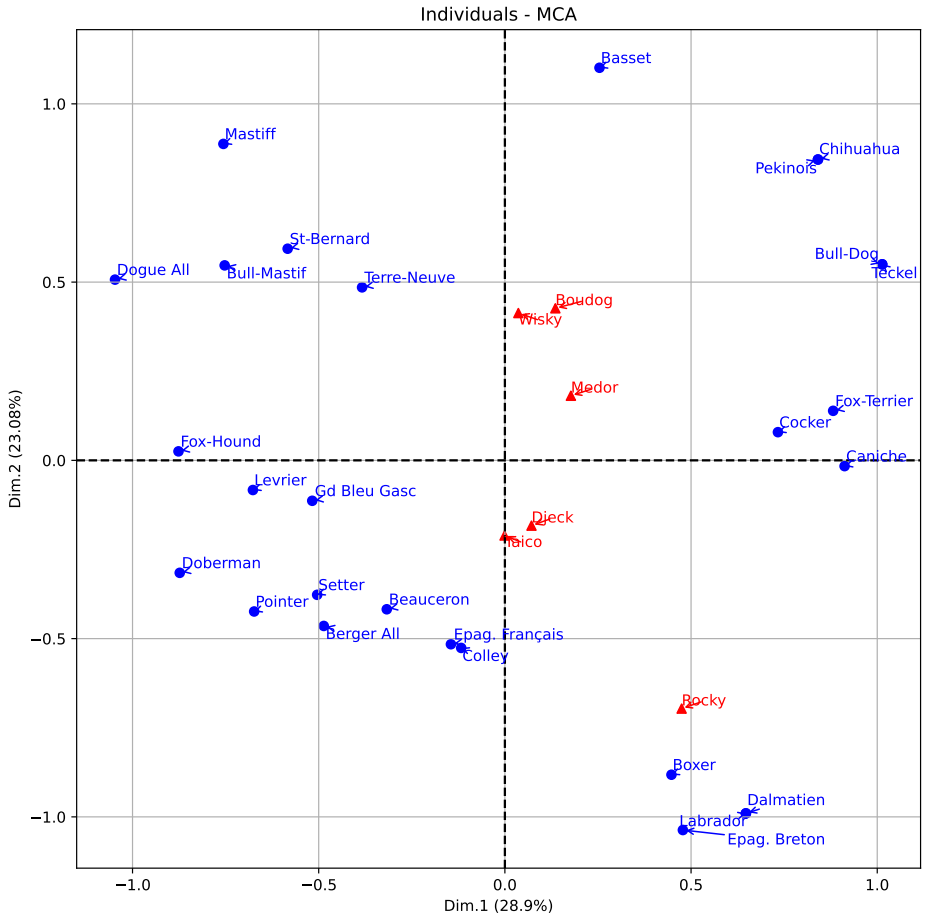
On ajoute la variable « Cote » comme variable continue illustrative quantitative et « Fonction » comme variable qualitative. Tapez la ligne de code suivante :

```
# ACM avec les éléments supplémentaires
my_mca2 = MCA(n_components=None,
              row_labels=A.index,
              var_labels=A.columns,
              mod_labels=None,
              matrix_type="completed",
              benzecri=True,
              greenacre=True,
              row_sup_labels=B.index,
              quali_sup_labels=["Fonction"],
              quanti_sup_labels=["Cote"],
              parallelize=False)
```

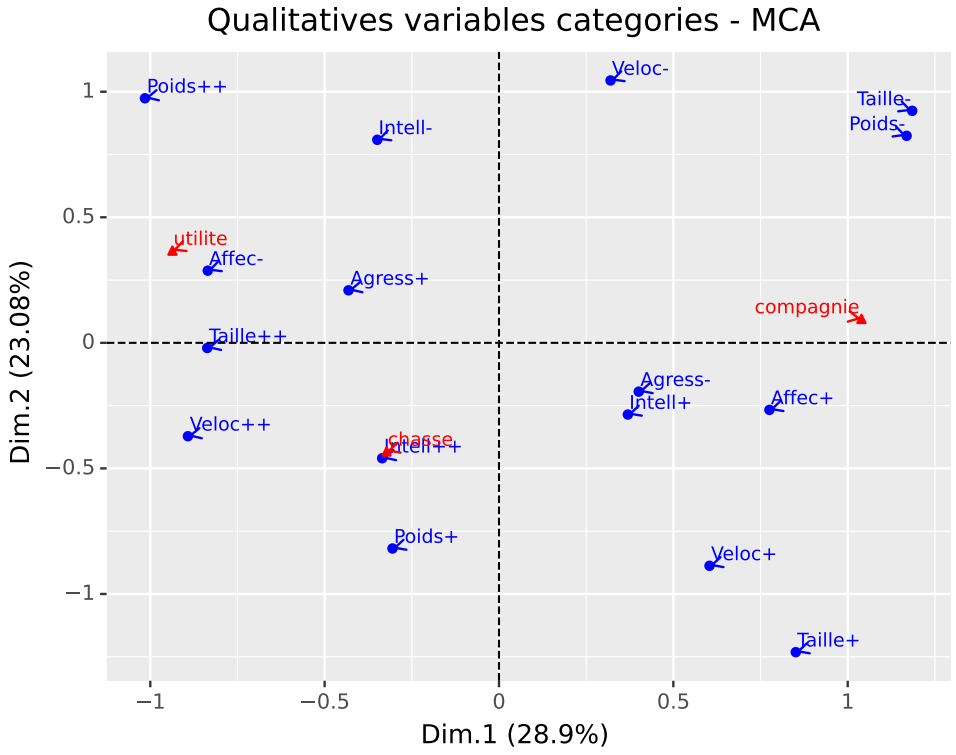
```
# Estimation
my_mca2.fit(Data)
```

```
## MCA(quali_sup_labels=['Fonction'], quanti_sup_labels=['Cote'],
##     row_labels=Index(['Beauceron', 'Basset', 'Berger All', 'Boxer', 'Bull-Dog', 'Bull-Mas
##        'Caniche', 'Chihuahua', 'Cocker', 'Colley', 'Dalmatien', 'Doberman',
##        'Dogue All', 'Epag. Breton', 'Epag. Français', 'Fox-Hound',
##        'Fox-Terrier', 'Gd Bleu Gasc', 'Labrador', 'Levrier', 'Mastiff',
##        'Pekinois', 'Pointer', 'St-Bernard', 'Setter', 'Teckel', 'Terre-Neuve'],
##       dtype='object', name='Chien'),
##     row_sup_labels=Index(['Medor', 'Djeck', 'Taico', 'Rocky', 'Boudog', 'Wisky'], dtype='
##     var_labels=Index(['Taille', 'Poids', 'Velocite', 'Intelligence', 'Affection',
##        'Agressivite'],
##       dtype='object'))
```

```
# Carte des individus
from scientisttools.pyplot import plotMCA
fig, axe = plt.subplots(figsize=(10,10))
plotMCA(my_mca2,choice="ind",ind_sup=True,repel=True,ax=axe)
plt.show()
```



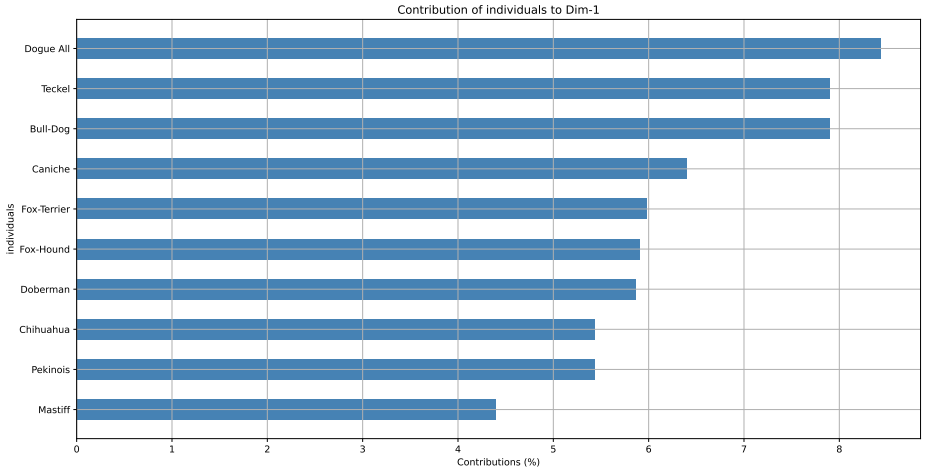
```
# Carte des modalités
print(fviz_mca_mod(my_mca2,color="blue",repel=True))
```



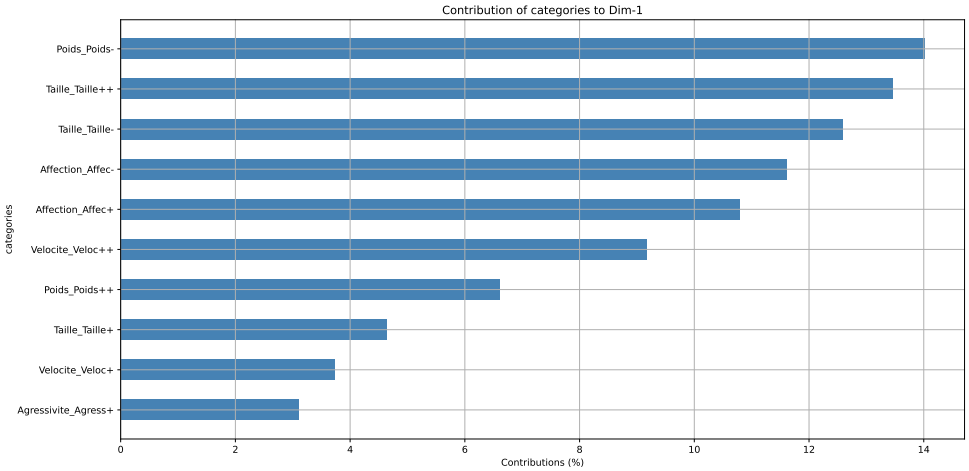
Interprétation des axes

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.

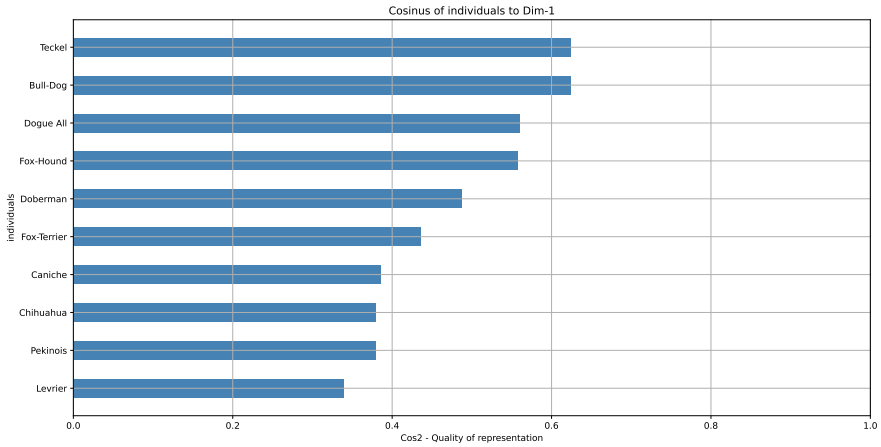
```
# Classement des points lignes en fonction de leur contribution au 1er axe
from scientisttools.pyplot import plot_contrib, plot_cosines
fig, axe = plt.subplots(figsize=(16,8))
plot_contrib(my_mca,choice="ind",axis=0,top_contrib=10,ax=axe)
plt.show()
```



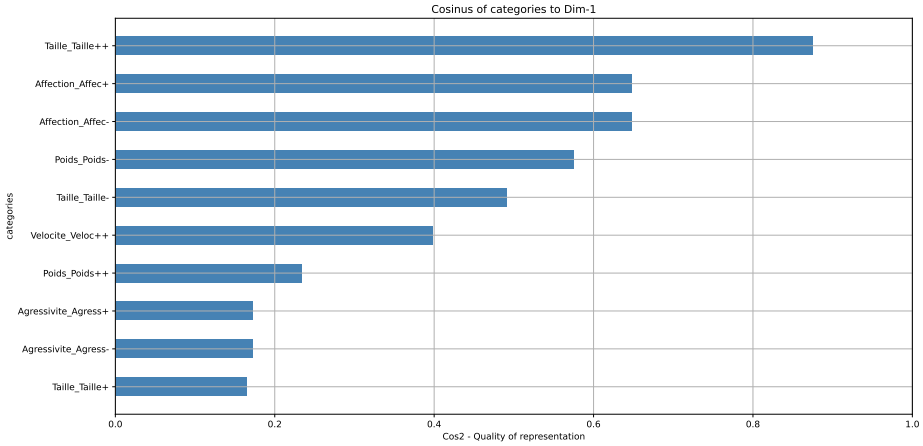

```
# Classement des modalités en fonction de leur contribution au 1er axe
fig, axe = plt.subplots(figsize=(16,8))
plot_contrib(my_mca,choice="mod",axis=0,ax=axe)
plt.show()
```




```
# Classement des individus en fonction de leur cos2 sur le 1er axe
fig, axe = plt.subplots(figsize=(16,8))
plot_cosines(my_mca,choice="ind",axis=0,top_cos2=10,ax=axe)
plt.show()
```




```
# Classement des modalités en fonction de leur cos2 sur le 1er axe
fig, axe = plt.subplots(figsize=(16,8))
plot_cosines(my_mca,choice="mod",axis=0,ax=axe)
plt.show()
```



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)[:5,:]
```

```
## array([[-0.31720005, -0.4177013 , -0.10146771, -0.21143628, -0.11850954,
##         -0.84491727, -0.08905015,  0.20198641, -0.16701884,  0.02280671],
##        [ 0.25410984,  1.10122699, -0.19070097,  0.29263727, -0.52400852,
##          0.03989468, -0.05283316, -0.44736292,  0.10073837,  0.1471022 ],
```

```
##        [-0.48639553, -0.46444958, -0.49813388,  0.57742525,  0.27590205,
##         -0.56776484,  0.12909684,  0.18733032, -0.234185  , -0.00891993],
##        [ 0.44736492, -0.88177794,  0.6920158 ,  0.26000184, -0.45558984,
##         -0.21374584,  0.00300768, -0.01981927, -0.00244629,  0.14090095],
##        [ 1.01335218,  0.54987949, -0.1634232 , -0.34991927,  0.33078648,
##         -0.20141418, -0.06354408, -0.07903568, -0.03560244,  0.06654325]])
```

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

```
## array([[-0.31720005, -0.4177013 , -0.10146771, -0.21143628, -0.11850954,
##         -0.84491727, -0.08905015,  0.20198641, -0.16701884,  0.02280671],
##        [ 0.25410984,  1.10122699, -0.19070097,  0.29263727, -0.52400852,
##          0.03989468, -0.05283316, -0.44736292,  0.10073837,  0.1471022 ],
##        [-0.48639553, -0.46444958, -0.49813388,  0.57742525,  0.27590205,
##         -0.56776484,  0.12909684,  0.18733032, -0.234185  , -0.00891993],
##        [ 0.44736492, -0.88177794,  0.6920158 ,  0.26000184, -0.45558984,
##         -0.21374584,  0.00300768, -0.01981927, -0.00244629,  0.14090095],
##        [ 1.01335218,  0.54987949, -0.1634232 , -0.34991927,  0.33078648,
##         -0.20141418, -0.06354408, -0.07903568, -0.03560244,  0.06654325]])
```

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 en Composantes Prnicipales (4 axes retenus)
# puis une régression logistique
pipe = Pipeline([("mca", MCA(n_components=4,var_labels=A.columns)),
                 ("logistic_regression",
                   LogisticRegression(multi_class="multinomial",
                                      solver="lbfgs",penalty=None))])
```

```
# Estimation du modèle
pipe.fit(X, y)
```

```
## Pipeline(steps=[('mca',
##                  MCA(n_components=4,
##                      var_labels=Index(['Taille', 'Poids', 'Velocite', 'Intelligence', 'Af
##        'Agressivite'],
##       dtype='object'))),
##                 ('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(var_labels=A.columns)),
                  ("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(var_labels=Index(['Taille', 'Poids', 'Velocit
##        'Agressivite'],
##       dtype='object'))),
##                                        ('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)
```

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