

Master :Mathématiques appliquées pour la science des données

Pratique d 'analyse des correspondances multiple avec python

Réalisée par:

- ☐ Touayba Baissa
- ☐ Salma El goude

Encadré par :

- ☐ Pr of.Aissam Hadri

PLAN

1

Analyse des Correspondances Multiples

2

Charge des données

3

Travailler sur profils lignes

4

Travailler sur profils colonnes

5

Pratique de ACM



Analyse des Correspondances Multiples

- ❖ L'**analyse des correspondances multiples (ACM)** est une méthode statistique multivariée qui permet d'explorer les relations entre des variables qualitatives nominales. Elle est particulièrement utile pour analyser des données catégorielles, c'est-à-dire des données qui se composent de catégories ou de classes distinctes.
- ❖ L'objectif principal de l'analyse des correspondances multiples (ACM) est de **faciliter la compréhension des relations entre les variables qualitatives..**



Domaines d'application

Analyse de l'opinion publique et de sondages:

L'ACM est couramment utilisée pour analyser les résultats des sondages d'opinion et des enquêtes. Elle permet d'identifier les segments de la population qui partagent des opinions ou des comportements similaires, et de comprendre les facteurs qui sous-tendent ces différences.

Exemple: Une enquête sur les habitudes de consommation peut révéler des segments de consommateurs distincts en fonction de leur âge, de leur revenu, de leur localisation et de leurs préférences en matière de produits.

Étude de marché et segmentation de la clientèle:

L'ACM est un outil précieux pour les entreprises qui cherchent à mieux comprendre leurs clients et à identifier des segments de marché potentiels. Elle permet d'analyser les données sur les préférences des consommateurs, les comportements d'achat et les attitudes vis-à-vis des marques et produits.

Exemple: Une entreprise de vêtements peut utiliser l'ACM pour analyser les données d'achat et les préférences de style de ses clients afin de développer des campagnes marketing ciblées et des produits plus adaptés à chaque segment de clientèle.

Analyse des réseaux sociaux et des communautés en ligne:

L'ACM est de plus en plus utilisée pour analyser les réseaux sociaux et les communautés en ligne. Elle permet d'identifier les groupes d'individus qui interagissent entre eux, de comprendre les relations entre ces groupes et de cartographier les flux d'informations au sein du réseau.

Exemple: Une plateforme de médias sociaux peut utiliser l'ACM pour analyser les interactions entre ses utilisateurs afin d'identifier les influenceurs clés, les communautés d'intérêt et les tendances émergentes.

Principe de l'Analyse des Correspondances Multiples (ACM)

Nous allons utiliser l'ACM pour analyser un tableau de données où les variables sont qualitatives qui nous permet de visualiser les similarités et différences entre les individus et de comprendre les relations entre les variables.. Les questions centrales sont :

- **Quels sont les chiens qui se ressemblent ? Qui sont dissemblables ? (proximité entre les individus)**
- **Sur quels caractères sont fondées ces ressemblances/dissimilarités ?**

Tableau des Données "Canidés"

Variables « actives » qualitatives
c.-à-d. sont utilisées pour la
construction des facteurs

$j : 1, \dots, p$

$i : 1, \dots, n$
Individus actifs

ID	Chien	Taille	Velocite	Affection
1	Beauceron	Taille++	Veloc++	Affec+
2	Basset	Taille-	Veloc-	Affec-
3	Berger All	Taille++	Veloc++	Affec+
4	Boxer	Taille+	Veloc+	Affec+
5	Bull-Dog	Taille-	Veloc-	Affec+
6	Bull-Mastif	Taille++	Veloc-	Affec-
7	Caniche	Taille-	Veloc+	Affec+
8	Labrador	Taille+	Veloc+	Affec+

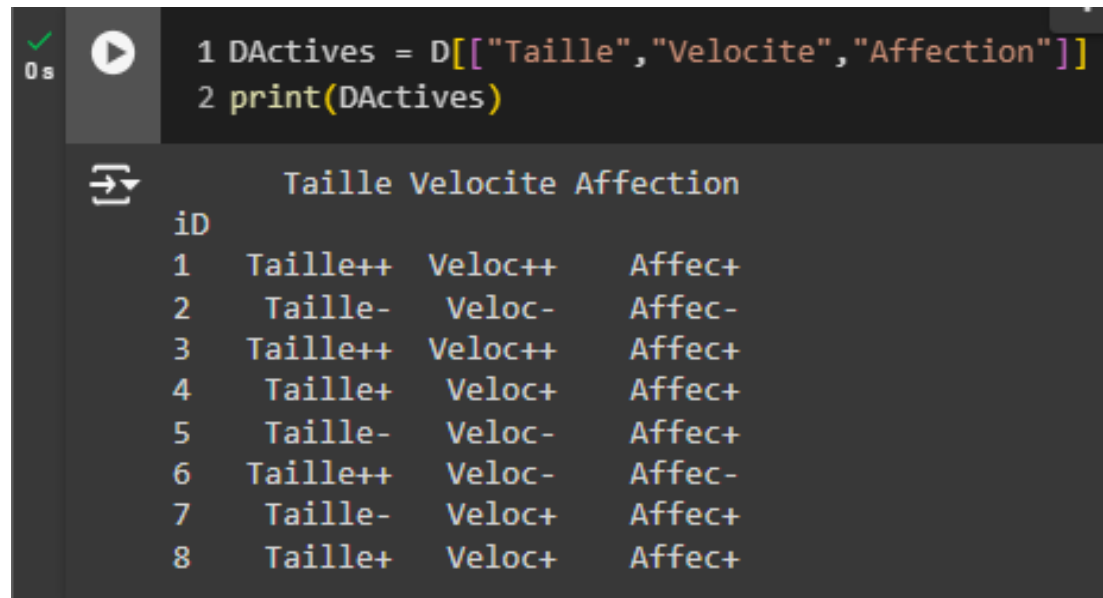
Charger des données



```
1 import pandas as pd
2 D=pd.read_excel("ACM.xlsx",index_col=0)
```

	Chien	Taille	Velocite	Affection
id				
1	Beauceron	Taille++	Veloc++	Affec+
2	Basset	Taille-	Veloc-	Affec-
3	Berger All	Taille++	Veloc++	Affec+
4	Boxer	Taille+	Veloc+	Affec+
5	Bull-Dog	Taille-	Veloc-	Affec+
6	Bull-Mastif	Taille++	Veloc-	Affec-
7	Caniche	Taille-	Veloc+	Affec+
8	Labrador	Taille+	Veloc+	Affec+

Récupération des variable actives



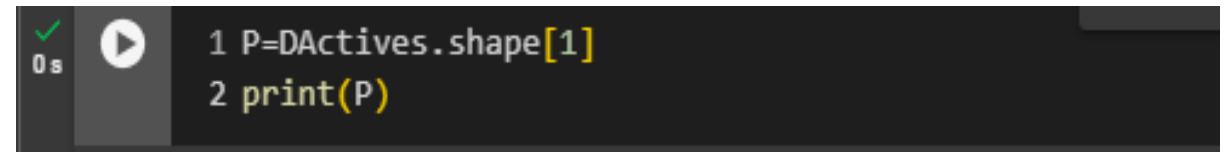
0 s

```
1 DActive = D[["Taille","Vitesse","Affection"]]  
2 print(DActive)
```

↕

	Taille	Vitesse	Affection
iD			
1	Taille++	Vitesse++	Affec+
2	Taille-	Vitesse-	Affec-
3	Taille++	Vitesse++	Affec+
4	Taille+	Vitesse+	Affec+
5	Taille-	Vitesse-	Affec+
6	Taille++	Vitesse-	Affec-
7	Taille-	Vitesse+	Affec+
8	Taille+	Vitesse+	Affec+

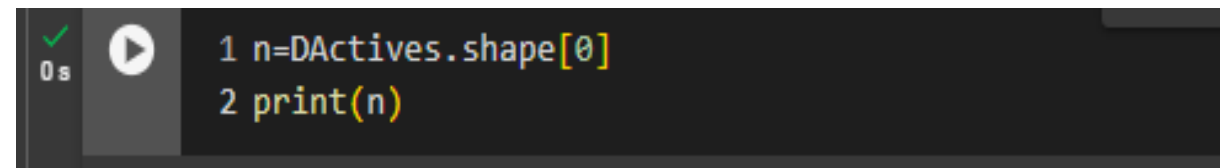
□ récupération des infos- nombre des variables



0 s

```
1 P=DActive.shape[1]  
2 print(P)
```

□ nombre d'observation



0 s

```
1 n=DActive.shape[0]  
2 print(n)
```

Codage disjonctif complet

Dans ce qui suit, nous travaillerons sur une version modifiée des données où les variables qualitatives seront transformées en indicatrices 0/1 à l'aide d'un codage disjonctif complet. Cela signifie que chaque catégorie de variable qualitative sera représentée par une colonne distincte contenant des 0 et des 1, indiquant l'absence ou la présence de cette catégorie pour chaque observation. Nous regroupons ces éléments dans le graphique suivant pour illustrer la notation (Figure 143).

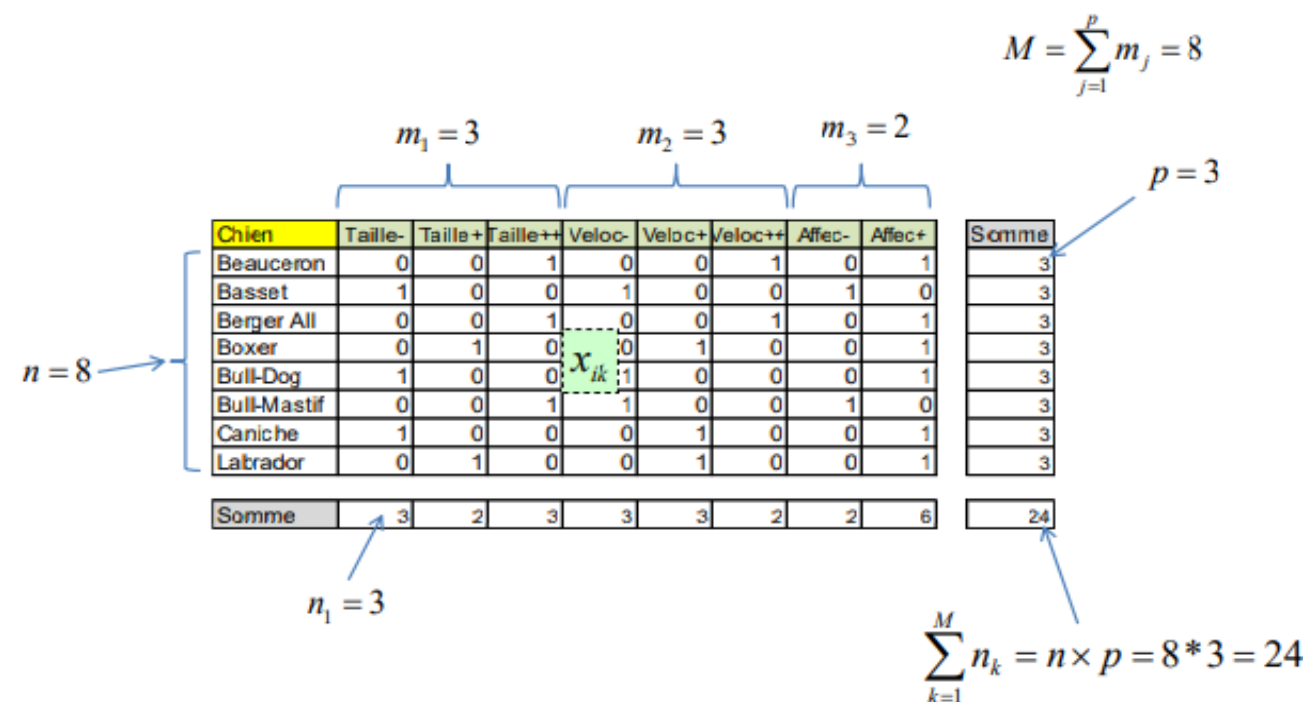


Figure 143 - Codage disjonctif complet - Notations

- Le tableau binaire comporte toujours $n=8$ lignes, mais avec $M=8$ colonnes. M représente la somme du nombre de modalités des variables. Les sommes des lignes et des colonnes sont importantes car elles vont définir les profils moyens. L'effectif total est modifié en raison de l'introduction de la redondance dans les données, et il est maintenant égal au produit $n \times p$

codage disjonctif complet sous python

```
[35] 1 X = pd.get_dummies(DActives,prefix='',prefix_sep='')
      2 X = X.astype(int)
      3 # Réorganisation des colonnes selon l'ordre souhaité
      4 desired_order = ['Taille-', 'Taille+', 'Taille++', 'Veloc-', 'Veloc+', 'Veloc++', 'Affec-', 'Affec+']
      5 X = X[desired_order]
      6 print(X)
```

	Taille-	Taille+	Taille++	Veloc-	Veloc+	Veloc++	Affec-	Affec+
id								
1	0	0	1	0	0	1	0	1
2	1	0	0	1	0	0	1	0
3	0	0	1	0	0	1	0	1
4	0	1	0	0	1	0	0	1
5	1	0	0	1	0	0	0	1
6	0	0	1	1	0	0	1	0
7	1	0	0	0	1	0	0	1
8	0	1	0	0	1	0	0	1

➤ La fonction '**pd.get_dummies**' transforme les variables catégorielles en variables indicatrices, créant ainsi une représentation numérique des catégories.

➤ **prefix** : Préfixe à ajouter aux noms de colonnes pour identifier la variable d'origine.

➤ **prefix_sep** : Séparateur à utiliser si un préfixe est spécifié.

➤ La méthode '**astype(int)**' convertit les valeurs de la colonne spécifiée en entiers.

Si les valeurs de la colonne sont de type float, bool, ou autre, elles seront converties en entiers en supprimant la partie décimale (en cas de float) ou en assignant **1** pour **True** et **0** pour **False** (en cas de bool).



■ Travailler sur les profils lignes

Analyse des proximités entre les individus

- Le premier prisme d'analyse consiste à examiner les proximités entre les individus à travers leurs profils lignes (Figure 144). L'individu "moyen" est défini par le profil moyen. Sa description est construite à partir des fréquences des catégories relatives à l'effectif total.

Barycentre (O)

$\frac{n_k}{n \times p}$

$\frac{x_{ik}}{p}$

Chien	Taille-	Taille+	Taille++	Veloc-	Veloc+	Veloc++	Affec-	Affec+
Beauceron	0.000	0.000	0.333	0.000	0.000	0.333	0.000	0.333
Basset	0.333	0.000	0.000	0.333	0.000	0.000	0.333	0.000
Berger All	0.000	0.000	0.333	0.000	0.000	0.333	0.000	0.333
Boxer	0.000	0.333	0.000	0.000	0.333	0.000	0.000	0.333
Bull-Dog	0.333	0.000	0.000	0.333	0.000	0.000	0.000	0.333
Bull-Mastif	0.000	0.000	0.333	0.333	0.000	0.000	0.333	0.000
Caniche	0.333	0.000	0.000	0.000	0.333	0.000	0.000	0.333
Labrador	0.000	0.333	0.000	0.000	0.333	0.000	0.000	0.333
Profil moyen	0.125	0.083	0.125	0.125	0.125	0.083	0.083	0.250

Figure 144 - Tableau des profils lignes et profil moyen

Analyse des proximités entre les individus sous python

```
✓ 0s ▶ 1 import numpy as np
      2 #profil individu moyen
      3 ind_moy = np.sum(X.values,axis=0)/(n*p)
      4 print(ind_moy)

[0.125      0.08333333 0.125      0.125      0.125      0.08333333
 0.08333333 0.25      ]
```

- ❖ **np.sum(X.values, axis=0)** : La fonction np.sum calcule la somme des valeurs le long de l'axe 0 (c'est-à-dire colonne par colonne). Cela donne un tableau où chaque élément est la somme des valeurs 0 et 1 pour cette colonne.
- ❖ **n** : Le nombre d'individus (lignes) dans le DataFrame.
- ❖ **p** : Le nombre de variables initiales avant le codage disjonctif complet.
- ❖ **(n * p)** : Ceci est le produit du nombre d'individus et du nombre de variables, représentant une normalisation basée sur l'effectif total.
- ❖ **Division par (n * p)** : Normalise la somme des indicatrices pour chaque modalité, résultant en une fréquence relative de chaque modalité dans l'ensemble des données.

Distance entre individus



Puisque nous traitons des variables catégorielles, nous utilisons la distance du χ^2 , qui a la particularité d'exacerber les différences entre les modalités rares.

$$d^2(i, i') = \sum_{k=1}^M \frac{1}{\frac{n_k}{n \times p}} \left(\frac{x_{ik}}{p} - \frac{x_{i'k}}{p} \right)^2$$

Distance entre individus sous python

```
[9] 1 #distance entre beauceron (n°0) et basset (n°1)  
2 print(np.sum(1/ind_moy*(X.values[0,:]/p-X.values[1,:]/p)**2))
```

```
⇒ 5.777777777777777
```

✓
0s

```
1 #distance entre basset(n°1) et caniche(n°6)  
2 print(np.sum(1/ind_moy*(X.values[1,:]/p-X.values[6,:]/p)**2))
```

```
⇒ 3.5555555555555554
```

Visiblement, le Basset a plus de caractères en commun avec le Caniche qu'avec le Beauceron.

Distance à l'origine



La distance à l'origine correspond à la distance des individus au profil moyen représenté par le vecteur (0.125, 0.083, ..., 0.250). .

Sous python

```
[12] 1 #distance à l'origine du basset (n°1)
      2 print(np.sum(1/ind_moy*(X.values[1,:]/p-ind_moy)**2))

2.1111111111111107

1 #distance à l'origine du caniche (n°6)
2 print(np.sum(1/ind_moy*(X.values[6,:]/p-ind_moy)**2))

1.2222222222222219
```

Nous concluons que le Caniche est plus proche du « chien moyen » que le Basset.

Inertie totale

L'inertie totale dans l'analyse des correspondances multiples (ACM) est une mesure qui exprime la quantité totale de variabilité ou d'information contenue dans les données. En ACM, elle peut être définie de deux manières : à travers les distances entre les paires d'individus :

$$I = \frac{1}{2n^2} \sum_{i=1}^n \sum_{i' \neq i} d^2(i, i')$$

ou via les distances à l'origine :

$$I = \frac{1}{n} \sum_{i=1}^n d^2(i)$$

$\frac{1}{n}$ Inertie d'un individu			
Chien	Poids	$d^2(O)$	Inertie
Beauceron	0.125	1.667	0.208
Basset	0.125	2.111	0.264
Berger All	0.125	1.667	0.208
Boxer	0.125	1.667	0.208
Bull-Dog	0.125	1.222	0.153
Bull-Mastif	0.125	2.111	0.264
Caniche	0.125	1.222	0.153
Labrador	0.125	1.667	0.208

Inertie totale	1.667
----------------	-------

Inertie total sous python

```
[14] 1 #distance à l'origine des individus (obs.)
      2 disto_ind = np.apply_along_axis(arr=X.values,axis=1,func1d=lambda x:np.sum(1
      3 /ind_moy*(x/p-ind_moy)**2))
      4 #poids des obs.
      5 poids_ind = np.ones(X.shape[0])/n
      6 #inertie
      7 inertie_ind = poids_ind*disto_ind
      8 print(inertie_ind)

[0.20833333 0.26388889 0.20833333 0.20833333 0.15277778 0.26388889
 0.15277778 0.20833333]

1 #inertie totale
2 inertie_tot_ind = np.sum(inertie_ind)
3 print(inertie_tot_ind)

1.6666666666666665
```

- ❖ **np.apply_along_axis**: Cette fonction applique une fonction à 1D (unidimensionnelle) le long de l'une des axes d'un tableau 2D (bidimensionnel).
- ❖ **arr=X.values**: Les valeurs du DataFrame X sont converties en un tableau NumPy.
- ❖ **axis=1**: L'opération est appliquée le long des lignes (chaque ligne représente un individu).
- ❖ **func1d=lambda x: np.sum(1 / ind_moy * (x / p - ind_moy)**2)**: La fonction à appliquer. C'est une fonction lambda (fonction anonyme) qui calcule la distance chi-deux (χ^2) entre un individu et le profil moyen.



■ Travailler sur les profils colonnes

Distance entre les modalités

La distance entre modalités est définie par la distance du χ^2 toujours :

$$d^2(k, k') = \sum_{i=1}^n \frac{1}{n} \left(\frac{x_{ik}}{n_k} - \frac{x_{ik'}}{n_{k'}} \right)^2$$

Barycentre : $\frac{p}{n \times p} = \frac{1}{n}$

Chien	Taille-	Taille+	Taille++	Veloc-	Veloc+	Veloc++	Affec-	Affec+	Profil moyen
Beauceron	0.000	0.000	0.333	0.000	0.000	0.500	0.000	0.167	0.125
Basset	0.333	0.000	0.000	0.333	0.000	0.000	0.500	0.000	0.125
Berger All	0.000	0.000	0.333	0.000	0.000	0.500	0.000	0.167	0.125
Boxer	0.000	0.500	0.000	0.000	0.333	0.000	0.000	0.167	0.125
Bull-Dog	0.333	0.000	0.000	0.333	0.000	0.000	0.000	0.167	0.125
Bull-Mastif	0.000	0.000	0.333	0.333	0.000	0.000	0.500	0.000	0.125
Caniche	0.333	0.000	0.000	0.000	0.333	0.000	0.000	0.167	0.125
Labrador	0.000	0.500	0.000	0.000	0.333	0.000	0.000	0.167	0.125

Distance entre les modalités sous python

```
✓ 0s ▶ 1 #somme en colonne
2 somme_col = np.sum(X.values,axis=0)
3 print(somme_col)
4 #distance entre taille- (2) et velocite- (5)
5 print(np.sum(n*((X.values[:,2]/somme_col[2]-X.values[:,5]/somme_col[5])**2)))
6 #distance entre taille- (2) et velocite+ (3)
7 print(np.sum(n*((X.values[:,2]/somme_col[2]-X.values[:,3]/somme_col[3])**2)))
```

```
→ [3 2 3 3 3 2 2 6]
1.3333333333333335
3.5555555555555554
```

les individus qui partagent les caractéristiques (taille-, vitesse-) sont plus nombreux que (taille-, vitesse+).

Distance à l'origine

La distance à l'origine est définie par la distance du χ^2 au profil moyen :

$$d^2(k) = \sum_{i=1}^n \frac{1}{\frac{1}{n}} \left(\frac{x_{ik}}{n_k} - \frac{1}{n} \right)^2$$

```
[18] 1 #profil moyen des variables-modalités
      2 moda_moy = np.ones(X.shape[0])/n
      3 print(moda_moy)

[0.125 0.125 0.125 0.125 0.125 0.125 0.125 0.125]

[19] 1 #distance à l'origine taille-
      2 print(np.sum(n*((X.values[:,0]/somme_col[0]-moda_moy)**2)))
      3
      4 #distance à l'origine taille+
      5 print(np.sum(n*((X.values[:,1]/somme_col[1]-moda_moy)**2)))
      6

1.6666666666666665
3.0
```

« Taille+ » est une caractéristique que l'on retrouve plus rarement que « Taille- » pour les animaux de notre base de données.

Inertie totale

L'inertie d'une modalité est exprimée par le produit entre son poids relatif ($\omega_k = n_k / n \times p$) et sa distance à l'origine :

$$I(k) = \omega_k \times d^2(k)$$

$$I(k) = \frac{n_k}{n \times p} \times d^2(k)$$

Et l'inertie totale, l'information portée par les données via le prisme des modalités, correspond à la somme de leur inertie :

$$I = \sum_{k=1}^M I(k)$$

$$\omega_k = \frac{n_k}{n \times p}$$

	Taille-	Taille+	Taille++	Veloc-	Veloc+	Veloc++	Affec-	Affec+
Poids	0.125	0.083	0.125	0.125	0.125	0.083	0.083	0.250
d ² (Moda)	1.667	3.000	1.667	1.667	1.667	3.000	3.000	0.333

Inertie d'une modalité →

Inertie	0.208	0.250	0.208	0.208	0.208	0.250	0.250	0.083
---------	-------	-------	-------	-------	-------	-------	-------	-------

Inertie totale	1.667
----------------	-------

Inertie totale sous python

```
[22] 1 #poids des variables_modalités (points modalités)
      2 poids_moda = somme_col/(n*p)
      3 #distance à l'origine des points modalités
      4 disto_moda = np.apply_along_axis(arr=X.values/somme_col,axis=0,func1d=lambda x:
      5 np.sum(n*(x-moda_moy)**2))
      6 #inertie
      7 inertie_moda = poids_moda * disto_moda
      8 #affichage
      9 print(pd.DataFrame(np.transpose([poids_moda,disto_moda,inertie_moda]),index
     10 =X.columns,columns=['Poids','Disto','Inertie']))
```

	Poids	Disto	Inertie
Taille-	0.125000	1.666667	0.208333
Taille+	0.083333	3.000000	0.250000
Taille++	0.125000	1.666667	0.208333
Veloc-	0.125000	1.666667	0.208333
Veloc+	0.125000	1.666667	0.208333
Veloc++	0.083333	3.000000	0.250000
Affec-	0.083333	3.000000	0.250000
Affec+	0.250000	0.333333	0.083333

```
1 #inertie totale des points-modalités
2 inertie_tot_moda = np.sum(inertie_moda)
3 print(inertie_tot_moda)
```

```
1.6666666666666665
```



Principe de l'analyse des correspondances multiple

Toujours dans l'optique de produire un système de représentation qui préserve au mieux les distances entre les modalités, l'ACM produit une succession de facteurs qui permettent de les discerner au mieux c.-à-d. qui maximisent la variance de leurs coordonnées factorielles. Pour le 1er facteur :

$$\lambda_1 = \sum_{k=1}^M \omega_k \times G_{k1}^2$$

Où :

- ω_k est le poids relatif de la modalité.
 - G_{kh} la coordonnée de la modalité « k » sur le facteur « h ».
 - λ_h est la variance associée au facteur « h », avec exactement la même valeur que son homologue obtenue via l'analyse des proximités entre les individus. Le 2nd facteur cherche à « modéliser » l'information non captée sur le 1er facteur ($I - \lambda_1$), etc. Le principe de la décomposition orthogonale reste valable. Les valeurs propres s'additionnent.
- 5.1.3.5 Quelques résultats remarquables A ce stade de notre présentation, considérons quelques résultats important



Quelques résultats remarquable

A ce stade de notre présentation, considérons quelques résultats importants.

- Le poids d'une modalité dépend de sa fréquence.

$$\omega_k = \frac{n_k}{n \times p}$$

- La distance à l'origine peut s'écrire plus simplement.
On se rend compte alors qu'une modalité est d'autant plus distante du profil moyen qu'elle est rare

$$d^2(k) = \frac{n}{n_k} - 1$$

- De fait, l'écriture de l'inertie d'une modalité est également simplifiée. Nous constatons qu'une modalité contribue d'autant plus à l'inertie quand elle est rare.

$$I(k) = \omega_k \times d^2(k) = \frac{1}{p} \left(1 - \frac{n_k}{n}\right)$$



Quelques résultats remarquable

- Et la contribution d'une variable à l'inertie globale est fonction du nombre de ses modalités

$$I(j) = \sum_{k=1}^{m_j} d^2(k) = \frac{1}{p}(m_j - 1)$$

- L'inertie totale ne dépend que des caractéristiques des variables : « p » leur nombre ; « M » le nombre total des modalités ; le nombre moyen de modalités par variable (M/p)

$$I = \sum_{j=1}^p I(j) = \frac{M}{p} - 1$$



*Merci
Pour votre
Attention*

