Analyse des correspondances avec python

1. Mise en œuvre pratique du programme (exemple « Mots_Educ »)

2. Contenu du fichier source téléchargeable : « ac_dtmF.py »

Le tableau que l'on va s'efforcer de décrire avec l'aide de l'analyse des correspondances est un tableau de fréquences croisant, en ligne, 14 mots utilisés dans les réponses à une question ouverte, et en colonne, 5 niveaux de diplômes[1] déclarés par chacune des personnes ayant répondu.

La question ouverte a pour libellé : *Quelles sont les raisons qui, selon vous, peuvent faire hésiter une femme ou un couple à avoir un enfant ?* (L'enquête était partiellement financée par la Caisse Nationale d'allocations familiales). Elle a été posée en 1981 à 2000 personnes représentant la population des résidents métropolitains de 18 ans et plus, dans l'enquête du CREDOC sur les Conditions de Vie et Aspirations des Français. Quatre mots et trois catégories supplémentaires (ou illustratifs, illustratives) interviennent *a posteriori* pour enrichir les interprétations.

Ce petit exemple introductif est commenté de façon détaillée dans le chapitre 3 de l'ouvrage « Statistique Textuelle » (Lebart et Salem, Dunod, 1994) librement téléchargeable sur ce site :
http://www.dtmvic.com/ST.html.

Tableau 1
Croisement (mots-Niveau de Diplôme). Effectifs bruts (fichier `Mots_Educ_act.txt`)

	Sans Dipl.	CEP	BEPC	Bacc	Univ	Total
Argent	51	64	32	29	17	193
Avenir	53	90	78	75	22	318
Chômage	71	111	50	40	11	283
Conjoncture	1	7	5	5	4	22
Difficile	7	11	4	3	2	27
Economique	7	13	12	11	11	54
Egoisme	21	37	14	26	9	107
Emploi	12	35	19	6	7	79
Finances	10	7	7	3	1	28
Guerre	4	7	7	6	2	26
Logement	8	22	7	10	5	52
Peur	25	45	38	38	13	159
Santé	18	27	20	19	9	93
Travail	35	61	29	14	12	151
Total	323	537	322	285	125	1592

[1] Sans diplôme, CEP (Certificat d'études primaires), BEPC (Brevet d'études du premier cycle ou équivalent), Bacc (Baccalauréat ou équivalent), Univ. (Université, grandes écoles ou équivalent).

Notons que l'unité statistique retenue ici n'est pas l'individu, mais l'occurrence d'un mot. Les colonnes constituent bien une partition de l'ensemble des personnes interrogées, mais pas les lignes : un répondant peut utiliser dans ses réponses plusieurs formes de la liste retenue ici. Les lignes constituent cependant une partition de l'ensemble des occurrences des mots.
On lit ce tableau de la manière suivante : le mot *Argent*, par exemple, a été utilisé 51 fois dans leurs réponses par les personnes appartenant à la catégorie "sans diplôme".
Les totaux de chaque ligne représentent les nombres d'occurrences de chaque mot alors que les totaux de chaque colonne représentent les nombres totaux de mots (de la liste) utilisées par les diverses catégories d'enquêtés.

Tableau 2

Quatre lignes supplémentaires (ou illustratives) (fichier `Mots_Educ_isup.txt`)

	Sans_Dipl	CEP	BEPC	Bac	Univ	TOTAL
Confort	2	4	3	1	4	14
Mésentente	2	8	2	5	2	19
Monde	1	5	4	6	3	19
Vivre	3	3	1	3	4	14

Tableau 3

Trois colonnes supplémentaires (ou illustratives) (fichier `Mots_Educ_vsup.txt`)

forme	Age-30	Age-50	Age+50
Argent	59	66	70
Avenir	115	117	86
Chômage	79	88	177
Conjoncture	9	8	5
Difficile	2	17	18
Economique	18	19	17
Egoisme	14	34	61
Emploi	21	30	28
Finances	8	12	8
Guerre	7	6	13
Logement	10	27	17
Peur	48	59	52
Santé	13	29	53
Travail	30	63	58
Total	433	575	663

C'est avec cet exemple « Modèle réduit » qu'il convient d'aborder le programme python (dont le listage complet figure aussi ci-dessous en section 2). Le programme (code python) figure dans le fichier : **`acomp_dtmF.py`**.

1. Mise en œuvre pratique du programme (exemple « Mots_Educ »).

Il convient de télécharger les trois fichiers de données précédents (**Mots_Educ_act.txt, Mots_Educ_isup.txt, Mots_Educ_vsup.txt**) ainsi que fichier code : **ac_dtmF.py** dans un dossier de travail. Le contenu du fichier de code python **ac_dtmF.py** figure en section 2 de cette note, mais constitue aussi un fichier séparé téléchargeable comme les données.

1. 1. Instructions préliminaires

Une fois le fichier code ac_dtmF.py copié dans votre dossier de travail (appelé ici « mydirectory ») avec les trois fichiers de données précédents, il suffit de taper une à une dans l'interface python (comme IDLE par exemple) les 4 instructions suivantes pour donner accès au programme et aux données :

Toutes les instructions qui suivent figurent en commentaires dans le fichier ac_dtmF.py, et peuvent donc être simplement copiées à partir de ce fichier.

```
import os
os.chdir("c:/mydirectory")
import ac_dtmF
from ac_dtmF import *
```

Il suffit, rappelons-le, de copier ces lignes directement à partir du fichier `ac_dtmF.py` que l'on aura ouvert dans un éditeur de texte libre (tel que Notepad ++, qui permet des éditions claires en python).

Ici, le dossier "mydirectory" est directement dans la racine "c:/". (A adapter pour chaque utilisateur, qui remplacera « mydirectory » par le chemin menant à son dossier de travail).

L'importation ci-dessus de `ac_dtmF.py` entraîne automatiquement le chargement des bibliothèques pandas, numpy et matplotlib.

1. 2. Calculs de base de l'AC : fonction AC_base( )

Pour l'exemple " Mini_Sem", on écrira dans l'interface python les 3 noms d'accès *sur une seule ligne* :

```
lane, lane_sup, lane_var_sup = "Mots_Educ_act.txt",
"Mots_Educ_isup.txt", "Mots_Educ_vsup.txt"
```

Pour la lecture des données et les calculs de base de l'ACP, on écrira :

```
X, psi, psi_u, phi, phi_u = AC_base(lane)
```

On obtient les impressions et graphiques suivants (figure 1) sur l'interface.

```
The coordinates are in the created file; AC_file_Mots_Educ_act.txt

 Eigenvalues

[3.54015389e-02 1.31147352e-02 7.30111400e-03 6.24391345e-03
 2.04504338e-33]

Coordinates of variables
     ident    c_var_1   c_var_2   c_var_3
0  No_Dipl -0.209318  0.080727 -0.072630
1      CEP -0.138577 -0.056047  0.018139
2     BEPC  0.108758  0.028483  0.147013
3     Bacc  0.274039  0.121344 -0.076653
4    Univ   0.231233 -0.317858 -0.094188

 complete coord. and coord. of indiv. are in the created file:
AC_file_Mots_Educ_act.txt
```

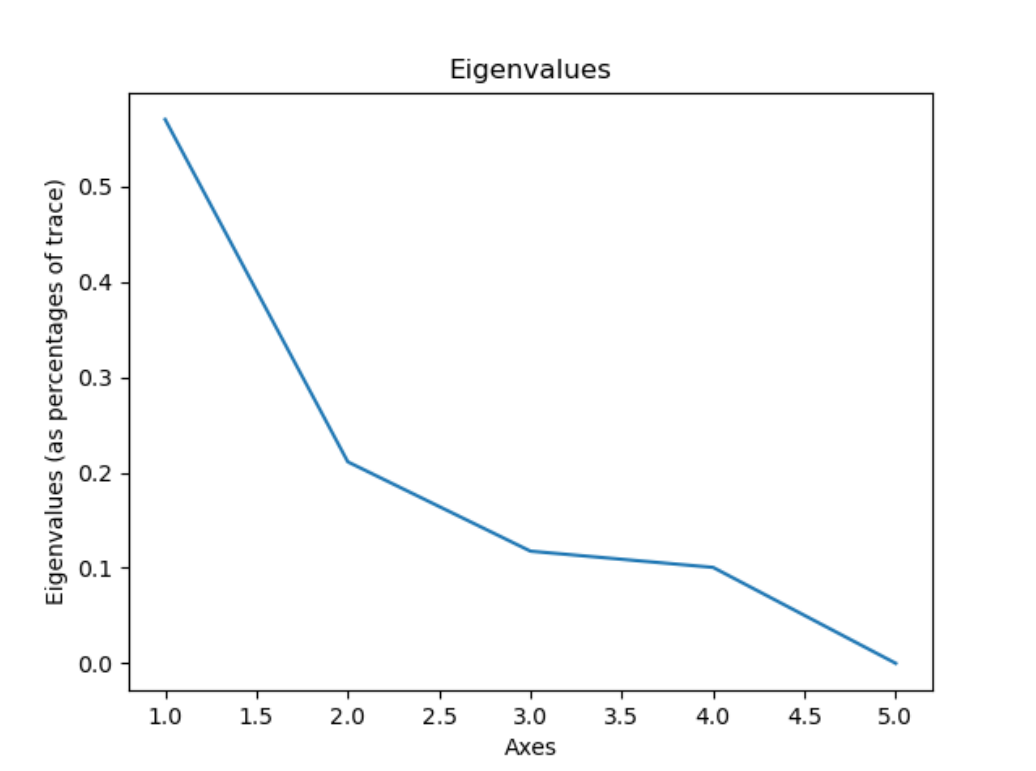

Figure 1. Série des valeurs propres

1. 3 Premières visualisations : éléments actifs

Selection d'une paire d'axes pour les visualisations.
On retiendra ici les visualisations sur les axes 1 et 2.

```
ax_h, ax_v = 1, 2
```

L'appel de la fonction `graf_act( )` produit les plans factoriels des éléments actifs (indiv. + variables)

```
graf_act (X, psi, phi, ax_h, ax_v )
```

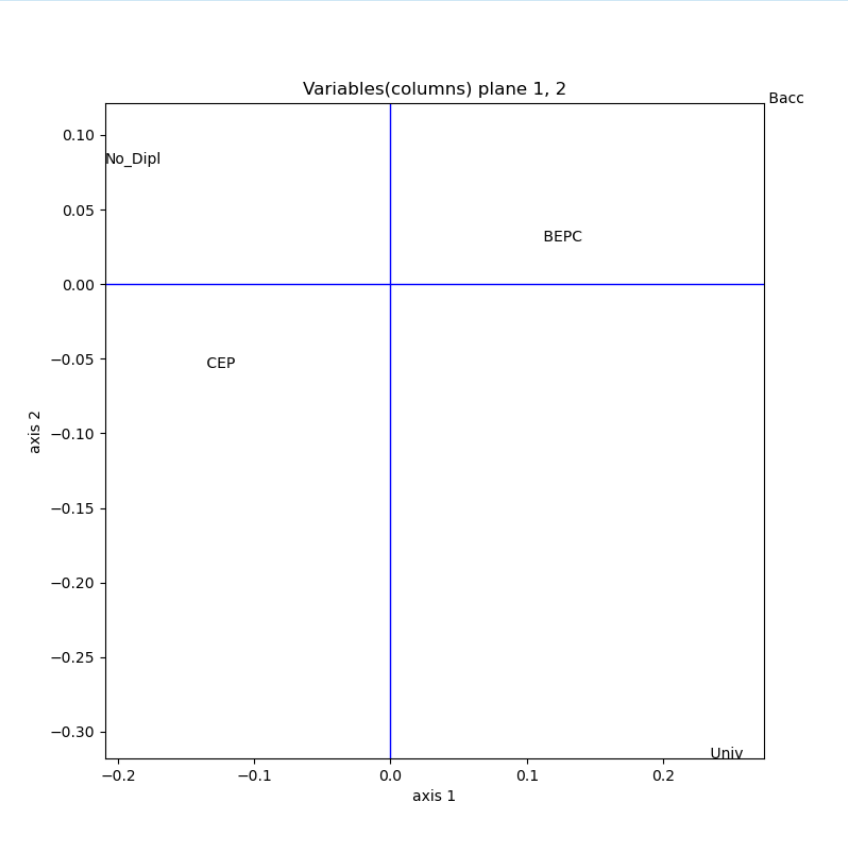

Figure 2. Position des colonnes (variables) dans le plan 1, 2

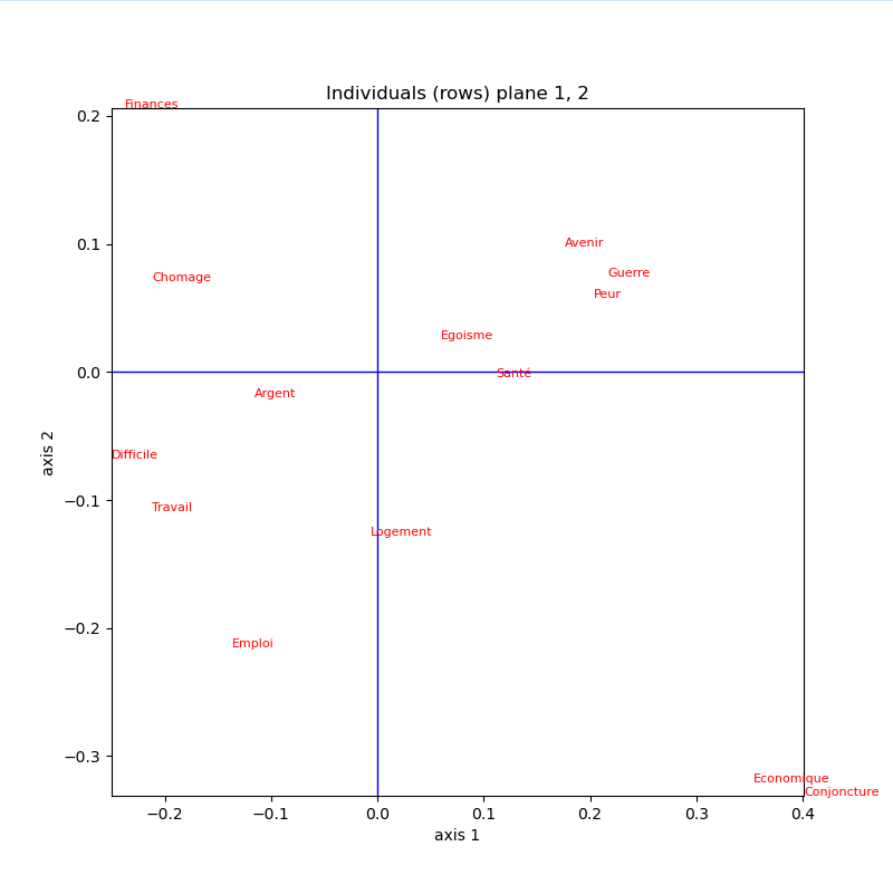

Figure 3. Position des lignes (individus) dans le plan 1, 2

1. 4. Représentation simultanée des lignes et colonnes actives

En raison des possibilités de représentation simultanée des lignes et des colonnes en AC, et en tenant compte des règles de lecture spécifiques, on peut désirer fusionner les figures 2 et 3 , avec la fonction :

```
graf_simult (X, psi, phi, ax_h, ax_v)
```

Rappelons que l'on ne peut pas interpréter la distance entre un point-ligne et un point-colonne. On peut en revanche interpréter la position d'un point-ligne par rapport à tous les points-colonnes, et la position d'un point-colonne par rapport à tous les points-lignes.

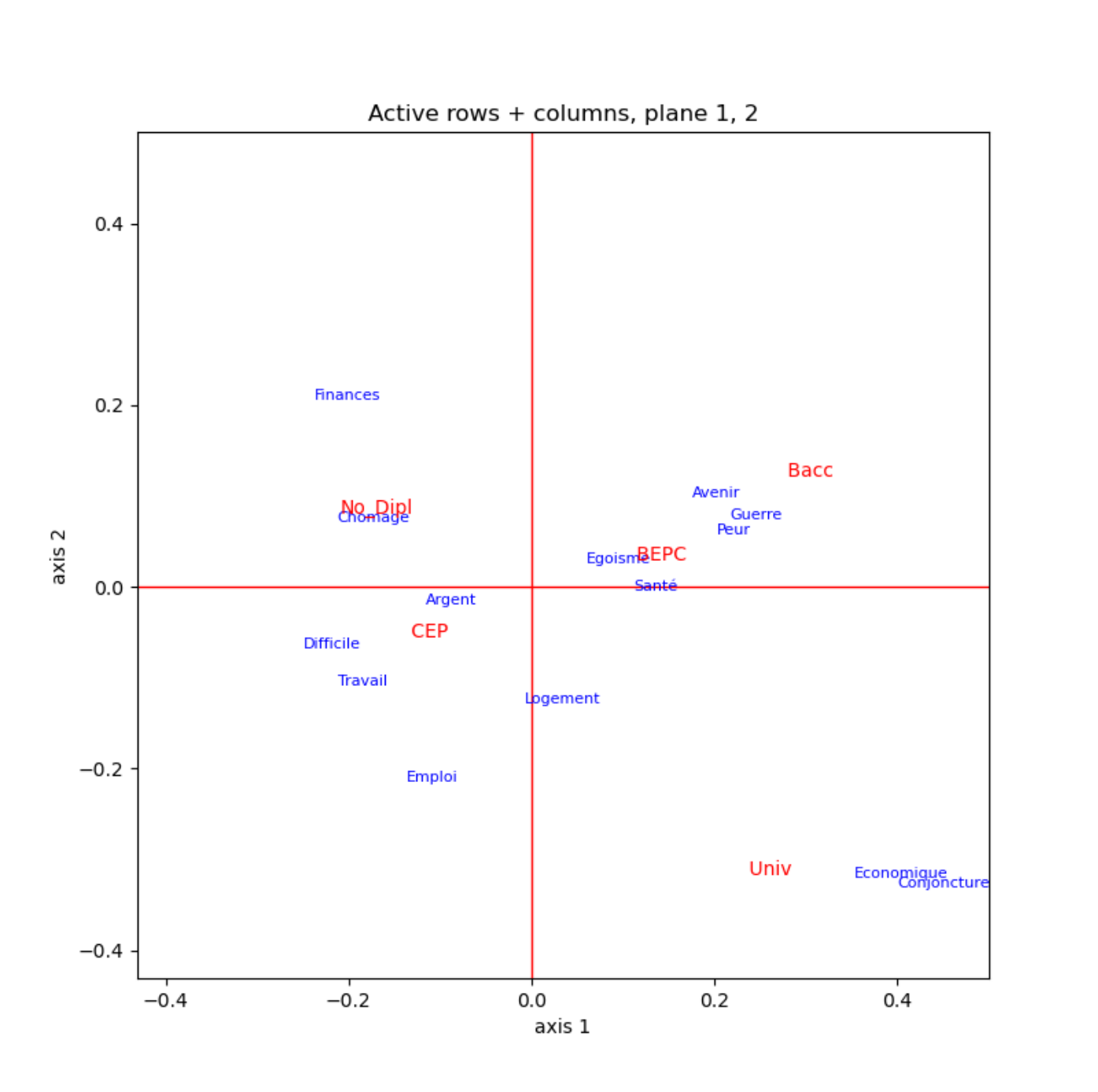

Figure 4. Représentation simultanée des lignes (mots) et des colonnes (niveau de diplôme) dans le plan (1, 2)

1. 5. Lignes (individus) supplémentaires

Tous les individus ou observations (lignes du tableau **`Mots_Educ_isup.txt`**) sont représentés par l'appel de la fonction **`AC_isup ( ).`**

```
AC_isup( X, psi,  phi, phi_u, lane_sup, ax_h, ax_v)
```

Ils sont repérables sur le plan (1, 2) de la figure 5 par une police légèrement plus grande et une couleur verte.

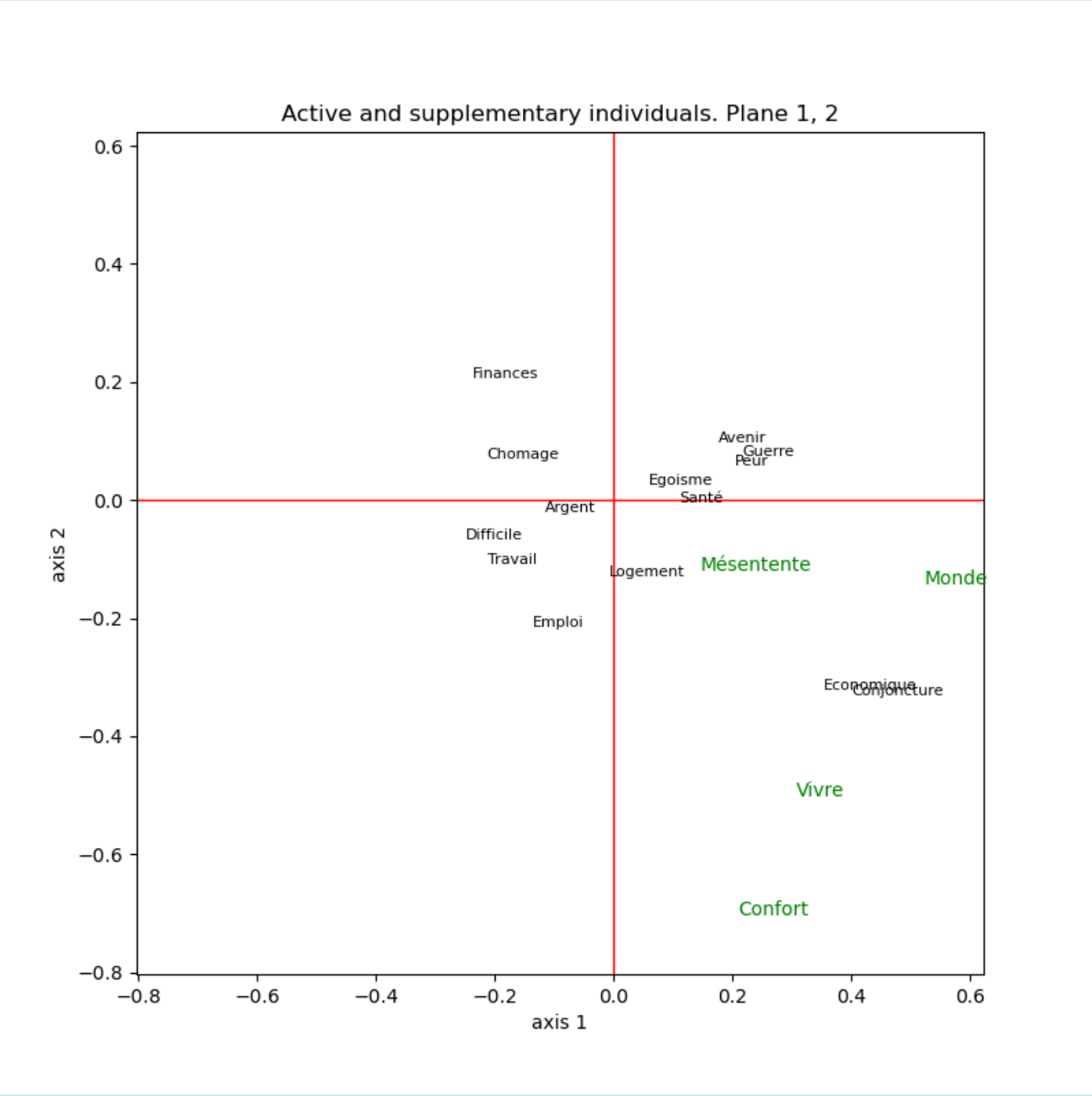

Figure 5. Représentation des 4 lignes supplémentaires dans le plan (1, 2)

1. 6. Variables (colonnes) supplémentaires

Les colonnes (variables) supplémentaires sont prises en compte par la fonction AC_vsup( ).

```
AC_vsup(X, psi, psi_u, phi, lane_var_sup, ax_h, ax_v)
```

On obtient une impression (sommaire) des coordonnées des colonnes supplémentaires sur les axes.

```
[[ 0.10541339  0.05969594  0.10322613  0.06977996]
 [-0.01706444 -0.04907657  0.01568923 -0.01306117]
 [-0.1770681   0.04813788 -0.10077299 -0.08517528]]
```

La figure 5 représente les positions des colonnes supplémentaires.

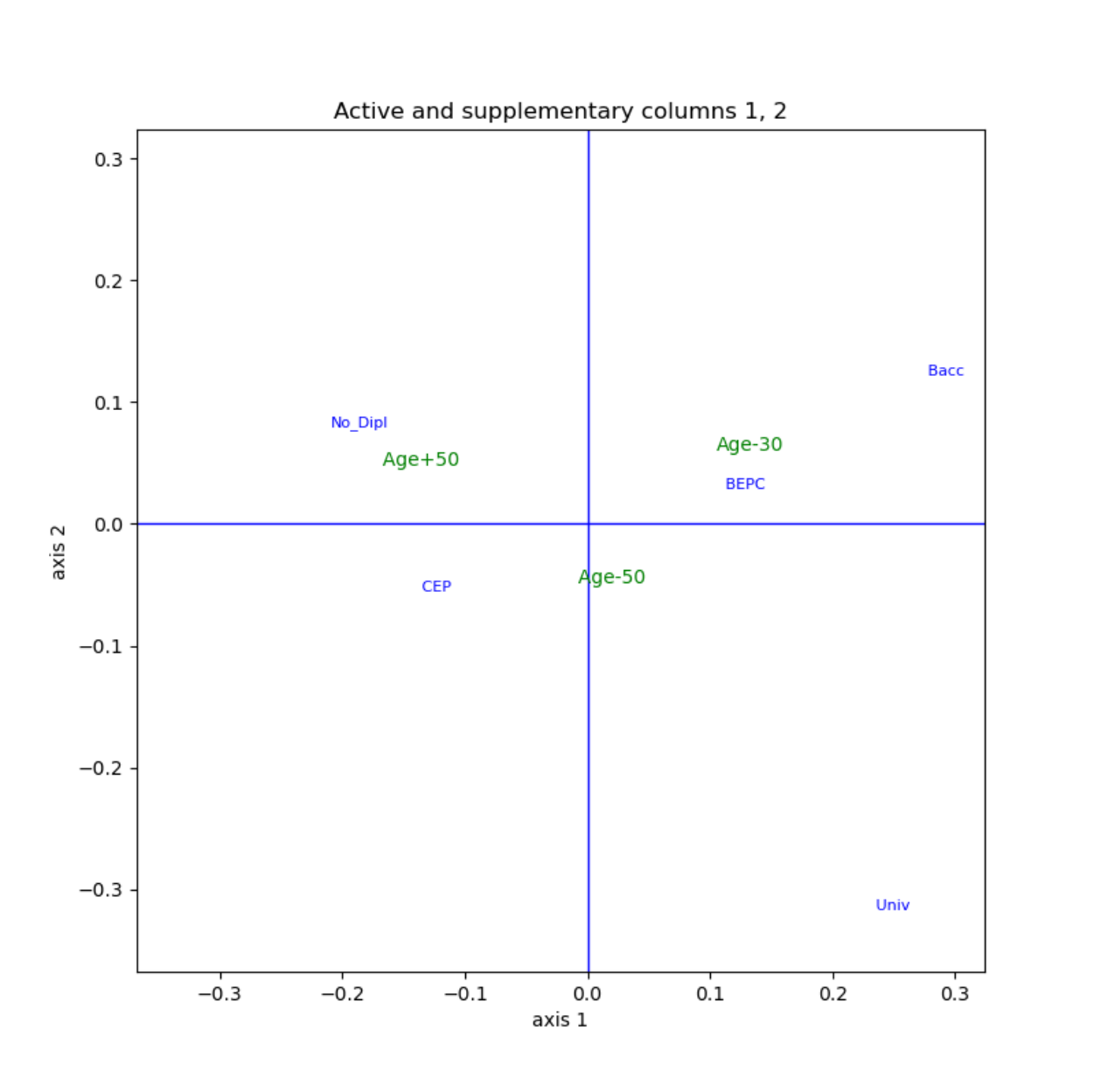

Figure 5. Position des colonnes supplémentaires (`Age-30, Age-50, Age+50`) parmi les colonnes actives dans le plan 1, 2

Remarque :

Ces 5 appels de fonctions auraient pu être remplacés par l'unique appel de la fonction synthétique AC ( ) figurant à la fin du code :

```
AC (lane, lane_sup, lane_var_sup, ax_h, ax_v)
```

Il faut, bien sûr, définir auparavant les cinq paramètres qui figurent dans la parenthèse ci-dessus.

2. Contenu du fichier source téléchargeable : `ac_dtmF.py`

Ce fichier contient aussi sous forme de commentaires le mode d'emploi du programme (instructions à saisir dans l'interface).

```
#**************** ac_dtmF.py ******************
#*** ici le fichier source "ac_dtmF.py" et les données
#*** sont dans un dossier "mydirectory" directement dans
#***la racine "c:/". A adapter pour chaque utilisateur qui
#*** remplacera « mydirectory »
#*** par le chemin menant à son dossier de travail.
#***-----------------------------------------------------
#*** Copier une à une les 4 lignes suivantes (sans les "# ")
#*** directement dans l'interface (IDLE…).
#***-----------------------------------------------------
# import os
# os.chdir("c:/mydirectory")
# import ac_dtmF
# from ac_dtmF import *
#***-----------------------------------------------------
#*** fonction d'importation: numpy, pandas, matplotlib
#*** Exécution automatique des 3 lignes ci-dessous au chargement du fichier
acomp_dtmF.py
#*** Inutile de les écrire dans l'interface
#
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#-----------------------------------------------------
#*** Copier successivement les instructions qui suivent le symbole « # ».
#*** Les commentaires (#***  ) n'ont évidemment pas besoin d'être copiés.
#***-----------------------------------------------------------
#*** SELECTION DES CHEMINS (adresses des fichiers)
#*** Exemple "Mots_Educ" : 3 chemins pour l'exemple Mots_Educ
# lane, lane_sup, lane_var_sup = "Mots_Educ_act.txt", "Mots_Educ_isup.txt",
"Mots_Educ_vsup.txt"
#***-----------------------------------------------------------
```

```
#*** SELECTION DES PARAMETRES
#*** Selection d'une paire d'axes pour les visualisations
# ax_h, ax_v = 1, 2
#***-------------------------------------------------------------
#*** EXECUTION de l'AC ****
#***-------------------------------------------------------------
#*** 1) AC_base() : lecture données et calculs de base
# X, psi, psi_u, phi, phi_u = AC_base(lane)
#
#*** 2) graf_act() : plan factoriels éléments actifs (indiv. + variables)
# graf_act (X, psi, phi, ax_h, ax_v )
#
#***  3) Repréentation simultanée des lignes et colonnes
# graf_simult (X, psi, phi, ax_h, ax_v)
#
#*** 4) AC_isup() Individus supplémentaires
# AC_isup( X, psi,  phi, phi_u, lane_sup, ax_h, ax_v)
#
#*** 5) AC_vsup() Variables numériques supplémentaires (de 1 -> vsup_lim)
# AC_vsup(X, psi, psi_u, phi, lane_var_sup, ax_h, ax_v)
#
#***-----------------------------------------------------------
#*** Ces 5 appels peuvent être remplacés par l'unique appel de la fonction AC()
#
# AC (lane, lane_sup, lane_var_sup, ax_h, ax_v)
#
#*** Mais on peut vouloir représenter d'autres plans factoriels,
#*** ou sélectionner les variables numériques supplémentaires...
#*** l'appel par fonctions séparées est plus souple.
#***-----------------------------------------------------------
#
#*** fin des lignes à copier dans l'interface
#
#***-----------------------------------------------------------
#***-------------- CODES DE TOUTES LES FONCTIONS APPELEES
#***-----------------------------------------------------------
#*** Rappel: pd, np, plt sont des variables globales pour toutes
#*** les fonctions de ac_dtmF.py
#***-----------------------------------------------------------
#*** Fonction AC_base(): appel du calcul, éditions
#
def AC_base(lane):
     X, F, psi, psi_u, phi, phi_u, pcent, vs, vp = AC_calc(lane)
     n,p = X.shape
     grafac_vp(pcent, p)
#---
     ch ="AC_file_" + lane     # nom du fichier créé
     g = open(ch, "w+")        # ouverture de "ch" (chemin)
     print ("\n The coordinates are in the created file; " + ch)
     print("\n Eigenvalues\n")
     print(vp)
     print("\n")
#---
     g.write("\n Dataset\n")
     g.write(lane)
     g.write("\n")
     g.write("\n Eigenvalues\n")
     g.write(str(vp))
     g.write("\n\n")
```

```
      pd.set_option('display.max_rows', 10)
#---
      ligne = "Coordinates of variables"
      print(ligne)
      print("\n")
      print(pd.DataFrame({'ident':X.columns, 'c_var_1': phi[:,0], 'c_var_2':
phi[:,1],'c_var_3': phi[:,2],  }))
      g.write(ligne)
      g.write("\n\n")
      info = "\n complete coord. and coord. of indiv. are in the created file:
" + ch + "\n\n"
      print(info)
      pd.set_option('display.max_rows', n)
#---
      a = pd.DataFrame({'ident':X.columns, 'c_var_1': phi[:,0], 'c_var_2':
phi[:,1],'c_var_3': phi[:,2],  })
      sa = str(a)
      g.write(sa)
      g.write("\n\n")
      ligne = "Coordinates of individuals or observations"
      g.write(ligne)
      g.write("\n\n")
#---
      aa = pd.DataFrame({'ident':X.index, 'c_ind_1': psi[:,0], 'c_ind_2':
psi[:,1],'c_ind_3': psi[:,2],  })
      saa = str(aa)
      g.write(saa)
      g.close()
      pd.reset_option('display.max_rows')
      return X, psi, psi_u, phi, phi_u
#-----------------------------------------------------
#
#*** Fonction AC_calc() : calculs de base
def AC_calc(lane):
      X = pd.read_csv(lane)
      n,p = X.shape
      ki = np.sum(X, axis = 1)
      kj = np.sum(X, axis = 0)
      k = np.sum(kj)
      fi, fj = ki/k, kj/k
      F = X/k
#---
      S = np.zeros((n,p))
      for i in range(n):
            for j in range(p):
                  S[i,j] = (F.iloc[i,j] -fi[i]*fj[j])/np.sqrt(fi[i]*fj[j])
#---  np.linalg.svd = singular values decomposition with numpy
      u, vs, tvh = np.linalg.svd(S, full_matrices=False)
      rang = min(n,p) -1   # cas de l'AC
      vp = vs*vs
#
      psi=np.zeros((n,rang))
      psi_u=np.zeros((n,rang))
# coordonnées psi des lignes (psi_u unitaire)
      for j in range(rang):
            for i in range(n):
                  psi_u [i,j] = u[i,j]/np.sqrt(fi[i])
                  psi [i,j] = u[i,j]*vs[j]/np.sqrt(fi[i])
#
```

```
      tphi = np.zeros((rang,p))   # coordonnées des colonnes
      tphi_u = np.zeros((rang,p)) # vect. unitaires des colonnes
      for jj in range(rang):
            for j in range(p):
                  tphi_u[jj,j] = tvh[jj,j]/np.sqrt(fj[j])
                  tphi[jj,j] = tvh[jj,j]*vs[jj]/np.sqrt(fj[j])
#
      trace = np.sum(vp)
      pcent = vp/trace
# tphi et tphi_u sont tranposés en phi     et phi_u
      phi = tphi.T
      phi_u = tphi_u.T
      return X, F, psi, psi_u, phi, phi_u, pcent, vs, vp
#-------------------------------------------------------
#*** Fonction grafac_vp() :Graphique des valeurs propres
def grafac_vp(pcent, p):
      plt.plot(np.arange(1, p+1), pcent)
      plt.title("Eigenvalues")
      plt.xlabel("Axes")
      plt.ylabel("Eigenvalues (as percentages of trace)")
      plt.show()
#-------------------------------------------------------
#*** Fonction graf_act(): Graphiques (plan (ax_h, ax_v) des éléments actifs)
def graf_act ( X, psi, phi, ax_h, ax_v ):
      xx = ax_h
      yy = ax_v
      n,p = X.shape
#---
      graf_col (X, phi, p, xx, yy)
      graf_rows (X, psi,  n, xx, yy)
      return
#-------------------------------------------------------
#*** Fonction graf_col(): Graphiques (plan (ax_h, ax_v) ) des variables
def graf_col ( X, phi, p, ax_h, ax_v):
      lax = 8
      xh = phi[:,ax_h - 1]
      xv = phi[:,ax_v - 1]
      fig, axes = plt.subplots(figsize = (lax,lax))
      mi_x, ma_x = min(xh), max(xh)
      axes.set_xlim (mi_x,ma_x)
      mi_y, ma_y = min(xv), max(xv)
      axes.set_ylim (mi_y,ma_y)
      FS = 10    # FS = fontsize
      for j in range(p):
            plt.annotate (X.columns[j],(phi[j,ax_h - 1], phi[j,ax_v - 1]),
fontsize = FS)
      plt.plot([mi_x,ma_x],[0,0],color = 'blue',linestyle = "-", linewidth = 1)
      plt.plot([0,0],[mi_y,ma_y],color = 'blue',linestyle = "-", linewidth = 1)
      plt.title("Variables(columns) plane "+ str(ax_h)+ ", "+ str(ax_v))
      plt.xlabel("axis " + str(ax_h))
      plt.ylabel("axis " + str(ax_v))
      plt.show()
#-------------------------------------------------------
#*** Fonction  graf_rows() : Graphiques (plan (ax_h, ax_v) ) des lignes
def graf_rows ( X, psi, n, ax_h, ax_v):
#--- cadrage
      xh = psi[:,ax_h - 1]
      xv = psi[:,ax_v - 1]
      lax = 8
```

```
      fig, axes = plt.subplots(figsize = (lax,lax))
      FS = 8   # FS = fontsize
      mi_x, ma_x = min(xh), max(xh)
      axes.set_xlim (mi_x,ma_x)
      mi_y, ma_y = min(xv), max(xv)
      axes.set_ylim (mi_y,ma_y)
      for i in range(n):
           plt.annotate (X.index[i],(psi[i,ax_h - 1], psi[i,ax_v - 1]),
fontsize = FS, color = 'red')
      plt.plot([mi_x, ma_x],[0,0], color = 'blue', linestyle = "-",linewidth =
1)
      plt.plot([0,0],[mi_y, ma_y ], color = 'blue', linestyle = "-",linewidth =
1)
      plt.title("Individuals (rows) plane " + str(ax_h)+ ", "+  str(ax_v))
      plt.xlabel("axis " + str(ax_h))
      plt.ylabel("axis " + str(ax_v))
      plt.show()
#-----------------------------------------------------
#*** Fonction graf_simult() : Graphiques simultanés(plan (ax_h, ax_v) ) des
lignes et colonnes
def graf_simult (X, psi, phi, ax_h, ax_v):
      n,p = X.shape
#--- cadrage (mêmes unités)
      xh = psi[:,ax_h - 1]
      xv = psi[:,ax_v - 1]
      xh_var = phi[:,ax_h - 1]
      xv_var = phi[:,ax_v - 1]
#--- a pour élargir le cadre
      lax = 8
      a = 0.1
      FS_ind = 8  # FS = fontsize
      FS_var = 10
      lmin = min(min(xv), min(xh),min(xv_var), min(xh_var)) - a
      lmax = max(max(xv), max(xh), max(xv_var), max(xh_var)) + a
#---
      fig, axes = plt.subplots(figsize = (lax,lax))
      axes.set_xlim (lmin,lmax)
      axes.set_ylim(lmin,lmax)
      for i in range(n):       # ind actifs
           plt.annotate (X.index[i],(psi[i,ax_h - 1], psi[i,ax_v - 1]),
fontsize = FS_ind, color = 'b')
      for j in range(p):
           plt.annotate (X.columns[j],(phi[j,ax_h - 1], phi[j,ax_v - 1]),
fontsize = FS_var, color ='r')
#--- tracé des axes
      plt.plot([-lax,lax],[0,0], color = 'red', linestyle = "-",linewidth = 1)
      plt.plot([0,0],[-lax,lax], color = 'red', linestyle = "-",linewidth = 1)
# titres et étiquettes
      plt.title("Active rows + columns, plane " + str(ax_h)+ ", "+  str(ax_v))
      plt.xlabel("axis " + str(ax_h))
      plt.ylabel("axis " + str(ax_v))
      plt.show()
#-----------------------------------------------------

#*** Fonction AC_isup() : appels des coordonnées et graphiques des lignes
supplémentaires
def AC_isup( X, psi,  phi, phi_u, lane_sup, ax_h, ax_v):
#--- Données lignes (individus) supplémentaires
      X_isup, psi_isup = rowsup(lane_sup, phi_u)
```

```
#--- Graphiques lignes (individus) supplémentaires
     graf_rows_sup (X, psi, psi_isup,  X_isup, ax_h, ax_v)
     return
#-----------------------------------------------------
#*** Fonction rowsup(): Coordonnées des lignes (individus) supplémentaires:
def rowsup(lane_sup, phi_u):
     X_isup = pd.read_csv(lane_sup)
     q,p = X_isup.shape
     X_isup_norm = np.zeros((q,p))
     for i in range(q):
          for j in range(p):
               X_isup_norm [i,j] = X_isup.iloc[i,j] /sum(X_isup.iloc[i,:])
     psi_isup = np.dot(X_isup_norm, phi_u)
     print("\n")
     print(psi_isup)
     return X_isup, psi_isup
#-----------------------------------------------------

#*** Fonction graf_rows_sup() : Graphiques (plan (ax_h, ax_v) ) des individus
supplémentaires
def graf_rows_sup (X, psi, psi_isup, X_isup, ax_h, ax_v):
#
     xh = psi[:,ax_h - 1]
     xv = psi[:,ax_v - 1]
     xh_sup = psi_isup[:,ax_h - 1]
     xv_sup = psi_isup[:,ax_v - 1]
#--- cadrage (mêmes unités), "a" pour élargir le cadre
     a = 0.1
     FS = 8  # FS = fontsize
     lmin = min(min(xv), min(xh),min(xv_sup), min(xh_sup)) - a
     lmax = max(max(xv), max(xh), max(xv_sup), max(xh_sup)) + a
     lax = 8   # largeur du graphique
#---
     n = X.shape[0]           # nombre d'indiv. actifs
     fig, axes = plt.subplots(figsize = (lax,lax))
     axes.set_xlim (lmin,lmax)
     axes.set_ylim(lmin,lmax)
     for i in range(n):       # ind actifs
          plt.annotate (X.index[i],(psi[i,ax_h - 1], psi[i,ax_v - 1]),
fontsize = FS)
     nsup = X_isup.shape[0]  # nombre d'indiv. suppl.
     for i in range(nsup):
          plt.annotate (X_isup.index[i],(psi_isup[i,ax_h - 1], psi_isup[i,ax_v
- 1]), color = 'g')
#--- tracé des axes
     plt.plot([-lax,lax],[0,0], color = 'red', linestyle = "-",linewidth = 1)
     plt.plot([0,0],[-lax,lax], color = 'red', linestyle = "-",linewidth = 1)
     # titres et étiquettes
     plt.title("Active and supplementary rows (individuals). Plane " +
str(ax_h)+ ", "+  str(ax_v))
     plt.xlabel("axis " + str(ax_h))
     plt.ylabel("axis " + str(ax_v))
     plt.show()
#-----------------------------------------------------
#*** Fonction AC_vsup(): Colonnes (variables) supplémentaires
def AC_vsup(X, psi, psi_u, phi, lane_var_sup, ax_h, ax_v):
#--- Appel données/calcul variables supplémentaires
     X_vsup, phi_vsup, q = colsup(X, psi_u, lane_var_sup)
     if (colsup == 0):
```

```
          return
#--- Appel Graphiques variables supplémentaires
     graf_col_sup (X, phi, X_vsup, phi_vsup, ax_h, ax_v)
#-------------------------------------------------
#*** Fonction colsup() : Données variables supplémentaires:
def colsup(X, psi_u, lane_var_sup):
     n, r = psi_u.shape
     X_vsup = pd.read_csv(lane_var_sup)
     n,q = X_vsup.shape #  q = nombre total de v. sup
     X_vsup_norm = np.zeros((n,q))
     for j in range(q):
          for m in range(n):
               X_vsup_norm [m,j] = X_vsup.iloc[m,j] /sum(X_vsup.iloc[:,j])
     phi_vsup = np.dot(X_vsup_norm.T, psi_u)
     print("\n")
     print(phi_vsup)
     return X_vsup, phi_vsup, q
#-------------------------------------------------
#*** Fonction graf_col_sup() : Graphiques (plan (ax_h, ax_v) ) des variables
supplémentaires
def graf_col_sup (X, phi, X_vsup, phi_vsup,  ax_h, ax_v):
     lax = 8
     q = X_vsup .shape[1]  # nombre de var. sup.
     p = X.shape[1]  # nombre de var. act.
     fig, axes = plt.subplots(figsize = (lax,lax))

#--- cadrage (mêmes unités)
     xh = phi[:,ax_h - 1]
     xv = phi[:,ax_v - 1]
     xh_sup = phi_vsup[:,ax_h - 1]
     xv_sup = phi_vsup[:,ax_v - 1]
#
     a = 0.05
     FS = 8  # FS = fontsize
     lmin = min(min(xv), min(xh),min(xv_sup), min(xh_sup)) - a
     lmax = max(max(xv), max(xh), max(xv_sup), max(xh_sup)) + a
     lax = 8
#---
     axes.set_xlim (lmin,lmax)
     axes.set_ylim(lmin,lmax)
#--- variables actives
     for j in range(p):
          plt.annotate (X.columns[j],(phi[j,ax_h - 1], phi[j,ax_v - 1]),
fontsize = FS, color = 'b')
#--- variables supplémentaires
     for j in range(q):
          plt.annotate (X_vsup.columns[j],(phi_vsup[j,ax_h - 1],
phi_vsup[j,ax_v - 1]), color = 'g')
#---
     plt.plot([-1,1],[0,0], color = 'blue', linestyle = "-", linewidth = 1)
     plt.plot([0,0],[-1,1], color = 'blue', linestyle = "-", linewidth = 1)
     plt.title("Active and supplementary columns "+ str(ax_h)+ ", "+
str(ax_v))
     plt.xlabel("axis " + str(ax_h))
     plt.ylabel("axis " + str(ax_v))
     plt.show()
#-------------------------------------------------------------------------
#-------------------------------------------------------------------------
#*** Fonction AC() Cette fonction réunit les 5 appels des 5 fonctions
```

```
#*** principales : AC_base, graf_act, graf_simult, AC_isup, AC_vsup.
#-------------------------------------------------------------------------
def AC (lane, lane_sup, lane_var_sup, ax_h, ax_v):
      X, psi, psi_u, phi, phi_u = AC_base(lane)
      graf_act (X, psi, phi, ax_h, ax_v )
      graf_simult (X, psi, phi, ax_h, ax_v)
      AC_isup( X, psi,  phi, phi_u, lane_sup, ax_h, ax_v)
      AC_vsup(X, psi, psi_u, phi, lane_var_sup, ax_h, ax_v)
      return
#-------------------------------------------------------------------------
```

Fin du code python

Le code est écrit de la façon la moins cryptique possible, de façon à permettre des adaptations et des compléments, notamment concernant la sortie des résultats numériques.
Les stylistes python pourront aussi rendre le code plus compact et élégant.