DUCHANOY Colin
FOURNIER Nicolas
LANCERY Hugo
SOUCOURRE Vincent

Bureau d'étude Ma314 : Analyse en Composantes Principales

AERO 3

M. COUFFIGNAL – M. EL MAHBOUBY

21 janvier 202

Table des matières

I. Un peu de théorie

1. $Var(Xv) = v^T Cov(X)v$

$$Var(Xv) = Cov(Xv, Xv) = Cov(\sum_{i=1}^{p} v_i x_i, \sum_{i'=1}^{p} v_{i'} x_{i'})$$

$$= \sum_{i=1}^{\rho} \sum_{i'=1}^{p} v_i v_{i'} \, Cov(x_i, x_{i'})$$

$$= \sum_{i=1}^{\rho} \sum_{i'=1}^{p} v_i v_{i'} Cov(X)$$

$$= v^T \, cov(X) v$$

2. Matrice Cov(X) diagonalisable

Soit $Cov(X)$ une matrice symétrique, alors $Cov(X)$ est diagonalisable dans une base orthonormée. Alors il existe une matrice V de passage et inversible telle que $V^{-1} = V^T$ et D une matrice diagonale vérifiant :

$$V^T Cov(X) V = V^{-1} Cov(X) V = D$$

Donc on en déduit :

$$\Leftrightarrow Cov(X) = VDV^T$$

3. k−composante principale de X

On a $Y_{k+1} = Xv_{k+1} = \sum_{i=1}^{n} v_{(k+1)i} \, X_i$

Donc $Y_k = Xv_k$

On a d'après la question 1 et 2 :

$$Cov(Xv_k, Xv_{k'}) = {v_k}^T \, cov(X) \, v_{k'}$$
$$\Leftrightarrow {v_k}^T \, VDV^T \, v_{k'}$$

Par propriété d'orthogonalité on a $V^T v_k = e_k$ avec k le k-eme élément de la base canonique.

$$\Leftrightarrow {e_k}^T \, D \, e_{k'} = D_{kk'}$$

Donc la matrice de covariance est la matrice diagonale D telles que les coefficients diagonaux correspondent à $Cov(Xv_k, Xv_{k'}) = Var(Y_k)$

Donc : $Var(Y_k) = \lambda_k$

4. Estimateur de la matrice de covariance de X, matrice « centrée réduite » R_{cr} et $proj_k$

a. Démonstration : $C \coloneqq {E_{cr}}^T E_{cr}$ est une approximation de la matrice de covariance de X

On note E_{cr}la matrice « centrée-réduite » associée à E dont les coefficients sont donnés par les variables aléatoires :

$$(E_{cr})_{jk} = \frac{X_{jk} - \overline{X_{mk}}}{\sqrt{m-1}\overline{\sigma_{mk}}}$$

On cherche à démontrer que C=${E_{cr}}^T E_{cr}$ est une approximation d'un estimateur de la matrice de covariance de X.

On suppose :

$$E\left(\left(\frac{X_{ik} - \overline{X_{mk}}}{\overline{\sigma_{mk}}}\right)\left(\frac{X_{jp} - \overline{X_{mp}}}{\overline{\sigma_{mp}}}\right)\right) \approx \frac{E((X_{ik} - \overline{X_{mk}})(X_{jp} - \overline{X_{mp}})}{E(\overline{\sigma_{mk}})E(\overline{\sigma_{mp}})}$$

On a donc : E = $(X_1 \dots \dots \dots \dots X_n)$

$$E_{cr} = (X_{1cr} \dots \dots \dots \dots X_{ncr}) \quad \text{Et} \quad {E_{cr}}^T = ({X_{1cr}}^T \dots \dots \dots \dots {X_{ncr}}^T)$$

Donc : C=$\begin{pmatrix} {X_{1cr}}^T \\ \vdots \\ {X_{ncr}}^T \end{pmatrix}(X_{1cr} \dots \dots \dots \dots X_{ncr}) = \begin{pmatrix} {X_{1cr}}^T X_{1cr} \\ {X_{1cr}}^T X_{2cr} \dots \dots \\ \vdots \\ \vdots \end{pmatrix}$

Pour tous les termes diagonaux de C on a : ${X_{ncr}}^T X_{ncr} = Var(X_{ncr})$

Les autres termes représentent les covariances : ${X_{1cr}}^T X_{2cr} = Cov(X_{1cr}, X_{2cr})$

On a donc : E(C) = $Cov\ (X_{cr})$

On peut donc généraliser à tous les termes en prenant un exemple :

$$\text{E}\ (Cov(X_{1cr}, X_{2cr})) = \text{E}\ (({X_{1cr}}^T, X_{2cr}))$$

$$E\left(\left(\frac{{X_1}^T - \overline{X_1}}{\overline{\sigma_1}}\right)\left(\frac{{X_2}^T - \overline{X_2}}{\overline{\sigma_2}}\right)\right) = \frac{1}{\overline{\sigma_1}\overline{\sigma_2}} Cov(X_1, X_2)$$

Pour tout i on a $\overline{\sigma_i} = 1$ car nous sommes en centré réduit.

Donc on peut en déduire que C=${E_{cr}}^T E_{cr}$ est une approximation d'un estimateur de la matrice de covariance de X.

b. k-directions principales v_k et décomposition en valeurs singulières de $R_{cr} = U\Sigma V^T$

Supposons donnée une réalisation R d'un m−échantillon ($Ind_1, Ind_2, \ldots, Ind_m$) i.e. R est la donnée d'une matrice de taille (m, n) :

$$R = \begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \cdots & \vdots \\ \vdots & \cdots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix} \epsilon M_{mn}(\mathbb{R})$$

Nous avons R_{cr} une matrice centrée réduite et elle possède les mêmes propriétés que E_{cr}. On admet qu'on connait une décomposition SVD de R_{cr}=$U\Sigma V^T$.

D'après la question précédente on peut déduire :

$$Cov(X) = {R_{cr}}^T R_{cr}$$

$$Cov(X) = (U\Sigma V^T)^T (U\Sigma V^T)$$

$$Cov(X) = V\Sigma U^T\, U\Sigma V^T = V\Sigma\Sigma V^T = VDV^T$$

Or on a vu à la question 2, que la matrice V de la décomposition SVD correspond à la matrice V orthogonale. On peut donc identifier que $\Sigma\Sigma = D$ et donc $Var(Y_k) = \lambda_k = \sigma_k^2$.

c. Projection de R_{cr} dans l'espace vectoriel engendré par les vecteurs de directions principales : $Vec(v_1, \ldots, v_k)$

On connait la décomposition SVD de R_{cr}= $U\Sigma V^T$ avec Σ contenant les valeurs singulières $\lambda_k = \sigma_k^2$.

Il existe alors un vecteur unitaire $v \epsilon V$ et $u \epsilon U$:

$$R_{cr}\, v = U\Sigma V^T v = U\Sigma = \sigma_k^2\, u$$

Ainsi on obtient :

$$proj_k(R_{cr}) = (\sigma_1^2\, u_1 \quad \sigma_2^2\, u_2 \ldots \sigma_k^2\, u_k)$$

Avec les u_j les -ème vecteurs colonnes de U dans la décomposition en valeurs singulières de $R_{cr} = U\Sigma V^T$.

5. $\frac{1}{n}\sum_{i=1}^{n} Var(Y_i) = 1$

$$\frac{1}{n}\sum_{i=1}^{n} Var(Y_i) = \frac{1}{n}\sum_{i=1}^{n} {\sigma_i}^2$$

R_{cr} Centrée réduite donc $\sigma_i = 1$

$$\Leftrightarrow \frac{1}{n}\sum_{i=1}^{n} 1 \; = \frac{n}{n} = 1$$

6. $Cor_i\ (Cor(Y_1, x_i)\ Cor(Y_2, x_i) \ldots . Cor(Y_k, x_i))$ et boule unité B^k de $\mathbb{R}^k$

On veut justifier que $Cor_i\ (Cor(Y_1, x_i)\ Cor(Y_2, x_i) \ldots . Cor(Y_k, x_i))$ appartient à la boule unité B^k de $\mathbb{R}^k$. Nous allons montrer que la corrélation est bornée par 1 en valeur absolue.

On sait que : $Cor\ (X, Y) = \frac{Cov\ (X,Y)}{\sigma_X \sigma_Y}$

La covariance $Cov\ (X, Y)$ entre X et Y est un produit scalaire entre ces deux vecteurs. De plus la norme associée à ce produit scalaire est $\sigma_X = \sqrt{Var(X)}$.

Soit $t \in \mathbb{R}^k$, on considère une variable $Z = X + tY$, nous obtenons :

$$Var(Z) = Var(X + tY) = \sum_{1=1}^{n} ((X_i - \bar{X})^2 + t(Y_i - \bar{Y}))^2$$

$$\Leftrightarrow \sum_{1=1}^{n} (X_i - \bar{X})^2 + 2t \sum_{1=1}^{n} (X_i - \bar{X})\,(Y_i - \bar{Y}) + t^2 \sum_{1=1}^{n} (Y_i - \bar{Y})^2$$

$$\Leftrightarrow Var(X) + 2tCov(X, Y) + t^2 Var(Y)$$

On a donc un polynôme du second degré en t. On sait aussi que $Var(Z) \geq 0$ (par définition de la variance) ce qui signifie que pour tout $t \in \mathbb{R}^k$, ce polynôme admet au maximum une racine réelle et que son discriminant est négatif.

$$\Delta = 4Cov(X,Y)^2 - 4Var(X)Var(Y)$$

Donc :

$$\Leftrightarrow 4Cov(X,Y)^2 - 4Var(X)Var(Y) \leq 0$$

$$\Leftrightarrow Cov(X,Y)^2 \leq Var(X)Var(Y)$$

$$\Leftrightarrow |Cov(X,Y)| \leq \sqrt{Var(X)}\sqrt{Var(Y)} = \sigma_{\dot{X}}\sigma_{\dot{Y}}$$

$$\Leftrightarrow \frac{|Cov(X,Y)|}{\sigma_{\dot{X}}\sigma_{\dot{Y}}} \leq 1$$

$$|Cor(X,Y)| \leq 1$$

Donc pour tout k, on a bien $|\mathrm{Cor}(\mathrm{X},\mathrm{Y})| \leq 1$,donc Cor_i appartient bien à la boule unité B^k de $\mathbb{R}^k$.

Concernant l'interprétation du cercle de corrélation, ce dernier permet de montrer les relations entre les variables. Ainsi, les variables qui sont représentées comme des flèches proches et groupées, dans la même direction sont positivement corrélées. Si jamais elles sont opposées, elles sont négativement corrélées. De plus, la distance entre les variables et l'origine représentée par la norme de la flèche mesure la qualité de représentation des variables. Ainsi, les variables qui sont loin de l'origine sont bien représentées par l'analyse en composantes principales.

II. Applications

Dans cette partie, nous détaillerons les analyses de nos résultats obtenus grâce à notre code python pour toutes les bases de données utilisées dans ce bureau d'étude.

1. Centre_Red

Tout d'abord nous définissions 2 fonctions *Esperance* et *Variance* qui nous seront utiles pour les différentes fonctions de la partie 2.

```
def Esperance(X):
    m = np.shape(X)[0]
    return(np.sum(X)/m)

def Variance(X):
    m = np.shape(X)[0]
    return(np.sum(((X-Esperance(X))**2)/(m-1)))
```

Nous pouvons maintenant définir la fonction *centre_red* qui prend en entrée une matrice R et qui renvoie la matrice R_{cr} centrée réduite associée. On récupère la taille de R (m,n) et on définit R_{cr} de même taille que R avec *np.zeros(m,n)*. On définit Xa une nouvelle variable prenant une par une les colonnes de R

Puis on remplace chaque colonne de R_{cr} en effectuant pour chaque coefficient de la colonne *Xa-Esperance (Xa)/*$\sqrt{Variance(Xa)}$.

```
for i in range(n):

Xa = R[:, i]
Rcr[:, i] = (Xa-Esperance(Xa)) / np.sqrt(Variance(Xa))
```

Ainsi on obtient notre nouvelle matrice centrée réduite.

2. Approx

Nous allons maintenant définir la fonction *approx(R,k)*. Pour cela on récupère la matrice centrée réduite souhaitée grâce à la fonction précédente qu'on assigne dans une nouvelle variable X.

On effectue une décomposition SVD de X afin obtenir trois matrices U, S,V^T(qu'on transposera et qu'on stockera dans une nouvelle variable V.

On définit une liste vide *projk* dans laquelle on viendra stocker toutes les valeurs.

```
u, s, vt = np.linalg.svd(X)
v=vt.T
projk = []
```

On réalise donc une boucle allant jusqu'à k. Pour chaque i, on va récupérer les colonnes de U et V qu'on va *reshape* sous forme de vecteur ligne (*uj* et *uk*).

Ainsi on peut donc obtenir notre $Y_k = X * v_k$ et avoir par définition (cf. Partie 1) *projk.* Pour chaque valeur de $projk_i$ on insère la valeur Var (*Yk)*uj* afin d'obtenir un vecteur ligne pour chaque i. On redimensionne *projk* en utilisant la fonction *np.block* afin que chaque colonne de *projk* correspondent à un i donné. (Dans le cas où on a approx(E,2) *projk* sera un array de taille 150,2).

```
        projk.insert(i, Variance(Yk)*uj)
    projk = np.block([projk[i] for i in range(k)])
    return(projk)
```

3. Correlationdirprinc

Afin de définir cette fonction, il est important de créer 2 fonctions distinctes : *Covariance* et *Cor* (pour la corrélation) qui nous serviront par la suite.

```
def Covariance(X, Y):

    m = np.shape(X)[0]
    return(np.sum((X-Esperance(X))*(Y-Esperance(Y)))*(1/m))

def cor(X, Y):
    Cori = []
    m, n = np.shape(X)
    for i in range(n):
        Corik = Covariance(X[:, i], Y) / (Variance(X[:, i])*Variance(Y))**0.5
        Cori.insert(i, Corik)
    return(Cori)
```

Nous pouvons donc définir la fonction *Correlationdirprinc(R,k)*. On récupère la matrice souhaitée puis on lui assigne une nouvelle valeur (X dans notre cas) puis on applique la fonction *centre_red()* à la matrice R pour l'avoir en centrée réduite.

On appelle la fonction approx(R,k) qu'on insère dans une nouvelle variable Y qui nous renvoie projk. On crée une liste vide Cori qui contiendra les différents éléments de la corrélation.

```
X = centre_red(R)
Y = approx(X, k)
Cori = []
```

Ensuite on effectue pour i allant jusqu'à k, la fonction précédemment défini, *Correlation* entre la matrice X et chaque colonne de Y qu'on stocke dans la variable corik. Ainsi chaque coefficient de Cori contiendra la valeur de Corik. Dans le cas où on réalise approx(E,2), on se retrouvera avec une matrice Cori de taille 2,4 car il y'a quatre individus. Chaque colonne de Cori nous donnera la position d'un individu dans le cercle des corrélations.

```
for i in range(k):
        Corik = cor(X, Y[:, i])
        Cori.insert(i, Corik)
    return(Cori)
```

La valeur en 1ère ligne nous donnera la position en x sur le cercle de corrélation et la valeur de la seconde ligne nous donnera la position en y.

4. ACP2D

Cette fonction est l'une des fonctions finales qui va nous permettre de tracer les variances de chaque composante ainsi que la participation de chaque variance. On utilisera la base de données des iris pour montrer son bon fonctionnement. Cette fonction se divise en plusieurs parties : la première permettant de tracer la variance de chaque composante dans un graphique en barre, la deuxième permettant de tracer un camembert montrant le pourcentage de participation de chaque variance à la variance totale, la troisième permettant de tracer le nuage de points ainsi que leur label ligne associé différent, la dernière permettant de tracer le cercle de corrélation.

En premier lieu, nous réutilisons les matrices décrites précédemment pour avoir les bonnes données centrées réduites à utiliser. On récupère donc notre matrice X et notre matrice Y_k de 2 colonnes regroupant les variances de chaque composante ainsi que le label ligne correspondant.

```
    m,k=np.shape(X)                    #dimensions de X
    Yk = np.zeros(shape=(k,2), dtype=object) # matrice Yk de 2 colonnes
    for i in range(k):                          #On remplit cette matrice Yk
        vk = v[:, i].reshape(len(v), 1)
        Yk[i,0]='$Y_{}$'.format(i+1) # label Yk
        Yk[i,1]=Variance(X@vk)       # valeur de la variance
correspondante
    Yksum=0
    for i in range (len(Yk)):
        Yksum+=Yk[i,1]
    ind=Yk[:,0]          #Liste contenant les labels
    Ykvalues=list([])    #Création d'une nouvelle liste qui ne contient que
les valeurs des différentes variances pour tracer le graphique en barres
    for i in range (k):  #On récupère les valeurs de variance contenue
dans Yk et on les reconvertie du foramt object vers float
        Ykvali=float(Yk[i,1])
        Ykvalues.append(Ykvali)
    labelsYkvalues=sorted(Ykvalues,reverse=True)
```

On peut alors tracer le graphique en barres en utilisant les fonctions de la bibliothèque *matplotlib* :

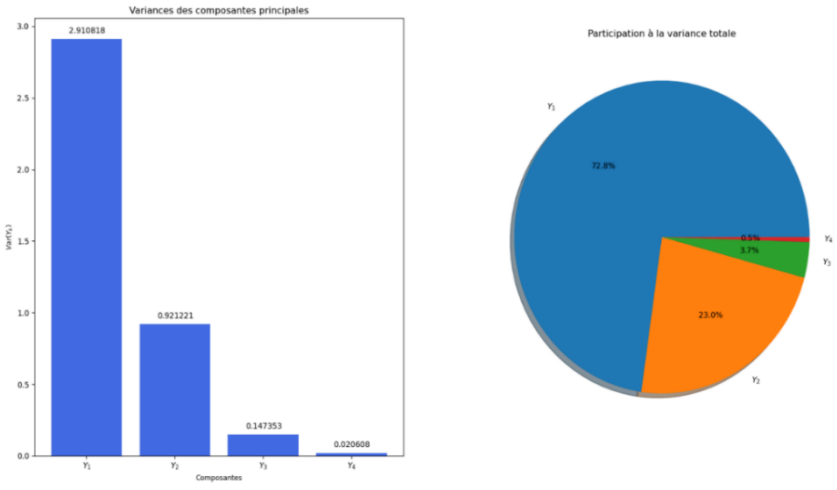
```
    ax1.bar(ind,Ykvalues,width=0.8,color='royalblue',)
    ax1.set_xlabel('Composantes',fontsize=9)
    ax1.set_ylabel('$Var(Y_{k})$',fontsize=9)
    ax1.set_title('Variances des composantes principales')

    for index, value in enumerate(labelsYkvalues):                #On place
les valeurs de chacune des barres au-dessus de celles-ci
        ax1.text(index-0.2, value+0.04, str(round(value, 6)))
```

La deuxième étape est de calculer le pourcentage de chaque variance de composantes. Pour cela on multiplie par 100 chaque variance que l'on divise ensuite par la somme des variances. Chaque pourcentage est mis dans une liste appelée *Ykpercent*. On peut alors tracer un camembert avec la *fonction .pie* et le paramètre *autopct* nous permet d'avoir la valeur du pourcentage sur chaque part du graphe :

```
    Ykpercent=[]
    for i in range(len(Ykvalues)):
        Ykpercent.append((Ykvalues[i]*100)/Yksum)

    ax2.pie(Ykpercent,labels=ind
            ,autopct = lambda Ykpercent: str(round(Ykpercent, 1)) +
'%',shadow=True) # "autopct" permet d'ajouter sur chacune des parts la
proportion en % qu'elle représente
    ax2.set_title('Participation à la variance totale')
```

Ces parties de code nous permettent d'obtenir les graphiques suivants :

Figure 1 - Valeurs des variances et pourcentage explicatif de celles-ci pour les composantes principales



On constate que dans le cas de la base de données des iris, la variance de la première composante principale représente quasiment 75% de la variance totale.

On souhaite maintenant tracer le nuage de points correspondant à la base de données des iris. Pour cela, on réutilise la fonction *Approx()* détaillée précédemment. De plus, nous avons décidé de colorier chaque point sur le graphique en fonction de son label ligne (pour

cette base de données, le label correspond à l'espèce de fleur : 0, 1 ou 2). Pour cela, nous avons généré un dictionnaire permettant d'associer une couleur à chaque label ligne différent. La couleur associée est complètement aléatoire pour cette raison il se peut que certaines fois deux couleurs similaires se retrouvent à côté. C'est également pour cette raison que si l'on lance deux fois la même fonction les couleurs changeront :

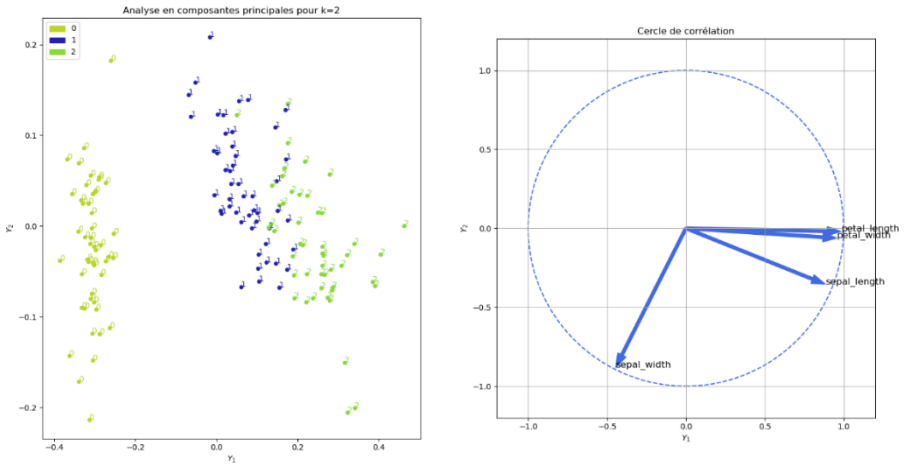
```
    Dico = {}
    legende= []
    n = len(set(labelsligne))
    lbl = labelsligne
    App = (approx(R,2))


    for i in labelsligne :
        if i not in Dico :
            color = "%06x" % random.randint(0, 0xFFFFFF)
            Dico[i] = color
            legende.append(i)
```

Finalement, pour le graphe du cercle de corrélation, nous utilisons la fonction *draw_circle* pour tracer le cercle de rayon 1. Ensuite, pour tracer les flèches incarnant les corrélations, on pose des vecteur Crx et Cory. Pour avoir les valeurs des vecteurs en abscisse et en ordonnée, on utilise la fonction *Correlationdirprinc(R,k)* et on applique la fonction *.arrow* pour tracer les flèches et la fonction *.annotate* pour afficher leur label ligne :

```
    Corx=correlationdirprinc(R, 2)[0]
    Cory=correlationdirprinc(R, 2)[1]
    #Tracé des flèches et de leur label
    for i in range(np.shape(R)[1]):

ax4.arrow(0,0,Corx[i],Cory[i],width=0.02,length_includes_head=True,color
='royalblue')
        ax4.annotate(text=labelscolonne[i], xy=(Corx[i],Cory[i]),
fontsize=12)
```

On obtient alors les deux graphes suivants :

Figure 2 - Représentation des données dans le plan (Y1, Y2) et le cercle de corrélations dans ce même plan

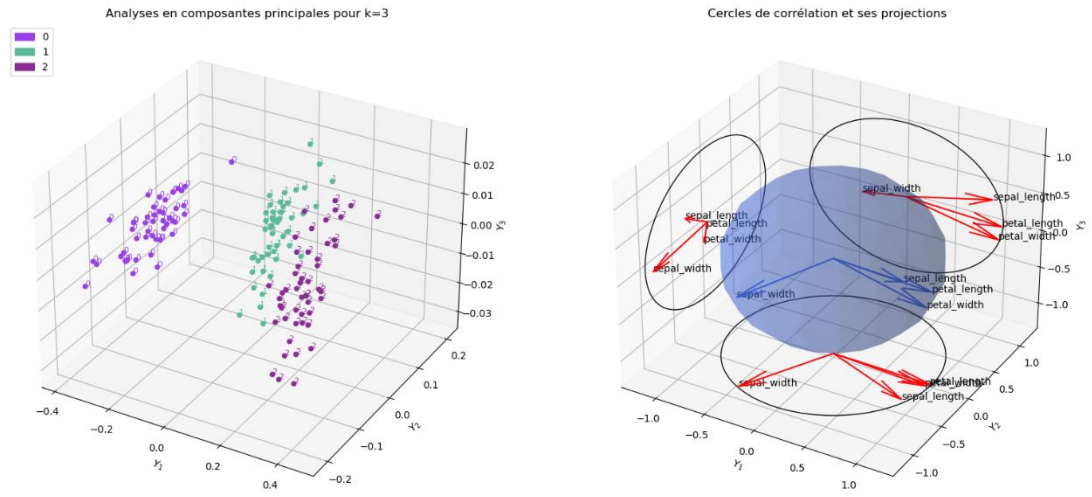


Grâce au graphe présent à droite, on peut déterminer les variables qui sont étroitement corrélés, c'est-à-dire dans notre cas, les paramètres qui jouent un rôle lié dans la détermination d'une espèce d'iris. On constate que selon l'axe des abscisses, la longueur et la largeur du pétale de la fleur étudiée sont représentées par des flèches très similaires et très proches, on peut donc en conclure que ces deux variables sont importantes pour la différenciation des espèces d'iris.

5. ACP3D

La fonction *ACP3D()* est très similaire à la fonction précédente. Les méthodes d'utilisation des données sont relativement identiques, on va juste utiliser une donnée en plus pour chaque composante pour les tracer en 3D. Les fonctions que l'on utilise pour ces tracés sont dans la bibliothèque *mpl_toolkits.mplot3d*. On peut alors tracer les deux nouveaux graphes suivants :

Figure 2 – Représentation des données dans l'espace (Y1, Y2, Y3) et le cercle de corrélations dans ce même espace



Les interprétations sont les mêmes que pour les graphes en 2D.

6. ACP

Cette fonction est la fonction finale de notre programme. Elle permet de tracer la variance de chaque composante dans un graphique en barres et un camembert montrant le pourcentage de participation de chaque variance à la variance totale, comme la fonction *ACP2D()*. Elle permet de tracer en plus de cela la matrice de sortie de la fonction *correlationdirprinc(R,k)*. Elle permet également de respecter la règle de Kaiser démontrée dans la partie 1.

On définit la règle de Kaiser respectant les conditions vues à la question 5 de la partie 1. On centre et réduit notre matrice E puis on effectue la décomposition SVD de cette matrice afin d'obtenir les valeurs singulières. Ainsi on demande pour quelle valeur de k les conditions de Kaiser sont respectés et l'algorithme nous renvoie k. Ainsi, on demande pour quelle valeur de k les conditions de Kaiser sont respectés et l'algorithme nous renvoie k.

```
def Kaiser(E,epsilon):
    Y=centre_red(E)
    epsilon=10**(-1)
    U,S,VT=np.linalg.svd(Y)

    for k in range(len(S)):
        if (S[k])**2>1-epsilon:
            if (S[k+1])**2<1-epsilon:

                q=k

    return(q)
```

Ainsi on obtient les différents k permettant de respectant la règle de Kaiser.

Base de données	Iris	Pizzamod	Howellmod	Winequality-red	Breast_cancer_data
Valeur de k	Pas de k	4	Pas de k	Pas de k	3

Pour certaines bases de données on ne trouve pas de k respectant la règle de Kaiser, cela est dû notamment à des valeurs singulières trop grandes et pas assez proches de 1.

Toutes les fonctions sont présentes dans le fichier ACP.py que l'on importe dans le fichier main.py. Dans ce fichier, on exécute juste les fonctions pour obtenir les tracés pour chaque base de données (les graphes et analyses des autres bases de données sont présentes en annexes).

III. (Bonus) Création de catégories par l'algorithme des k-moyennes

1. Algorithme des k-moyennes

a. Etape 1

```
def etape1(A, k):
    m, n = np.shape(A)
    nb_random = np.random.randint(m, size=(1, k))[0]
    uj0 = []
    for i in range(k):
        uj0.insert(i, A[nb_random[i], :])
    return(np.asarray(uj0))
```

L'étape 1 nous renvoie une matrice de k-vecteurs lignes qui sont des vecteurs lignes aléatoires de la matrice d'entrée A.

Exemple du return de l'étape 1 avec :

```
Kmoy(E, k=3, epsilon=10**-1)
```

$S_1^{(1)}$ {

$S_3^{(1)}$ {

$S_2^{(1)}$ {

```
Matrice uj0:
 [[4.6 3.4 1.4 0.3]
 [6.  3.4 4.5 1.6]
 [5.1 3.8 1.6 0.2]]
Matrice uj0:
 [[4.9 2.5 4.5 1.7]
 [5.6 3.  4.5 1.5]
 [4.8 3.1 1.6 0.2]]
Matrice uj0:
 [[6.2 2.9 4.3 1.3]
 [6.2 2.8 4.8 1.8]
 [6.9 3.1 5.4 2.1]]
```

b. Etape 2

```
def etape2(A, u0j, k, j) :
    m = np.shape(A)[0]
    Sj1 = []
    p = []
    for i in range(k):
        if i != j:
            p.append(i)
    for i in range(m):
        norme1 = np.linalg.norm(A[i, :]-u0j[j], 2)
        if norme1 <= np.linalg.norm(A[i, :]-u0j[p[0]], 2):
            if norme1 <= np.linalg.norm(A[i, :]-u0j[p[1]], 2):
                Sj1.insert(i, A[i, :])
    return(np.asarray(Sj1))
```

L'étape 2 nous renvoie une matrice de k-vecteurs lignes qui sont des vecteurs lignes aléatoires de la matrice d'entrée A.

Malheureusement nous n'avons pas réussi à bien coder cette étape ce qui nous a freiné sur la suite.

En effet en ajoutant les vecteurs dont la norme avec le $u_j^{(0)}$ est plus petite qu'avec le avec $u_p^{(0)}$ avec j ∈ $[\![1, k]\!] / \{j\}$.

Exemple du return de l'étape 1 avec :

```
Kmoy(E, k=3, epsilon=10**-1)[0]
```

	0	1	2	3
0	7.0	3.2	4.7	1.4
1	6.4	3.2	4.5	1.5
2	6.9	3.1	4.9	1.5
3	5.5	2.3	4.0	1.3
4	6.5	2.8	4.6	1.5
..	...	...	...	...
86	6.7	3.0	5.2	2.3
87	6.3	2.5	5.0	1.9
88	6.5	3.0	5.2	2.0
89	6.2	3.4	5.4	2.3
90	5.9	3.0	5.1	1.8

[91 rows x 4 columns]

Nous nous attendions à récupérer un $S_1^{(1)}$de taille≈(50,4), mais a la place la taille de $S_1^{(1)}$ est de≈(90,4) et varie énormément à chaque relancement de la fonction.

c. Etape 3

```
def etape3(Sj, k):
    u1 = []
    sum = np.zeros((np.shape(Sj[0])))
    for j in range(k):
        for ai in Sj:
            sum = sum+ai
        u1.insert(j, sum/np.shape(Sj)[0])
    return(np.asarray(u1))
```

On calcule ici les nouveaux barycentres des partitions $S_j^{(1)}$pour les insérer à la fin dans une matrice tel que (pour k=3) :

$$\mu^{(1)} := \begin{pmatrix} \mu_1^{(1)} \\ \mu_2^{(1)} \\ \vdots \\ \mu_k^{(1)} \end{pmatrix}$$

```
Matrice u10 :
 [[ 5.01818182  3.32181818  1.63272727  0.31454545]
 [10.03636364  6.64363636  3.26545455  0.62909091]
 [15.05454545  9.96545455  4.89818182  0.94363636]]
Matrice u11 :
 [[ 7.15  3.17  6.06  2.16]
 [14.3   6.34 12.12  4.32]
 [21.45  9.51 18.18  6.48]]
Matrice u12 :
 [[ 6.1         2.82666667  4.704       1.59066667]
 [12.2         5.65333333  9.408       3.18133333]
 [18.3         8.48       14.112       4.772     ]]
```

d. Etape 4 et 5

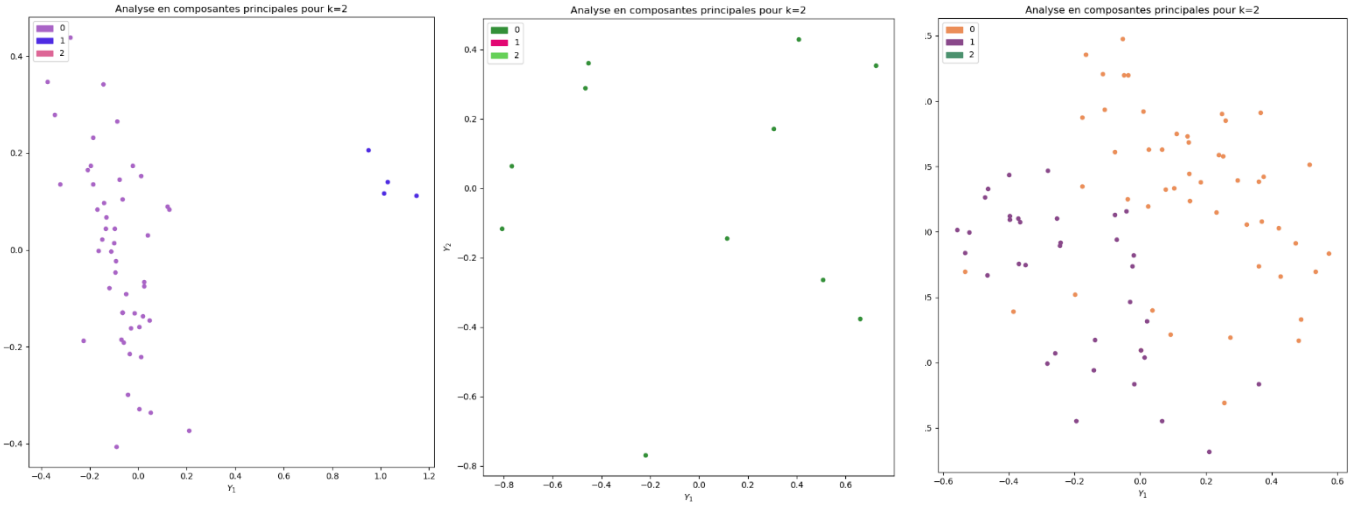
```
def etape45(A, k, epsilon):  # étape 4 et 5 qui utilise les fonctions
précedentes
    f = 1
    uj0 = etape1(A, k)
    ujk = []
    Sj1 = etape2(A, uj0, k, f-1)
    uj1 = etape3(Sj1, k)
    u = [uj0, uj1]
    S = [Sj1]
    # itération de l'étape 2 et 3 f fois et renvoie S
    while np.linalg.norm(u[f-1]-u[f]) > epsilon and f < k:

        Sjk = etape2(A, uj0, k, f)
        ujk = etape3(Sjk, k)
        f = f+1
        u.insert(f, ujk)
        S.insert(f, Sjk)

        u.insert(f, ujk)
    return(S)
```

Cette fonction consiste juste a faire tourner les fonctions étape 2 et 3 tant que la condition n'est pas vérifié :

```
while np.linalg.norm(u[f-1]-u[f]) > epsilon and f < k:
```

e. Etape 5

```
def Kmoy(A, k=0, epsilon=10**-1):
    """
    Parameters
    ----------
    A : Matrice d'entrée
    k : int >= 2. The default is 0
    epsilon : float. The default is 10**-1.
    Returns
    -------
    S
    """
    S = etape45(A, k, epsilon)
    return(S)
```



Les résultats que nous obtenons ne correspondent pas à ce que nous devons obtenir. Ici nous affichons séparément les différentes partitions de S. Les différentes couleurs sur un même graphe doivent correspondre aux erreurs dans les catégories.

En théorie les trois graphes devraient correspondre à quelques erreurs près aux catégories données par les botanistes.
On constate que le premier semble correspondre à la catégorie 0 à la différence près que les données semblent avoir été inversées (effet miroir) et retournées de 180°.
Les deux graphes suivants eux ne correspondent pas aux catégories. Le nombre d'élément est soit trop faible (graphe 2) soit trop grand (graphe 3).

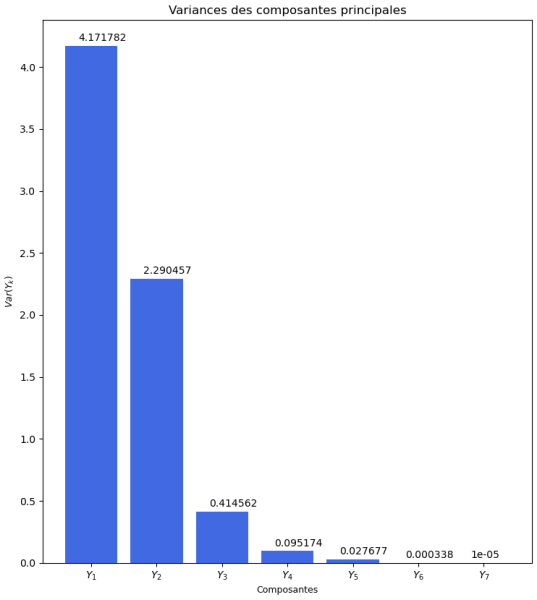
IV. Annexes

On va maintenant donner les graphes pour les autres bases de données utilisées dans ce bureau d'étude ainsi qu'une rapide interprétation de leur cercle de corrélation. Les capture d'écrans ont été réalisées pour k=2

Base de données Pizzamod

ACP2D

Figure 1 – Valeurs des variances et pourcentages explicatifs de celles-ci pour les composantes principales



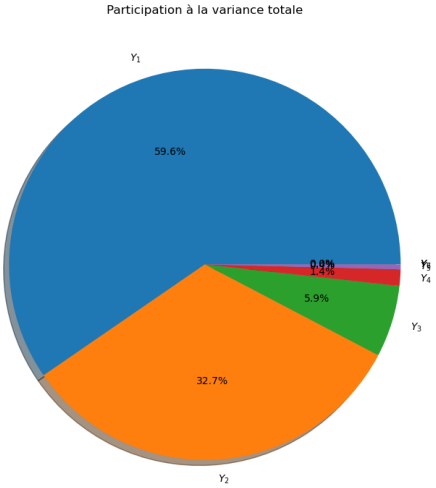
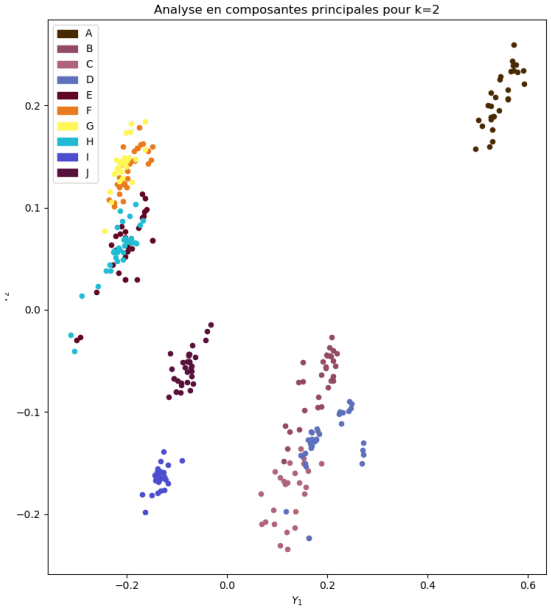
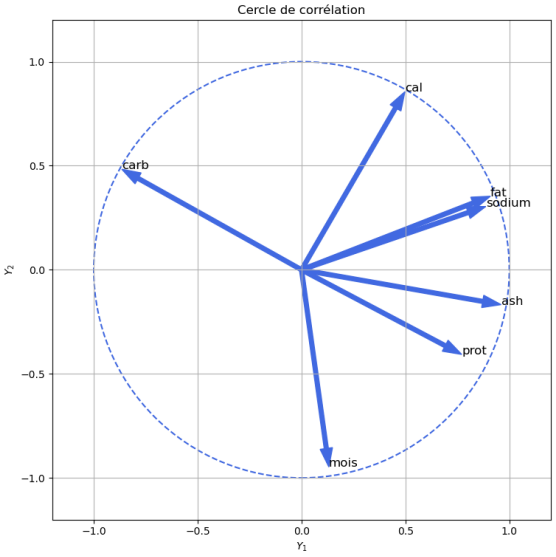


Figure 2 – Représentation des données dans le plan (Y1, Y2) et le cercle de corrélations dans ce même plan

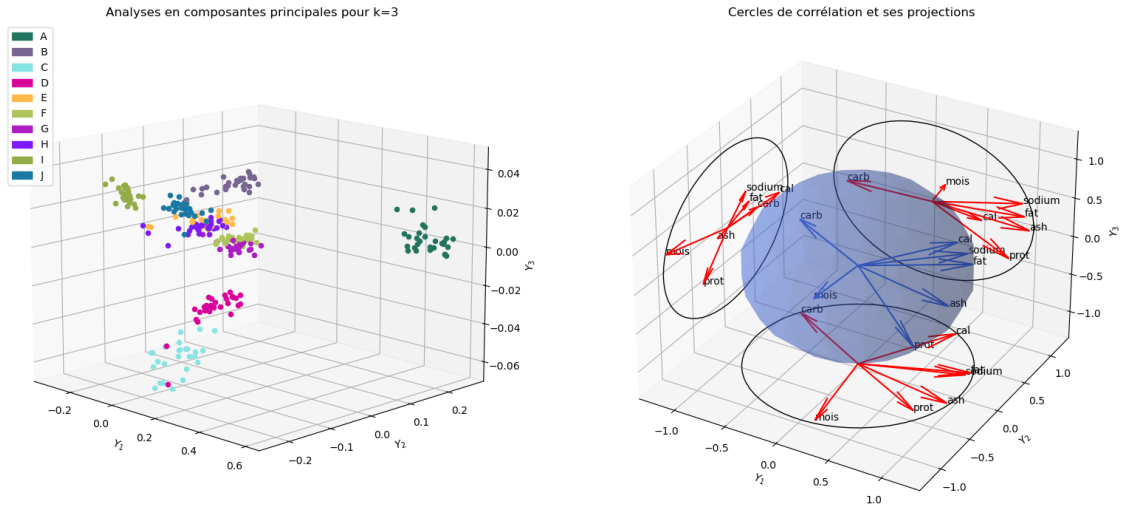




D'après le cercle de corrélation, on constate que les flèches de la graisse (« fat ») et le sodium, le sel sont très proches et de même longueur. Ainsi, ces deux variables sont très corrélées positivement. On remarque aussi que les flèches des glucides (« carb ») et des protéines (« prot ») sont opposées. Ces deux variables sont donc négativement corrélées.

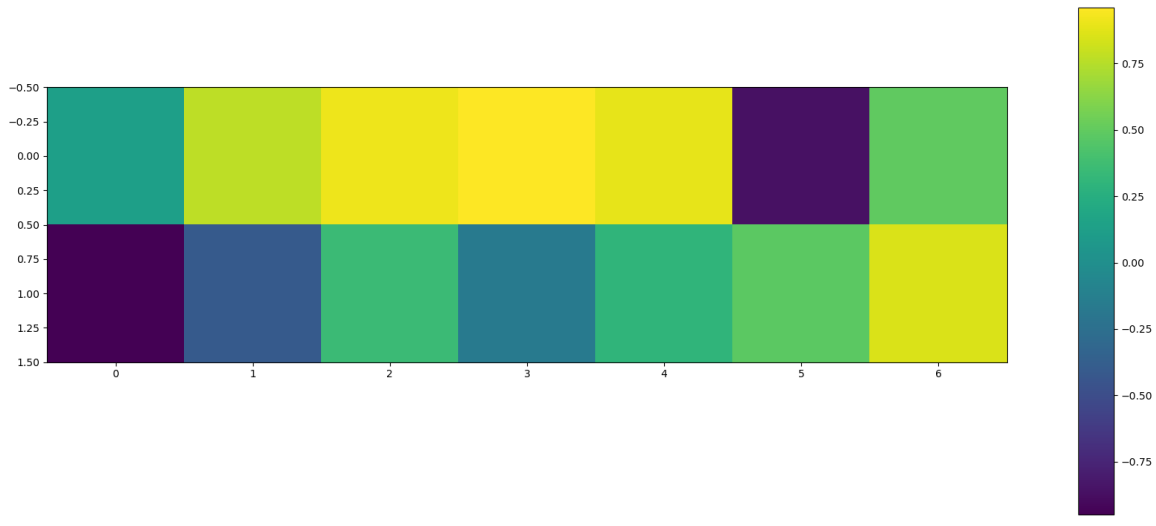
ACP3D

Figure 2 – Représentation des données dans l'espace (Y1, Y2, Y3) et le cercle de corrélations dans ce même espace



ACP

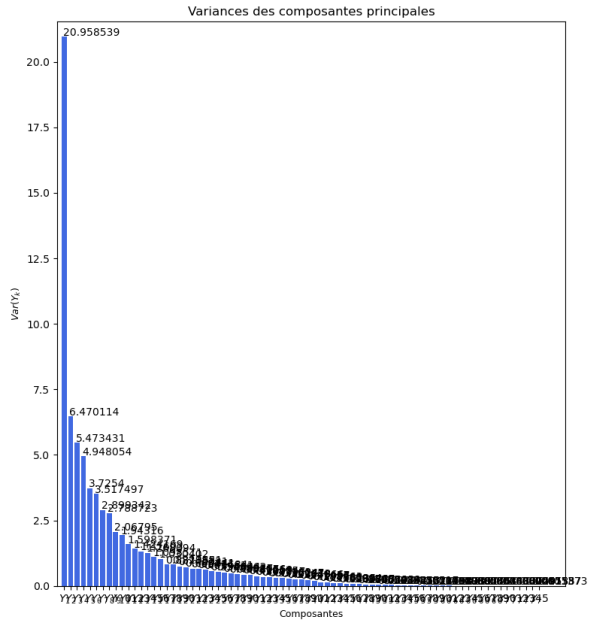
Figure 2 – Représentation de la matrice de corrélation entre les Yi et les Xj (k=2 d'après la règle de Kaiser avec Epsilon = 10^{-1})



Base de données Howellmod

ACP2D

Figure 1 – Valeurs des variances et pourcentage explicatif de celles-ci pour les composantes principales



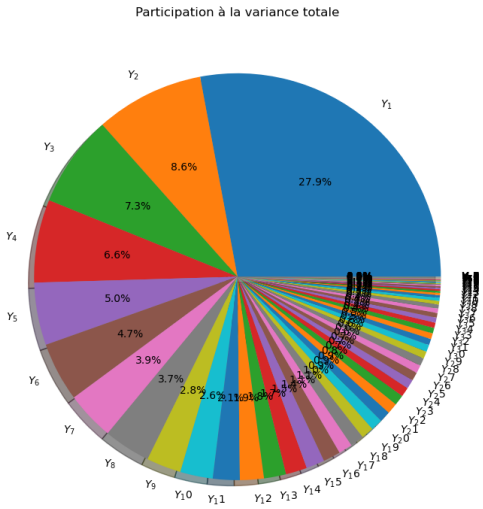
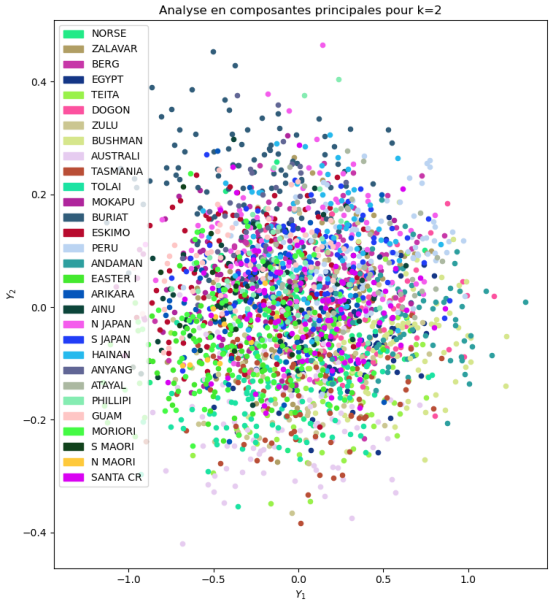
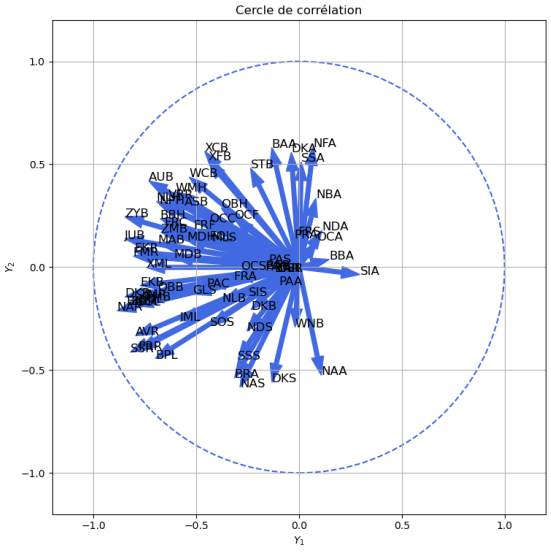


Figure 2 – Représentation des données dans le plan (Y1, Y2) et le cercle de corrélations dans ce même plan

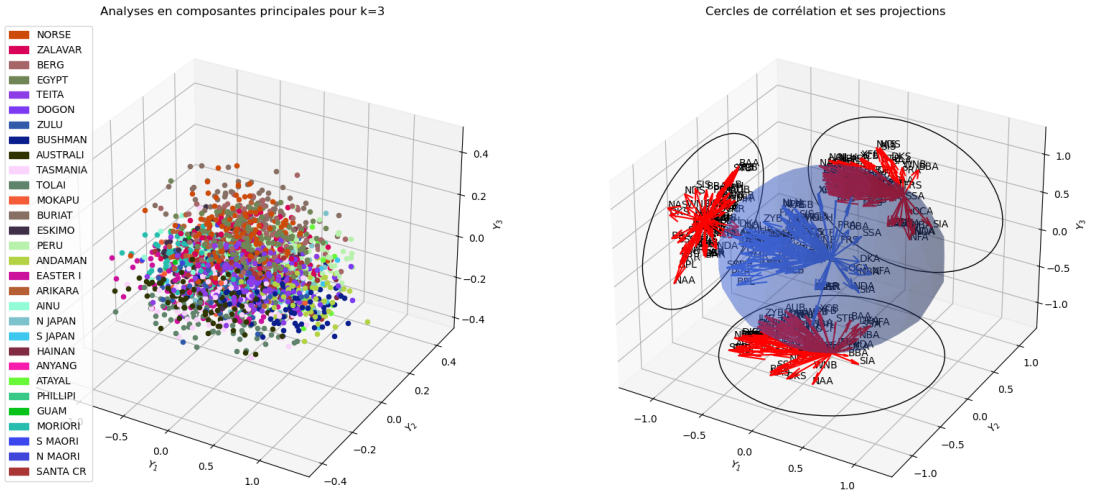




Nous ne savons pas à quoi correspond la base de données Howellmod. Il est compliqué d'interpréter les résultats obtenus sur le cercle de corrélation.

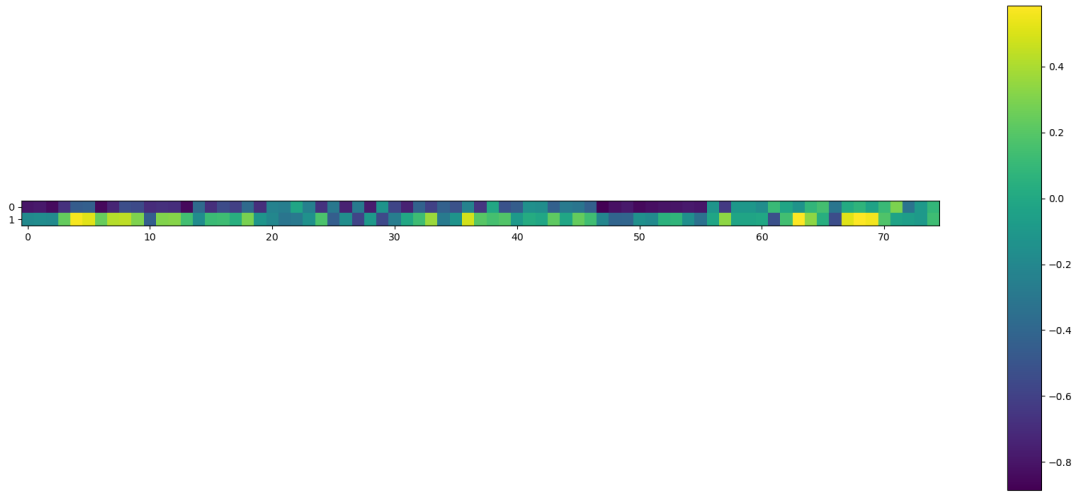
ACP3D

Figure 2 – Représentation des données dans l'espace (Y1, Y2, Y3) et le cercle de corrélations dans ce même espace



ACP

Figure 2 – Représentation de la matrice de corrélation entre les Yi et les Xj (k=2 d'après la règle de Kaiser avec Epsilon = 10^{-1})

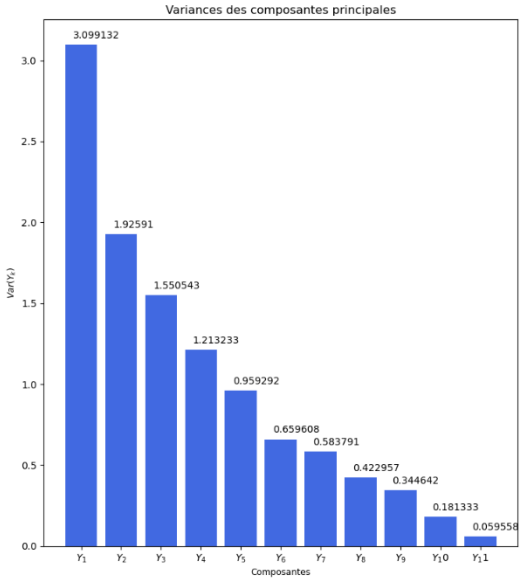


Base de données winequality-red

Cette base de données répertorie différents vins catégorisés selon leur qualité. Cette base donne plusieurs caractéristiques sur les vins comme les sulfates présents dedans, le taux d'alcool, la densité etc.

ACP2D

Figure 1 – Valeurs des variances et pourcentage explicatif de celles-ci pour les composantes principales



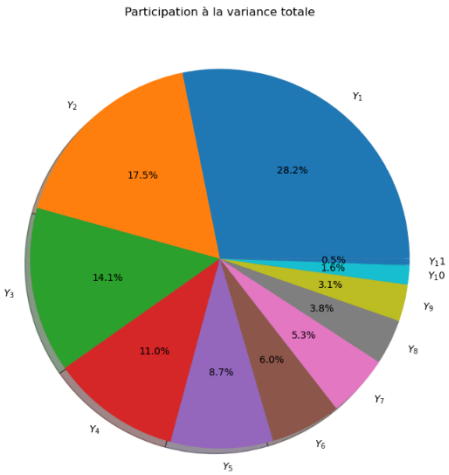
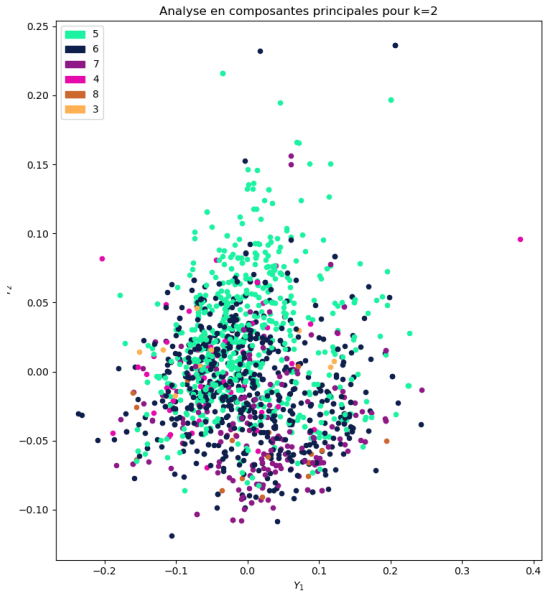
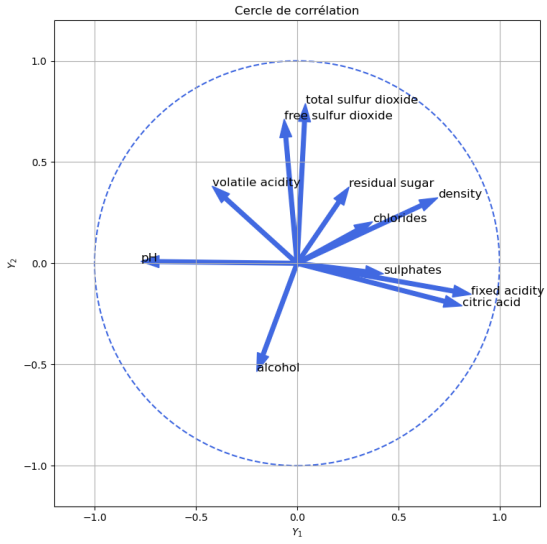


Figure 2 – Représentation des données dans le plan (Y1, Y2) et le cercle de corrélations dans ce même plan

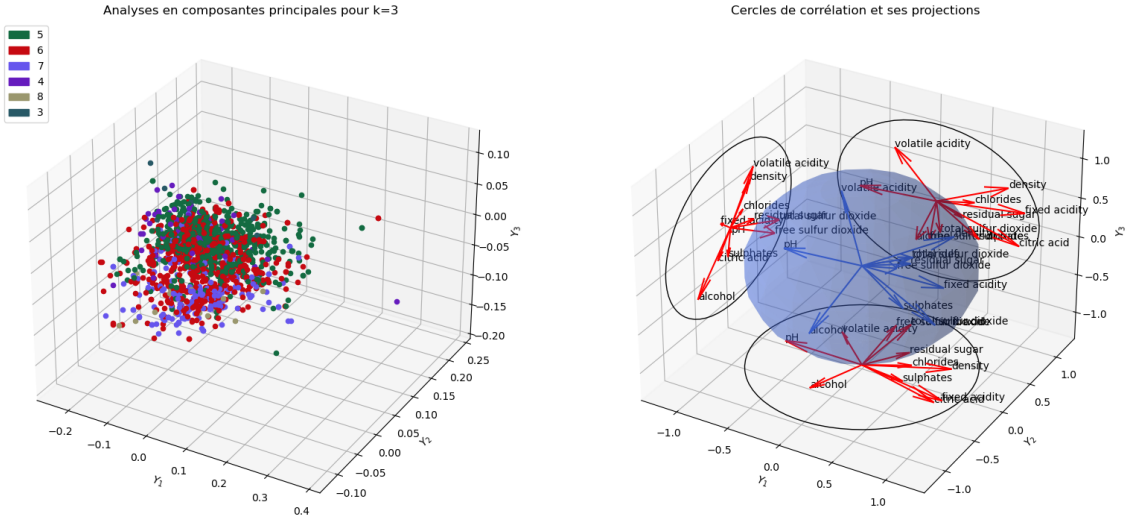




On constate que l'acidité et les acides citriques sont corrélées positivement et que les variables de dioxyde de sulfure également.

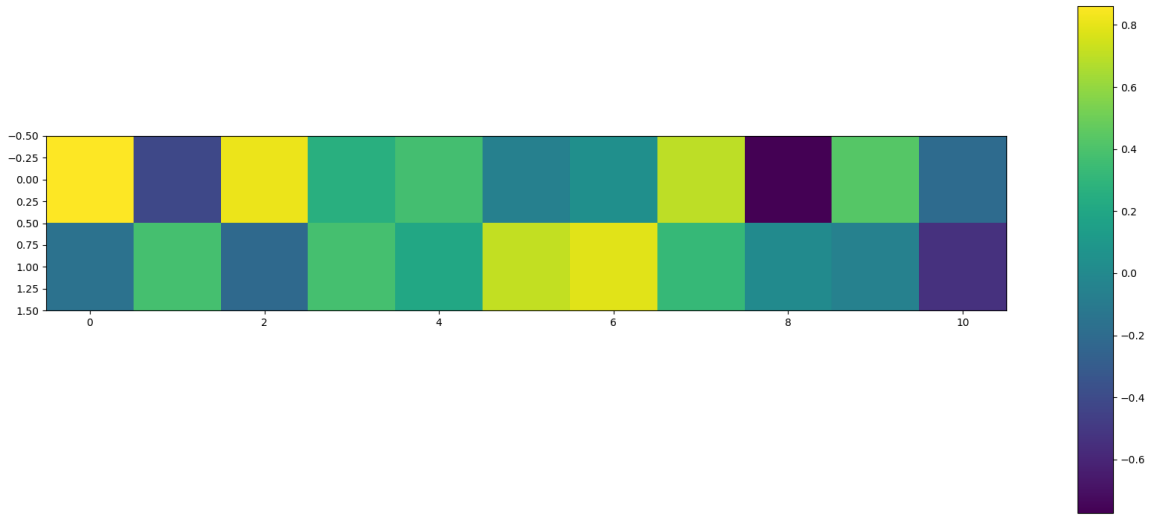
ACP3D

Figure 2 – Représentation des données dans l'espace (Y1, Y2, Y3) et le cercle de corrélations dans ce même espace



ACP

Figure 2 – Représentation de la matrice de corrélation entre les Yi et les Xj (k=2 d'après la règle de Kaiser avec Epsilon = 10^{-1})

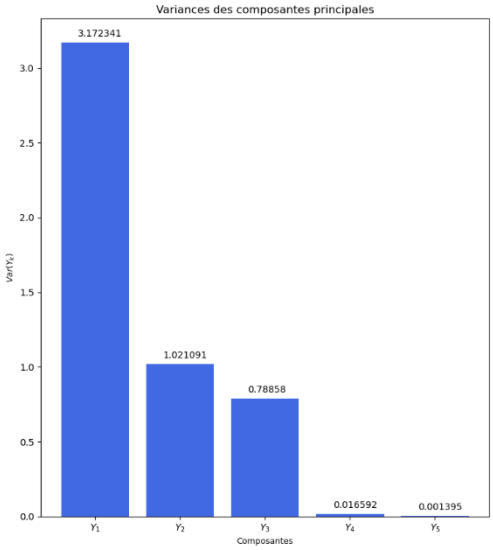


Base de données Breast_cancer_data

Cette base de données répertorie les informations relatives à des zones suspectes du sein possiblement touchées par un cancer (périmètre moyen, rayon moyen, texture, rugosité…) catégorisées selon un diagnostic (bénigne ou maline).

ACP2D

Figure 1 – Valeurs des variances et pourcentage explicatif de celles-ci pour les composantes principales



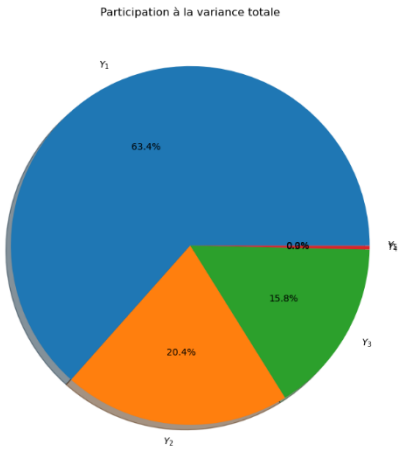
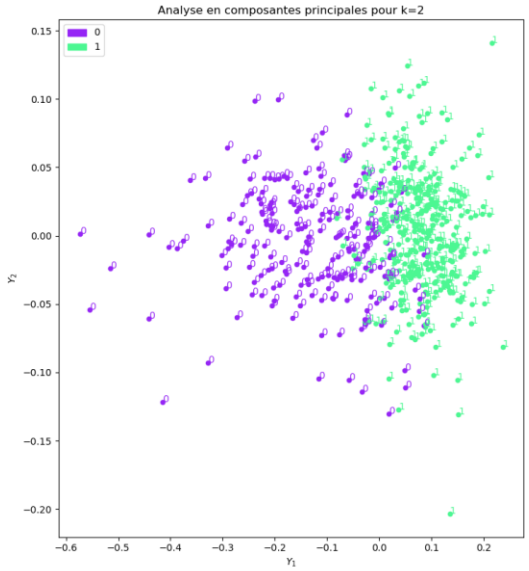
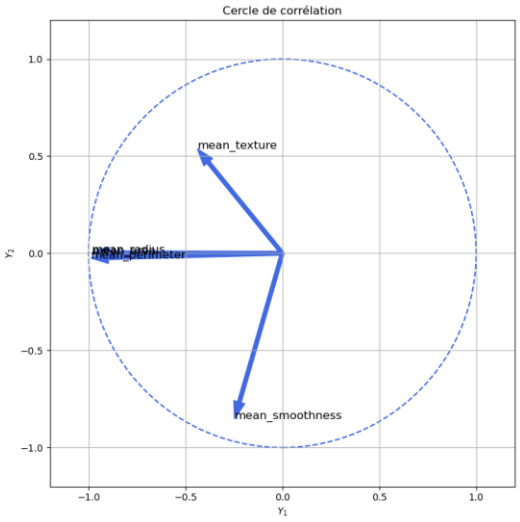


Figure 2 – Représentation des données dans le plan (Y1, Y2) et le cercle de corrélations dans ce même plan

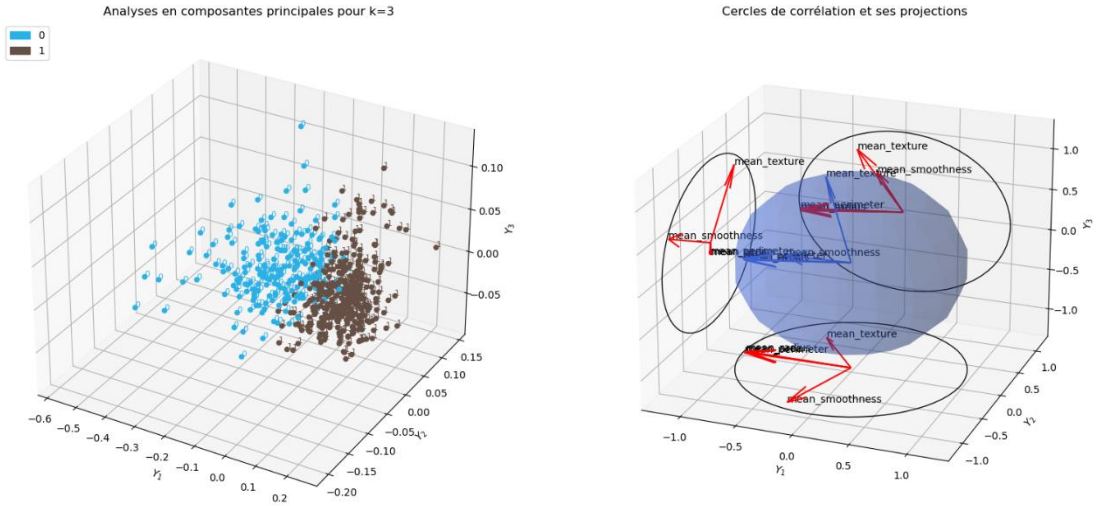




Ici, on peut observer que les variables de l'aire moyenne, le périmètre moyenne et le rayon moyen sont positivement corrélées.

ACP3D

Figure 2 – Représentation des données dans l'espace (Y1, Y2, Y3) et le cercle de corrélations dans ce même espace



ACP

Figure 2 – Représentation de la matrice de corrélation entre les Yi et les Xj (k=2 d'après la règle de Kaiser avec Epsilon = 10^{-1})