

Statistique Avec Python

De la théorie à la pratique

Duvérier DJIFACK ZEBAZE



Statistique avec Python : de la théorie à la pratique

Tome 1

Duvérier DJIFACK ZEBAZE

Duvérier DJIFACK ZEBAZE

Statistique avec Python : de la théorie à la pratique

Tome 1

SOMMAIRE

I Statistique exploratoire	viii
1 Analyse en Composantes Principales (ACP)	1
1.1 Données et problématique de l'étude	2
1.1.1 Description des données	2
1.1.2 Problématique de l'ACP	4
1.1.3 Transformation des données	9
1.1.4 La notion d'inertie	13
1.2 Mise en oeuvre de l'ACP	14
1.2.1 Diagonalisation de la matrice des corrélations	14
1.2.2 Décomposition en valeurs singulières	16
1.2.3 Choix du nombre d'axes	18
1.3 Représentation des individus et des variables	19
1.3.1 Nuage des individus	19
1.3.2 Nuage des variables	21
1.3.3 Relation entre les représentations des nuages N_I et N_K	25
1.3.4 Représentation simultanée : graphe biplot	27
1.4 Outils pour l'aide à l'interprétation	28
1.4.1 Analyse du point de vue des individus	29
1.4.2 Analyse du point de vue des variables	31
1.5 Éléments supplémentaires	33
1.5.1 Individus supplémentaires	33
1.5.2 Variables supplémentaires	36
1.6 Les variantes de l'analyse en composantes principales	43
1.6.1 ACP avec rotation	43
1.6.2 ACP des rangs	45
1.6.3 ACP sur les variables qualitatives	45
2 Analyse Factorielle des Correspondances (AFC)	47
2.1 Données et notations	47
2.1.1 Du tableau de contingence au tableau de probabilités	50
2.1.2 Indépendance entre variables qualitatives	52
2.1.3 Comment l'AFC appréhende l'écart à l'indépendance ?	54
2.2 Les nuages et leur ajustement	60
2.2.1 Nuage des profils lignes	60

2.2.2	Nuage des profils colonnes	63
2.2.3	Pourcentage d'inertie et inertie en AFC	65
2.2.4	Ajustement des nuages N_I des profils lignes et N_J des profils colonnes	69
2.2.5	Représentation simultanée des profils - Relation de transition	72
2.3	Aides à l'interprétation	74
2.3.1	Analyse du point de vue des lignes	74
2.3.2	Analyse du point de vue des colonnes	77
2.3.3	Quelques mesures dérivées du χ^2	78
2.4	Éléments supplémentaires	80
3	Analyse Factorielle des Correspondances Multiples (AFCM)	84
3.1	Données - Objectifs et problématique	84
3.1.1	Statistiques et graphiques	86
3.1.2	Du tableau de codage condensé au tableau disjonctif complet	88
3.1.3	Transformation du tableau disjonctif complet	92
3.1.4	Objectifs - Problématique	93
3.2	Mise en œuvre d'une ACM	101
3.2.1	ACM via une AFC sur le tableau disjonctif complet	102
3.2.2	ACM via une AFC sur le tableau de Burt	104
3.2.3	ACM via une ACP sur le tableau des profils	105
3.2.4	Inertie et pourcentage d'inertie en ACM	108
3.3	Représentation des nuages N_I et N_K	110
3.3.1	Nuage N_I des individus	110
3.3.2	Nuage N_K des modalités	112
3.3.3	Représentation simultanée	112
3.3.4	Relation de transition	114
3.3.5	Représentation simultanée barycentrique	115
3.4	Aide à l'interprétation des résultats	117
3.4.1	Analyse du point de vue des individus	117
3.4.2	Analyse du point de vue des modalités	119
3.4.3	Analyse du point de vue des variables	121
3.4.4	Éléments supplémentaires	123
3.5	Approche par machine learning	131
4	Classification et KMeans	134
4.1	La classification ascendante hiérarchique (CAH)	134
4.1.1	Mise en œuvre de la classification	135
4.1.2	Choix du nombre de classes	140
4.1.3	Interprétation des classes	143
4.2	La méthode de partitionnement : KMeans	147
4.2.1	Algorithme d'agrégation autour des centres mobiles	147
4.2.2	Mise en œuvre de la méthode des kmeans	148
4.2.3	Aide à la détection du nombre adéquat de groupes	151
5	Analyse Factorielle Multiple (AFM)	154
5.1	Données - objectifs	154
5.1.1	Exemple illustratif	154
5.1.2	Structure du jeu de données	156
5.1.3	Objectifs	157
5.2	Équilibre et ACP globale	157

5.2.1	Importance d'équilibrer l'influence de chaque groupe	157
5.2.2	L'AFM : une ACP pondérée	158
6	Analyse Factorielle des Données Mixtes (AFDM)	160
6.1	Données	161
6.1.1	Exemple illustratif	161
6.1.2	Statistiques sur les variables quantitatives	162
6.1.3	Statistiques sur les variables qualitatives	164
6.2	Représentation des nuages N_I et N_K	166
6.2.1	Représentation des individus dans N_K	166
6.2.2	Représentation des variables dans N_I	168
6.3	Mise en œuvre de l'AFDM	168
6.3.1	L'inertie maximale	168
6.3.2	AFDM sous forme d'ACP sur données transformées	169
6.3.3	Coordonnées des individus	171
6.3.4	Coordonnées des variables - modalités	171
6.3.5	Relations de transition	174
6.4	Aide à l'interprétation	176
6.4.1	Analyse du point de vue des individus	176
6.4.2	Analyse du point de vue des variables quantitatives	176
6.4.3	Analyse du point de vue des modalités	177
6.4.4	Analyse du point de vue des variables qualitatives	177
6.5	Éléments supplémentaires	179
6.5.1	Individus supplémentaires	179
6.5.2	Variables supplémentaires	182
7	Anova à 1 facteur	186
7.1	Exemple introductif	186
7.2	Le modèle d'analyse de la variance à 1 facteur	188
7.2.1	Notations	188
7.2.2	Le modèle et ses hypothèses	190
7.2.3	Les estimateurs du modèle	190
7.2.4	Équation de décomposition de la variance	191
7.2.5	Le critère et le test	192
8	Analyse Factorielle Discriminante (AFD)	195
8.1	Données - Problématique	195
8.1.1	Données	195
8.1.2	Analyse descriptives des variables	198
8.1.3	Problématique	200
8.2	Mise en œuvre de l'AFD	200
8.2.1	Recherche de variables discriminantes	201
8.2.2	Les différents types de corrélation en AFD	206
8.3	Évaluation globale du modèle	210
8.3.1	Distance entre centre classes	210
8.3.2	Pouvoir discriminant des facteurs	211
8.3.3	Calcul du Lambda de Wilks	212
8.3.4	Transformation de Bartlett	212
8.3.5	Transformation de RAO	213
8.3.6	Test sur ensemble de facteurs	214

8.3.7	Taux de bon classement et prédiction	215
8.3.8	Évaluation des contributions des variables	218
II	Régression linéaire	220
9	Le modèle linéaire de régression simple	221
9.1	Le modèle et ses hypothèses	221
9.1.1	Présentation du modèle	221
9.1.2	Les hypothèses du modèle	222
9.1.3	Estimation des paramètres par moindres carrés ordinaires (MCO)	222
9.1.4	Estimation par maximum de vraisemblance (MV)	227
9.2	Équation d'analyse de la variance	228
9.2.1	Équation de décomposition de la variance	228
9.2.2	Le coefficient de détermination R^2	229
9.2.3	Tableau d'analyse de la variance	230
9.3	Prévision, tests de significativité des paramètres et intervalles de confiance	230
9.3.1	Prévision de la variable endogène	230
9.3.2	Tests de significativité	232
9.3.3	Intervalle de confiance pour β_i	234
9.3.4	Région de confiance simultanée de β_0 et β_1	234
9.3.5	Intervalle de confiance pour σ^2	235
10	Modèle linéaire de régression multiple	236
10.1	Spécification du modèle	236
10.1.1	Présentation du modèle	236
10.1.2	Le modèle sous forme matricielle	237
10.1.3	Hypothèses de base	237
10.2	Estimation et propriétés des estimateurs	239
10.2.1	Estimation des coefficients de régression par MCO	239
10.2.2	Estimation des coefficients de régression par MV	242
10.3	L'analyse de la variance	243
10.3.1	Équation d'analyse de la variance	244
10.3.2	Le coefficient de détermination multiple : R^2	244
10.3.3	Le coefficient de détermination ajustée ou corrigé : \bar{R}^2	244
10.3.4	Tableau d'analyse de la variance	245
10.4	Quelques tests et intervalles de confiance	245
10.4.1	Intervalle de confiance de la variance de l'erreur	245
10.4.2	Test de comparaison d'un ensemble de paramètres à un ensemble de valeur	245
10.4.3	Test de significativité globale du modèle	246
10.4.4	Prévision de la variable endogène	247
10.5	Influence et diagnostics	249
10.5.1	Résidus et PRESS	251
10.5.2	Analyse de la normalité	255
10.5.3	Analyse de l'homoscédasticité	256
10.5.4	Analyse de la structure des résidus	257
10.6	Autres mesures de diagnostiques	259
10.6.1	Effet levier et observations influentes	259
10.6.2	Distance de Cook	261
10.6.3	Sélection des variables	263

11 Le modèle linéaire général	266
11.1 Introduction	266
11.2 Modélisation du problème	268
11.2.1 Modélisation	268
11.2.2 Tests sur les variables explicatives	270
11.3 Analyse de la covariance	273
11.3.1 Position du problème	273
11.3.2 Hypothèse gaussienne	276
III Régression logistique	281
12 Introduction à la régression logistique	282
12.1 Retour sur le modèle linéaire gaussien	282
12.1.1 Rappels	282
12.1.2 Limites	283
12.2 Vers le modèle linéaire généralisé	284
12.2.1 Exemples de fonctions de liens	285
12.2.2 Données	287
13 Régression logistique binaire	290
13.1 Un cadre bayésien pour l'apprentissage supervisée	290
13.1.1 Problématique - mise en oeuvre	290
13.1.2 Limite de l'approche basée sur les fréquences	292
13.2 Analyse discriminante logistique	293
13.2.1 Modèle linéaire généralisé : GLM	293
13.2.2 Exemples de modèle GLM	294
13.3 L'estimateur du maximum de vraisemblance	295
13.3.1 Estimation des paramètres	295
13.3.2 Dimensions explicatives, variables explicatives	298
13.3.3 Interprétation des coefficients β	300
13.3.4 Coefficients standardisés en régression logistique	305
13.4 Précision des estimateurs	308
13.4.1 Loi asymptotique des estimateurs	308
13.4.2 Intervalle de confiance	308
13.4.3 Test de nullité de q coefficients libres	309
13.5 Première évaluation de la régression logistique : les pseudo- R^2	315
13.5.1 Estimation du paramètre β_0 et de la déviance du modèle trivial	315
13.5.2 Quelques pseudo- R^2	317
14 Sélection et validation des modèles	320
14.1 Sélection ou choix de modèle	320
14.1.1 La déviance	320
14.1.2 Test de déviance entre 2 modèles emboîtés	321
14.1.3 Critère de choix de modèle	322
14.1.4 Subdivision apprentissage - test	323
14.1.5 Sélection automatique	326
14.2 Validation du modèle	328
14.2.1 Analyse des résidus	328
14.2.2 Points leviers et points influents	330

14.2.3 Application	331
15 Évaluation de la régression logistique binaire	344
15.1 Matrice de confusion	344
15.1.1 Construction de la matrice de confusion	344
15.1.2 Les indicateurs associés à la matrice de confusion	347
15.1.3 Inconvénients de la matrice de confusion	352
15.1.4 Diagramme de fiabilité	352
15.2 Quelques tests statistiques	354
15.2.1 Test de Hosmer-Lemeshow	354
15.2.2 Test de Mann-Whitney	357
15.3 Quelques courbes d'évaluation	360
15.3.1 La courbe ROC	360
15.3.2 La courbe LIFT	363
15.3.3 La courbe rappel-prévision	365
16 Régression logistique multinomiale	368
16.1 Le modèle polytomique nominal	368
16.1.1 Modèle polytomique nominal	368
16.1.2 Données	369
16.1.3 Estimation et interprétations des paramètres	370
16.1.4 Odds et odds ratios	373
16.1.5 Quelques statistiques de test	375
16.2 Évaluation des classifiants	376
16.2.1 Matrice de confusion et taux d'erreur	376
16.2.2 Indicateurs synthétiques pour le rappel et la précision	379
16.3 Test sur les coefficients de la régression multinomiale	381
16.3.1 Estimation de la matrice de variance covariance	381
16.3.2 Significativité d'un coefficient dans un logit	385
16.3.3 Significativité d'un coefficient dans tous les logit	386
16.3.4 Test d'égalité d'un coefficient dans tous les logit	389
16.4 Modèle multinomial et régresseurs catégorielles	390
16.4.1 Problématique : Données et modèle	390
16.4.2 Application	391
17 Régression logistique polytomique ordinaire	399
17.1 Le modèle à catégories adjacentes	399
17.2 Le modèle à odds proportionnels	400
17.2.1 Écriture du modèle	400
17.2.2 Pourquoi « odds proportionnels » ?	400
17.2.3 Égalité des pentes	401
17.2.4 Test d'égalité des pentes	402
17.3 Exemple d'application	402
17.3.1 Données de portefeuille	402
17.3.2 Données d'études universitaires	407
17.4 Modèle Log-linéaire : Régression de Poisson	408
17.4.1 Présentation du modèle	408
17.4.2 Application aux données réelles	409
17.4.3 Sur-dispersion	418

Annexes	425
Bibliographie	426

Première partie

Statistique exploratoire

CHAPITRE 1

ANALYSE EN COMPOSANTES PRINCIPALES (ACP)

Sommaire

1.1	Données et problématique de l'étude	2
1.2	Mise en oeuvre de l'ACP	14
1.3	Représentation des individus et des variables	19
1.4	Outils pour l'aide à l'interprétation	28
1.5	Éléments supplémentaires	33
1.6	Les variantes de l'analyse en composantes principales	43

Introduite par [Hot33] suivant des idées avancées par Pearson dès 1901, l'Analyse en Composantes Principales (ACP) s'applique à des tableaux croisant des individus et des variables quantitatives, appelés de façon concise tableaux Individus \times Variables quantitatives. C'est une technique qui permet d'obtenir une carte des unités d'observations (individus, ménages, entreprises, etc...) en fonction de leur proximité et une carte des variables en fonction de leur corrélation au sein d'un tableau de mesures quantitatives. Elle cherche à repérer une structuration de la population par rapport à un ensemble de variables et les interactions entre ces variables.

L'objectif de l'Analyse en Composantes Principales est de : repérer les groupes d'individus « semblables » vis-à-vis de l'ensemble des variables; relever des différences entre individus ou groupes d'individus relativement à l'ensemble des variables; mettre en évidence les individus dont les comportements sont « atypiques » vis-à-vis de l'ensemble des variables; rechercher si l'information contenue dans le tableau brut ne pourrait pas être obtenue avec un nombre petit de variables, ces dernières pouvant être différentes des variables d'origine.

Nous pouvons distinguer trois variantes de l'analyse en composantes principales :

1. l'ACP générale

Ici, la recherche des directions de variance maximale est faite sans transformer les données. On travaille donc directement sur la matrice des données brutes. Cette variante est utilisée très rarement, essentiellement pour tenir compte du zéro naturel de certaines variables.

2. L'ACP centrée

Ici, les variables sont d'abord « centrées », c'est-à-dire la moyenne de chaque variable est soustraite pour chaque observation de la valeur de cette variable. Cette variante est utilisée lorsque les variables initiales sont directement comparables (de même nature, intervalles de variation comparables)

3. l'ACP normée

Les variables sont à la fois « centrées » et « réduites ». La réduction (après centrage) consiste à diviser pour chaque observation la valeur de cette variable par son écart-type. Chaque variable possède une moyenne nulle et un écart-type unitaire. Cette variante, la plus fréquemment rencontrée, est employée lorsque les variables quantitatives sont de nature différente (par exemple, distances, poids, durées, etc...) ou présentent des intervalles de variation très différentes.

1.1 Données et problématique de l'étude

1.1.1 Description des données

L'Analyse en Composantes Principales s'intéresse à des tableaux de données rectangulaires de mesures X avec en lignes des individus et en colonnes des variables qui sont de nature quantitative :

Individu \ Variable	1	...	k	...	K
			⋮		
i	x_i^k
			⋮		
			⋮		

TABLE 1.1 – Tableau de données en ACP

Ce tableau de mesures décrit I individus à l'aide de K variables. x_i^k est la valeur prise par l'individu i pour la variable k . On suppose généralement que $I > K$. Chaque individu étant décrit dans le tableau par une ligne de K chiffres, il peut donc être représenté par un point dans un espace à K dimensions (espace direct). De même, chaque variable étant traduite dans le tableau par une colonne de I chiffres, elle peut donc être représentée par un point dans un espace à I dimensions (espace dual).

Pour une variable k , on a : $x^k = (x_1^k, \dots, x_I^k)$ un vecteur de dimension I , donc $x^k \in \mathbb{R}^I$. Pour chaque individu i , on a : $x_i = (x_i^1, \dots, x_i^K)$, un vecteur de dimension K , donc $x_i \in \mathbb{R}^K$. Pour chaque variable k , on note $\bar{x}^k = \frac{1}{I} \sum_{i=1}^I x_i^k$ sa moyenne et $\sigma^k = \sqrt{\frac{1}{I} \sum_{i=1}^I (x_i^k - \bar{x}^k)^2}$ son écart type.

Des tableaux de données de ce type avec des individus en lignes et des variables en colonnes, on en trouve dans de très nombreux domaines d'application tels que l'économie, la biologie, le marketing, etc..

Pour illustrer ce chapitre, nous allons prendre un exemple sur les données météorologiques. Les données sur lesquelles nous allons travailler proviennent d'un jeu de données d'étudiants français qui avaient pris cela comme sujet d'examen. On retrouve ce jeu de données dans le livre Analyses factorielles simples et multiples : cours et études de cas de [EP98].

En lignes, les individus statistiques sont représentés par les 15 villes de France sélectionnées et en colonnes les températures mensuelles moyennes. Ces températures mensuelles moyennes ont été calculées sur 30 ans. Par exemple, à Bordeaux en Janvier, il fait en moyenne 5.6 degrés. Cette valeur de 5.6 degrés est la moyenne sur tous les jours de Janvier pendant 30 ans. On a ainsi 12 variables correspondantes au 12 mois de l'année. On retrouve également en colonnes deux variables dues à la position géographique des villes (Latitude et Longitude).

ville	jan	fev	mars	avril	mai	juin	juil	août	sept	oct	nov	dec	lati	long
Bordeaux	5.6	6.6	10.3	12.8	15.8	19.3	20.9	21.0	18.6	13.8	9.1	6.2	44,50	-0,34
Brest	6.1	5.8	7.8	9.2	11.6	14.4	15.6	16.0	14.7	12.0	9.0	7.0	48,24	-4,29
Clermont	2.6	3.7	7.5	10.3	13.8	17.3	19.4	19.1	16.2	11.2	6.6	3.6	45,47	3,05
Grenoble	1.5	3.2	7.7	10.6	14.5	17.8	20.1	19.5	16.7	11.4	6.5	2.3	45,10	5,43
Lille	2.3	2.9	6.0	8.9	12.4	15.3	17.1	17.1	14.7	10.4	6.1	3.5	50,38	3,04
Lyon	2.1	3.3	7.7	10.9	14.9	18.5	20.7	20.1	16.9	11.4	6.7	3.1	45,45	4,51
Marseille	5.5	6.6	10.0	13.0	16.8	20.8	23.3	22.8	19.9	15.0	10.2	6.9	43,18	5,24
Montpellier	5.6	6.7	9.9	12.8	16.2	20.1	22.7	22.3	19.3	14.6	10.0	6.5	43,36	3,53
Nantes	5.0	5.3	8.4	10.8	13.9	17.2	18.8	18.6	16.4	12.2	8.2	5.5	47,13	-1,33
Nice	7.5	8.5	10.8	13.3	16.7	20.1	22.7	22.5	20.3	16.0	11.5	8.2	43,42	7,15
Paris	3.4	4.1	7.6	10.7	14.3	17.5	19.1	18.7	16.0	11.4	7.1	4.3	48,52	2,20
Rennes	4.8	5.3	7.9	10.1	13.1	16.2	17.9	17.8	15.7	11.6	7.8	5.4	48,05	-1,41
Strasbourg	0.4	1.5	5.6	9.8	14.0	17.2	19.0	18.3	15.1	9.5	4.9	1.3	48,35	7,45
Toulouse	4.7	5.6	9.2	11.6	14.9	18.7	20.9	20.9	18.3	13.3	8.6	5.5	43,36	1,26
Vichy	2.4	3.4	7.1	9.9	13.6	17.1	19.3	18.8	16.0	11.0	6.6	3.4	46,08	3,26

TABLE 1.2 – Données de températures

Dans un premier temps, nous importons le tableau des individus (les villes) et variables actives (les mois : Jan – Dec) pour la construction des axes factoriels. Les variables liées à la position géographique seront utilisées comme variables supplémentaires. Pour importer notre base, nous utilisons la fonction `.read_excel()` de la librairie `pandas` de Python.

```
# chargement des données
import pandas as pd

donnee = pd.read_excel('temperature.xlsx', sheet_name=0, header=0, index_col=0)
```

`sheet_name=0` signifie tout simplement que les variables actives dont on a besoin pour l'analyse en composantes principales se trouvent dans la première feuille du classeur excel. `header=0` indique à la machine que la première ligne correspond aux noms des variables (les mois) tandis que `index_col=0` indique que la première colonne correspond aux identifiants des observations, à savoir les individus (les villes).

Grâce à l'attribut `.describe()`, on peut faire ressortir quelques statistiques descriptives (moyenne, écart type, maximum, minimum, etc...) sur notre jeu de données.

```
# Statistiques sur les variables
print(donnee.describe(include = "all").round(2))
```

stats	Jan	Fev	Mars	Avril	Mai	Juin	Juil	Août	Sept	Oct	Nov	Dec
count	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00
mean	3.97	4.83	8.23	10.98	14.43	17.83	19.83	19.57	16.99	12.32	7.93	4.85
std	2.01	1.87	1.53	1.41	1.50	1.79	2.13	2.01	1.85	1.83	1.80	1.96
min	0.40	1.50	5.60	8.90	11.60	14.40	15.60	16.00	14.70	9.50	4.90	1.30
25%	2.40	3.35	7.55	10.00	13.70	17.15	18.90	18.45	15.85	11.30	6.60	3.45
50%	4.70	5.30	7.80	10.70	14.30	17.50	19.40	19.10	16.40	11.60	7.80	5.40
75%	5.55	6.20	9.55	12.20	15.35	19.00	20.90	20.95	18.45	13.55	9.05	6.35
max	7.50	8.50	10.80	13.30	16.80	20.80	23.30	22.80	20.30	16.00	11.50	8.20

TABLE 1.3 – Statistiques descriptives sur les variables

On stocke dans deux variables `n` et `p` les dimensions de notre tableau.

```
# Dimension des données actives
n = donnee.shape[0] # nombre de lignes
p = donnee.shape[1] # nombre de colonnes
print(f'n = {n} et p = {p}')
```

`n = 15 et p = 12`

1.1.2 Problématique de l'ACP

On peut étudier le tableau 1.1 de deux façons différentes. On peut le considérer comme un ensemble de lignes et chercher les différences et les ressemblances qu'il peut y avoir d'une ligne à l'autre. On peut également considérer ce tableau comme un ensemble de colonnes et chercher à voir les ressemblances entre colonnes.

1) Notion de distance entre individus

Pour l'étude sur les individus statistiques (les lignes), on essaie d'évaluer leur **ressemblance**. nous devons définir quand 2 individus se ressemblent et quand est-ce qu'ils se ressemblent du point de vue de l'ensemble des colonnes. Si nous avons beaucoup d'individus, nous voudrons faire un bilan des ressemblances : ces individus se ressemblent et sont très proches du point de vue de l'ensemble des colonnes ; ces autres individus sont proches entre eux et différents du premier groupe d'individus. Nous cherchons ici à faire une **typologie des individus**, une partition des individus, c'est-à-dire à construire des groupes d'individus homogènes du point de vue de l'ensemble des variables : à l'intérieur d'un groupe, les individus se ressemblent et d'un groupe à l'autre, ils sont différents. En ACP, la distance $d(i, j)$ entre deux individus i et j est définie par

$$d^2(i, j) = \sum_{k=1}^K (x_i^k - x_j^k)^2 \quad (1.1)$$

Il s'agit ici de la distance euclidienne classique. Deux points sont très voisins si, dans l'ensemble, leurs K coordonnées sont très proches. Les deux individus concernés sont alors caractérisés par des valeurs presque égales pour chaque variable.

```
# Distance entre villes
import numpy as np

def distance(x,y):
```

```

    return np.sum((x-y)**2)

# Calcul de la distance
rowdist = pd.DataFrame(np.zeros(shape=(n,n),dtype=float),index = donnee.index,
                       columns = donnee.index)

for i in range(n):
    for j in range(i+1,n):
        rowdist.values[i,j] = distance(donnee.values[i,:],donnee.values[j,:])

# Affichage
print(rowdist.round(2))

```

Villes	Bordeaux	Brest	Clermont	Grenoble	Lille	Lyon	Marseille	Montpellier	Nantes	Nice	Paris	Rennes	Strasbourg	Toulouse	Vichy
Bordeaux	133.9	70.9	78.1	157.9	59.8	17.2	7.9	36.6	32.1	56.9	67.1	175.1	6.7	84.9	
Brest		75.5	115.1	53.6	121.4	231.3	191.4	40.3	241.3	61.9	21.9	142.3	108.3	74.7	
Clermont			5.	23.3	6.9	132.5	105.9	17.2	190.7	2.3	18.3	26.7	37.7	0.8	
Grenoble				38.7	2.5	132.1	108.4	34.3	200.8	11.1	40.2	22.1	43.6	6.	
Lille					51.9	253.2	214.	47.4	315.8	27.6	27.9	25.4	108.4	17.2	
Lyon						103.7	83.8	30.4	168.9	10.7	41.6	33.7	30.4	9.8	
Marseille							2.2	94.6	13.8	117.8	141.2	253.9	31.6	150.2	
Montpellier								69.6	16.3	92.	110.	221.2	18.5	122.	
Nantes									123.	8.4	4.9	82.1	19.4	21.8	
Nice										167.3	171.5	346.2	59.7	211.7	
Paris											11.1	39.2	29.8	4.3	
Rennes												72.4	42.4	20.	
Strasbourg													120.9	20.7	
Toulouse														47.6	
Vichy															

TABLE 1.4 – Distance entre villes à partir des données du tableau initial

Nous pouvons visualiser cette matrice rapidement à l'aide d'un [heatmap](#).

```

# Visualisation - heatmap
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize = (10,8))
sns.heatmap(rowdist,xticklabels=rowdist.columns,yticklabels = rowdist.columns,
            cmap = sns.color_palette("Blues",12),linewidths=0.5)
plt.title('Heatmap des distances entre villes')
plt.show()

```

A l'aide du heatmap (figure 1.1), on peut déceler certaines distances très élevées : Lille et Nice, Nice et Strasbourg ou encore Lille et Marseille.

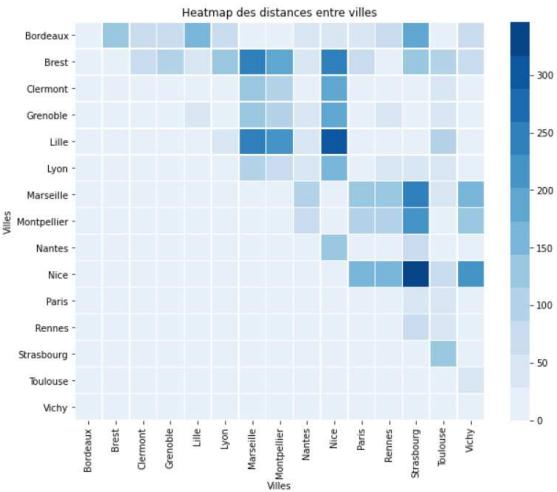


FIGURE 1.1 – Heatmap des distances entre les villes

2) Notion de liaison entre variables

Au niveau des variables, de façon symétrique, nous cherchons des ressemblances entre variables : quelles sont les variables qui apportent une information à peu près identiques. Quelles sont les variables qui apportent des informations différentes ? Entre variables, plutôt que de ressemblance, on parle souvent de **liaison** et les liaisons les plus connues sont les liaisons linéaires. Ce sont des liaisons simples, très fréquentes et finalement qui peuvent résumer de nombreuses liaisons, même quand une liaison est non linéaire entre deux variables, il est fréquent que sur certains domaines, la liaison puisse être proche d'une liaison linéaire. Une question revient : Comment mesurer la liaison entre deux variables ?. On peut la mesurer par un indicateur comme le **coefficent de corrélation linéaire** : deux variables qui ont un coefficient de corrélation très élevé, proche de +1 seront des variables qui apportent la même information. Notre objectif dans l'analyse sera de faire un bilan des ressemblances entre variables et de visualiser la matrice des corrélations. Mais aussi de trouver des indicateurs qui résument beaucoup de variables. Autrement dit, à partir des nombreuses variables du tableau, nous voulons avoir une vision de l'ensemble des liaisons sans passer en revue chaque coupe de variables. Cette vision peut se faire par l'intermédiaire de **variables synthétiques** appelées ici **composantes principales**. Des indicateurs synthétiques, il en existe a priori, par exemple la moyenne. Mais ici, on va plutôt chercher des indicateurs a posteriori, c'est-à-dire qu'on va utiliser l'information qui est contenue dans les données pour trouver un indicateur.

Matrice de corrélation linéaire

L'analyse des liaisons entre deux variables quantitatives s'effectue généralement à travers le coefficient de corrélation linaire de Pearson. C'est un indicateur statistique, noté généralement ρ , compris entre -1 et 1. Plus la valeur absolue du coefficient est importante, plus la relation linéaire entre les variables est forte. Ainsi, pour la corrélation linéaire de Pearson, une valeur absolue de 1 indique une relation linéaire parfaite. Une corrélation proche de 0 indique l'absence de relation linéaire entre les variables. Pour deux variables quantitatives x^k et x^l , on a :

$$\text{Corr}(k,l) = \rho(k,l) = \frac{\text{Cov}(x^k, x^l)}{\sigma_{x^k} \sigma_{x^l}} = \frac{\sum_{i=1}^I (x_i^k - \bar{x}^k)(x_i^l - \bar{x}^l)}{\sqrt{\sum_{i=1}^I (x_i^k - \bar{x}^k)^2 \sum_{i=1}^I (x_i^l - \bar{x}^l)^2}} \quad (1.2)$$

Dans le cas d'espèce, le coefficients de corrélation de Pearson permet de savoir quelles sont les villes les plus corrélées et celles les moins corrélées. La librairie Pandas dispose d'une méthode `.corr()` qui permet de calculer la corrélation entre variables.

```
# Matrice de corrélation linéaire entre paires de variables
corr = donnee.corr(method = "pearson")
# Affichage
print(corr.round(2))
```

	Jan	Fev	Mars	Avril	Mai	Juin	Juil	Août	Sept	Oct	Nov	Dec
Jan	1.00											
Fev	0.97	1.00										
Mars	0.84	0.93	1.00									
Avril	0.61	0.76	0.92	1.00								
Mai	0.36	0.55	0.77	0.95	1.00							
Juin	0.34	0.52	0.76	0.94	0.99	1.00						
Juil	0.30	0.49	0.72	0.91	0.98	0.99	1.00					
Août	0.41	0.59	0.80	0.95	0.98	0.99	0.99	1.00				
Sep	0.60	0.76	0.91	0.98	0.94	0.94	0.93	0.97	1.00			
Oct	0.85	0.94	0.97	0.91	0.77	0.76	0.74	0.81	0.93	1.00		
Nov	0.95	0.99	0.93	0.78	0.59	0.57	0.55	0.64	0.80	0.96	1.00	
Dec	0.99	0.97	0.83	0.62	0.38	0.36	0.32	0.43	0.62	0.87	0.96	1.00

TABLE 1.5 – Matrice des corrélations des variables

On construit un heatmap afin de visualiser cette matrice des corrélations.

```
# heatmap de la matrice des corrélation
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize = (10,8))
sns.heatmap(corr,xticklabels=corr.columns,yticklabels = corr.columns,
            vmin = -1, vmax = +1, center = 0,
            cmap = 'RdBu', linewidths=0.5)
plt.title('Heatmap des corrélations croisées entre variables')
plt.show()
```

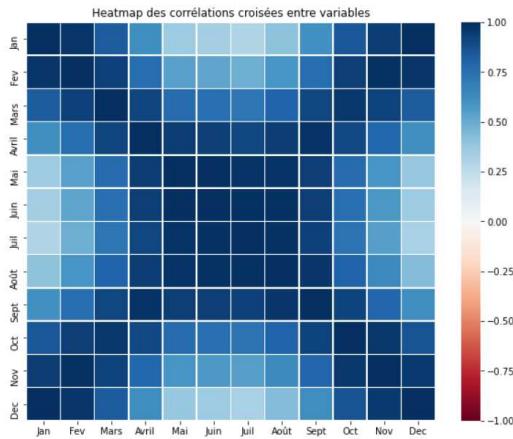


FIGURE 1.2 – Heatmap des corrélations croisées entre variables

Nous notons que les plus fortes corrélations sont entre (Fev, Nov), (Jan, Dec), (Mai, Juin), etc.. Puisque c'est le même tableau qui est vu en lignes, puis en colonnes, il y a donc un lien entre ces 2 études, entre l'étude sur les individus et l'étude sur les variables. Dans l'étude sur les individus, nous construisons des groupes d'individus et nous cherchons à caractériser les différents groupes d'individus (ou classes d'individus). Pour faire ce travail, plutôt que de lister tous les individus du groupe, nous préférons utiliser les variables. Pour cela, on a besoin d'une procédure automatique, surtout si on dispose de beaucoup de variables.

De même, quand nous étudions les liaisons entre variables, nous dirons que ces différentes variables sont très liées entre elles, mais ce langage est plutôt abstrait. On pourrait l'illustrer par une opposition entre des individus spécifiques, c'est-à-dire des individus qui sont très particuliers, qui sont très extrêmes. Par exemple, la variable taille et la variable poids sont deux variables très liées. On peut dire qu'il existe une corrélation linéaire forte entre ces deux variables. Mais on peut aussi illustrer cette liaison en opposant deux individus extrêmes : en disant par exemple, les individus qui sont petits sont légers et les individus qui sont grands lourds. L'illustration de la liaison par des individus extrêmes n'est ici par décisive, mais si nous disposons de beaucoup de variables et celles-ci sont moins bien connues, l'illustration par des individus extrêmes peut être utile.

Pour résumer, l'ACP est une méthode de statistique exploratoire (statistique descriptive multidimensionnelle). Cette méthode va synthétiser, résumer, hiérarchiser l'information contenue dans un tableau de données individus \times variables quantitatives. Et pour ce faire, elle va fournir une visualisation du tableau de données par des graphiques simples.

3) Comment considérer le tableau de données ?

Si nous considérons le tableau 1.1 comme un ensemble de lignes, nous sommes en train d'étudier les individus. Étudier les individus va revenir à considérer un nuage de point. Un point correspondant à un individu. Ce nuage de point évolue dans un espace de dimension élevée. Si le tableau contient K variables, le nuage de point vit dans un espace à K dimensions.

De façon symétrique, si nous considérons le tableau comme un ensemble de colonnes, nous sommes en train d'étudier les variables. Une variable est un point dans l'espace à I dimensions.

Chaque variable a I coordonnées et correspond donc à un point dans un espace à I dimensions. L'ensemble de ces points variables forme un nuage de variables qui vit dans un espace à I dimensions.

4) Problématique des données météorologiques

Le but général de l'étude est de comparer les températures mensuelles des différentes villes.

i) Analyse du point de vue des lignes (des villes)

Chaque ville étant caractérisée par ses 12 températures moyennes mensuelles. Quelles sont les villes qui se ressemblent vis-à-vis de l'ensemble des variables (les mois). Quelles sont celles qui diffèrent. Plus généralement, peut-on faire une typologie des villes mettant en évidence l'ensemble des ressemblances ainsi définies. Cette typologie faite, on peut se demander si ces ressemblances (ou dissemblances) correspondent à des proximités (ou des éloignements) géographiques. Ainsi, l'étude des individus revient donc analyser leur variabilité.

ii) Analyse du point de vue des colonnes (les mois)

Chaque mois est vu au travers des températures moyennes mensuelles des 15 villes. Le problème n'est pas de séparer les mois chauds des mois froids pour l'ensemble des 15 villes mais de comparer la répartition de ces villes (des plus chaudes au plus froides) pour deux mois différents sans tenir compte du fait que d'un mois à l'autre les températures sont globalement plus ou moins élevées. Les comparaisons entre mois se font au travers de la notion de liaison, plus précisément de corrélation. Deux mois sont d'autant plus corrélés que, pour chacun, on observe la même répartition des 15 villes selon leur température. A l'inverse, ils sont peu corrélés si ce ne sont pas dans les mêmes villes que l'on trouve les températures les plus élevées (ou les plus basses).

Cela posé, les questions sont les suivantes : Quels mois sont corrélés entre eux ? Quels sont ceux qui le sont peu ? Plus généralement, peut-on faire un bilan de corrélation entre les 12 mois ? Les températures mensuelles sont-elles liées à la position géographique (variables supplémentaires) ?

1.1.3 Transformation des données

Toutefois, la définition des proximités entre variables dans leurs espaces est assez fruste. Des problèmes d'échelle de mesure se posent d'emblée. Comment calculer la distance entre deux variables si l'une est exprimée en centimètre et l'autre en kilogramme. Comment intégrer un éloignement moyen dans \mathbb{R}^K ? Est-ce que deux individus assez proches dans \mathbb{R}^K ont des valeurs assez voisines pour chacune des variables, ou au contraire très proches pour certaines et éloignées pour d'autres ?

En ACP, le tableau des données est toujours centré (en pratique, le centrage des données est inclus dans les programmes d'ACP). A chaque valeur numérique, on soustrait la moyenne de la variable en cause. Le tableau obtenu est alors de terme général :

$$y_i^k = x_i^k - \bar{x}^k \quad (1.3)$$

```
# Centrage des données
def centrage(x):
    # x est un vecteur
    return (x-x.mean())
```

```

# Application
Y = donnee.transform(centrage)
# Affichage
print(Y.round(2))

```

ville	jan	fev	mars	avril	mai	juin	juil	août	sept	oct	nov	dec
Bordeaux	1.63	1.77	2.07	1.82	1.37	1.47	1.07	1.43	1.61	1.48	1.17	1.35
Brest	2.13	0.97	-0.43	-1.78	-2.83	-3.43	-4.23	-3.57	-2.29	-0.32	1.07	2.15
Clermont	-1.37	-1.13	-0.73	-0.68	-0.63	-0.53	-0.43	-0.47	-0.79	-1.12	-1.33	-1.25
Grenoble	-2.47	-1.63	-0.53	-0.38	0.07	-0.03	0.27	-0.07	-0.29	-0.92	-1.43	-2.55
Lille	-1.57	-1.93	-2.23	-2.08	-2.03	-2.53	-2.73	-2.47	-2.29	-1.92	-1.83	-1.35
Lyon	-1.87	-1.53	-0.53	-0.08	0.47	0.67	0.87	0.53	-0.09	-0.92	-1.23	-1.75
Marseille	1.53	1.77	1.77	2.02	2.37	2.97	3.47	3.23	2.91	2.68	2.27	2.05
Montpellier	1.63	1.87	1.67	1.82	1.77	2.27	2.87	2.73	2.31	2.28	2.07	1.65
Nantes	1.03	0.47	0.17	-0.18	-0.53	-0.63	-1.03	-0.97	-0.59	-0.12	0.27	0.65
Nice	3.53	3.67	2.57	2.32	2.27	2.27	2.87	2.93	3.31	3.68	3.57	3.35
Paris	-0.57	-0.73	-0.63	-0.28	-0.13	-0.33	-0.73	-0.87	-0.99	-0.92	-0.83	-0.55
Rennes	0.83	0.47	-0.33	-0.88	-1.33	-1.63	-1.93	-1.77	-1.29	-0.72	-0.13	0.55
Strasbourg	-3.57	-3.33	-2.63	-1.18	-0.43	-0.63	-0.83	-1.27	-1.89	-2.82	-3.03	-3.55
Toulouse	0.73	0.77	0.97	0.62	0.47	0.87	1.07	1.33	1.31	0.98	0.67	0.65
Vichy	-1.57	-1.43	-1.13	-1.08	-0.83	-0.73	-0.53	-0.77	-0.99	-1.32	-1.33	-1.45

TABLE 1.6 – Données centrées

En réalité, cette transformation n'a aucune incidence sur les définitions de la ressemblance entre individus et de la liaison entre variables. Toutefois, les résultats de l'ACP sur ce type de données sont alors très sensibles au choix des unités de mesure. Lorsque les unités de mesures ne sont pas les mêmes¹, on fait jouer à chaque variable un rôle identique dans la définition des proximités entre individus : on parle alors d'**analyse en composantes principales normée**. Pour cela, on corrige les échelles en adoptant la transformation suivante :

$$z_i^k = \frac{x_i^k - \bar{x}^k}{\sigma^k} \quad (1.4)$$

où \bar{x}^k et σ^k sont respectivement la moyenne et l'écart-type de la variable k . En effectuant cette transformation, toutes les variables présentent la même variabilité et de ce fait la même influence dans le calcul des distances entre individus².

```

# Centrage et réduction des données
def StandardScaler(x):
    # x est vecteur
    return (x - x.mean())/x.std(ddof=0)

# Application
Z = donnee.transform(StandardScaler)
#Affichage
print(Z.round(2))

```

1. Dans les études où toutes les variables s'expriment dans la même unité, on peut souhaiter ne pas réduire les variables.

2. Sklearn dispose d'une fonction `StandardScaler` pour centrer et réduire les données.

ville	jan	fev	mars	avril	mai	juin	juil	août	sept	oct	nov	dec
Bordeaux	0.84	0.98	1.4	1.33	0.94	0.85	0.52	0.74	0.90	0.84	0.67	0.72
Brest	1.10	0.54	-0.29	-1.3	-1.95	-1.98	-2.06	-1.83	-1.28	-0.18	0.62	1.14
Clermont	-0.71	-0.63	-0.5	-0.50	-0.44	-0.31	-0.21	-0.24	-0.44	-0.63	-0.76	-0.66
Grenoble	-1.28	-0.9	-0.36	-0.28	0.05	-0.02	0.13	-0.03	-0.16	-0.52	-0.82	-1.35
Lille	-0.81	-1.07	-1.51	-1.52	-1.4	-1.46	-1.33	-1.27	-1.28	-1.09	-1.05	-0.71
Lyon	-0.97	-0.85	-0.36	-0.06	0.32	0.38	0.42	0.27	-0.05	-0.52	-0.7	-0.92
Marseille	0.79	0.98	1.20	1.48	1.63	1.71	1.69	1.66	1.63	1.52	1.30	1.09
Montpellier	0.84	1.03	1.13	1.33	1.22	1.31	1.39	1.41	1.30	1.29	1.19	0.87
Nantes	0.53	0.26	0.11	-0.13	-0.37	-0.37	-0.50	-0.50	-0.33	-0.07	0.16	0.35
Nice	1.82	2.03	1.74	1.70	1.56	1.31	1.39	1.51	1.86	2.08	2.05	1.77
Paris	-0.30	-0.41	-0.43	-0.2	-0.09	-0.19	-0.36	-0.45	-0.55	-0.52	-0.47	-0.29
Rennes	0.43	0.26	-0.23	-0.64	-0.92	-0.94	-0.94	-0.91	-0.72	-0.41	-0.07	0.29
Strasbourg	-1.84	-1.85	-1.78	-0.86	-0.3	-0.37	-0.41	-0.65	-1.06	-1.6	-1.74	-1.87
Toulouse	0.37	0.42	0.65	0.45	0.32	0.5	0.52	0.69	0.74	0.55	0.39	0.35
Vichy	-0.81	-0.79	-0.77	-0.79	-0.57	-0.42	-0.26	-0.39	-0.55	-0.75	-0.76	-0.76

TABLE 1.7 – Données centrées et réduites

Ce tableau de données centrées et réduites (Table 1.7) donne déjà quelques informations : on voit par exemple qu'à Brest, en Juillet on a une valeur de -2.06. Cela signifie que les températures à Brest sont assez extrêmes en Juillet. La valeur -2 signifie que ce sont des valeurs inférieures à la moyenne et assez extrêmes. On sait que si les valeurs d'une variable suivent une loi normale, alors 95% des valeurs centrées-réduites sont comprises entre -1.96 et 1.96. Ici, on ne sait pas si les données suivent une loi normale, mais une valeur centrée-réduite de -2 est très extrême. Donc à Brest, il fait particulièrement froid en Juillet. A Nice, en revanche, il fait particulièrement chaud surtout en Février et Octobre-Novembre. Les températures sont supérieures à la moyenne et particulièrement grandes.

L'ACP va permettre d'analyser ce tableau centrée-réduit. Analyser ce tableau signifie le visualiser.

Notion de distance euclidienne pondérée

La distance euclidienne pondérée entre deux individus i et j est donnée par :

$$d^2(i,j) = \sum_{k=1}^K \frac{1}{(\sigma^k)^2} (x_i^k - x_j^k)^2 \quad (1.5)$$

Cette formule revient à calculer la distance entre deux individus dans le tableau des données centrées et réduites car :

$$d^2(i,j) = \sum_{k=1}^K \frac{1}{(\sigma^k)^2} (x_i^k - x_j^k)^2 = \sum_{k=1}^K \left[\left(\frac{x_i^k - \bar{x}^k}{\sigma^k} \right) - \left(\frac{x_j^k - \bar{x}^k}{\sigma^k} \right) \right]^2 = \sum_{k=1}^K (z_i^k - z_j^k)^2$$

```
# Calcul de la distance euclidienne pondérée
dist = pd.DataFrame(np.zeros(shape=(n,n), dtype=float), index = donnee.index,
                     columns = donnee.index)
for i in range(n):
    for j in range(i+1,n):
        dist.values[i,j] = distance(Z.values[i,:],Z.values[j,:])
# Visualisation - heatmap
```

```

plt.figure(figsize = (10,8))
sns.heatmap(dist,xticklabels=dist.columns,yticklabels = dist.columns,
            cmap = sns.color_palette("Blues",12),linewidths=0.5)
plt.title('Heatmap des distances entre villes')
plt.show()

```

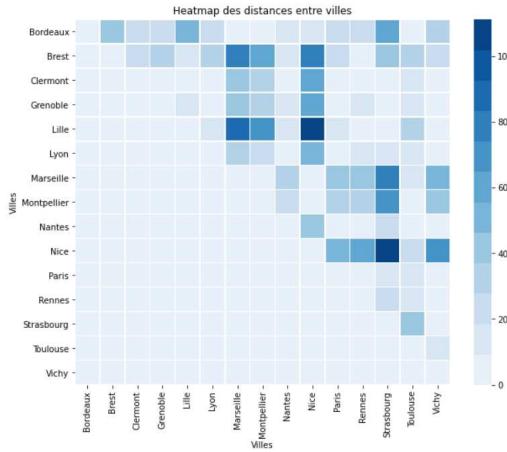


FIGURE 1.3 – Distance euclidienne pondérée

Distance à l'origine

La distance (au carré) d'un individu i à l'origine du nuage se calcule comme suivant :

$$d^2(i, G_I) = \sum_{k=1}^J z_{ik} \quad (1.6)$$

Rappelons qu'en Analyse en Composantes Principales, chaque individu à un poids $1/I$.

```

# Information sur les individus
rowdisto = Z.apply(lambda x : np.sum((x)**2),axis=1)
rowweight = np.ones(n)/n
rowinertie = rowweight*rowdisto.values
rowinfos = pd.DataFrame(np.transpose([rowdisto,rowweight, rowinertie]),
                        columns = ["Distro2", "Poids", "Inertie"],
                        index = donnee.index)
print(rowinfos.round(3))

```

ville	Disto2	Poids	Inertie
Bordeaux	10.287	0.067	0.686
Brest	21.948	0.067	1.463
Clermont	3.386	0.067	0.226
Grenoble	5.452	0.067	0.363
Lille	18.303	0.067	1.220
Lyon	3.913	0.067	0.261
Marseille	24.222	0.067	1.615
Montpellier	17.449	0.067	1.163
Nantes	1.402	0.067	0.093
Nice	36.819	0.067	2.455
Paris	1.734	0.067	0.116
Rennes	4.930	0.067	0.329
Strasbourg	21.731	0.067	1.449
Toulouse	3.164	0.067	0.211
Vichy	5.259	0.067	0.351

TABLE 1.8 – Résumé des informations sur les individus

1.1.4 La notion d'inertie

L'inertie représente la quantité d'information contenue dans un jeu de données. Elle indique la dispersion totale des données. Deux approches permettent de calculer l'inertie totale d'un tableau de données.

Une première approche pour le calcul de l'inertie est de la définir comme la moyenne des carrés des distances entre paires d'observations. Soit :

$$In_K = \frac{1}{I^2} \sum_{i,j=1}^I d^2(i,j) \quad (1.7)$$

```
# Inertie : moyenne des carrés des distances
InK = (1/n**2)*rowdist.sum().sum()
print('Inertie totale : %.2f' % (InK))
```

Inertie totale : 37.12

Une autre manière de considérer l'inertie est de la voir comme une généralisation multidimensionnelle de la variance. Elle exprime alors la dispersion autour du barycentre G du nuage de points, défini par le vecteur composé de moyennes des K variables $\bar{x} = (\bar{x}^1, \dots, \bar{x}^K)$:

$$In_K = \frac{1}{I} \sum_{i=1}^I d^2(i,G) \quad (1.8)$$

```
# Moyennes des variables
meanvar = donnee.mean(axis=0)

# Inertie totale
distorig = donnee.apply(lambda x: np.sum((x-meanvar)**2), axis = 1)
InK2 = (1/n)*distorig.sum()
print('Inertie totale : %.2f' % (InK2))
```

Inertie totale : 37.12

L'approche 2 a pour avantage de mettre en lumière un résultat important : **l'inertie est égale à la somme des variances des variables**. En effet :

$$In_K = \frac{1}{I} \sum_{i=1}^I d^2(i, G) = \frac{1}{I} \sum_{i=1}^I \sum_{k=1}^K (x_i^k - \bar{x}^k)^2 = \sum_{k=1}^K \frac{1}{I} \sum_{i=1}^I (x_i^k - \bar{x}^k)^2 = \sum_{k=1}^K \sigma_k^2$$

Vérification - variance des variables

```
variancevar = donnee.var(ddof = 0)
```

Inertie totale

```
InK3 = variancevar.sum()
```

```
print('Inertie totale : %.2f' % (InK3))
```

Inertie totale : 37.12

Lorsque les variables sont centrées et réduites, leur variance vaut 1. Par conséquent, l'inertie totale est alors égale au nombre de variables, soit K .

Inertie sur données centrées et réduites

```
print('Inertie totale (donnée centrée réduite) : %.1f' % (Z.var(ddof=0).sum()))
```

Inertie totale (donnée centrée réduite) : 12.0

1.2 Mise en oeuvre de l'ACP

Deux procédures permettent de mettre en oeuvre une Analyse en Composantes Principales : la diagonalisation de la matrice des corrélations et la décomposition en valeurs singulières de la matrice des données centrées et réduites.

1.2.1 Diagonalisation de la matrice des corrélations

L'analyse du nuage des points-individus dans \mathbb{R}^K nous a amené à effectuer une translation de l'origine au centre de gravité de ce nuage et à changer, dans le cas l'analyse normée, les échelles sur les différents axes. L'analyse du tableau transformé Z nous conduit à diagonaliser la matrice $C = Z'Z$. Le terme général c_{jk} de cette matrice s'écrit :

$$c_{jk} = \frac{1}{I} \sum_{i=1}^I z_{ij} z_{ik} \quad (1.9)$$

soit :

$$c_{jk} = \frac{1}{I} \sum_{i=1}^I \left(\frac{x_{ij} - \bar{x}_j}{\sigma_j} \right) \left(\frac{x_{ik} - \bar{x}_k}{\sigma_k} \right) \quad (1.10)$$

c'est-à-dire :

$$c_{jk} = \text{Corr}(j, k) \quad (1.11)$$

c_{jk} n'est autre que le coefficient de corrélation empirique entre les variables j et k . La matrice à diagonaliser est donc la *matrice de corrélations*. Le calcul est fait uniquement dans l'espace des

variables mais les résultats sont disponibles pour les deux points de vue (individus et variables). Elle consiste à mettre la matrice C sous la forme :

$$C = PDP^{-1} \quad (1.12)$$

où P est la matrice de passage (ou matrice des vecteurs propres). Puisque la matrice des corrélations est symétrique, l'équation précédente devient :

$$C = PDP^t \quad (1.13)$$

Principe de l'ACP

On cherche à estimer le vecteur $u = (u_1, \dots, u_K)$ telle que la somme des carrés des corrélations des variables avec le facteur soit maximale. Nous pouvons donc le programme d'optimisation suivant :

$$\begin{cases} \max_u u^T C u \\ \text{s.c : } \|u\| = u^T u = 1 \end{cases}$$

Nous avons un problème d'optimisation sous contraintes d'égalités. Nous passons par le Lagrangien pour le résoudre. Notre Lagrangien s'écrit :

$$L(u, \lambda) = u^T C u - \lambda(u^T u - 1) \quad (1.14)$$

Les dérivées partielles de cette fonction par rapport à u et λ nous donnent les résultats importants suivants :

$$\begin{cases} \frac{\partial}{\partial u} L(u, \lambda) = 0 \\ \frac{\partial}{\partial \lambda} L(u, \lambda) = 0 \end{cases} \rightarrow \begin{cases} Cu = \lambda u \\ u^T u = 1 \end{cases}$$

La première équation du système nous permet de voir que le vecteur solution u est le premier vecteur propre de la matrice des corrélations C : λ est la valeur propre associée.

La fonction `.linalg.eig` de Numpy nous aide dans le calcul de la diagonalisation de la matrice des corrélations. Cette fonction retourne 2 éléments : le premier élément est le vecteur des valeurs propres et le second est la matrice des valeurs propres.

```
# Diagonalisation de la matrice des corrélations
eigenvalue, eigenvector = np.linalg.eig(corr)

# eigenvalue dataframe
percent = np.array([100*x/sum(eigenvalue) for x in eigenvalue])
cumpercent = np.cumsum(percent)
columns = ['valeur propre', 'pourcentage d\'inertie',
           'pourcentage d\'inertie cumulée']
index = ['Dim.{}'.format(x+1) for x in range(p)]
Eigen = pd.DataFrame(np.transpose([eigenvalue, percent, cumpercent]),
                      index=index, columns=columns)
Eigen.index.name = 'Dimension'
# Affichage
print(Eigen.round(2))
```

Dimension	valeur propre	pourcentage d'inertie	pourcentage d'inertie cumulée
Dim.1	9.58	79.85	79.85
Dim.2	2.28	18.97	98.82
Dim.3	0.07	0.58	99.40
Dim.4	0.04	0.33	99.73
Dim.5	0.01	0.12	99.85
Dim.6	0.01	0.07	99.92
Dim.7	0.01	0.05	99.97
Dim.8	0.00	0.01	99.98
Dim.9	0.00	0.01	99.99
Dim.10	0.00	0.00	100.00
Dim.11	0.00	0.00	100.00
Dim.12	0.00	0.00	100.00

TABLE 1.9 – Valeurs propres

1.2.2 Décomposition en valeurs singulières

L'approche alternative pour l'ACP est de passer par une décomposition en valeurs singulières de la matrice des données centrées et réduites. Elle présente un double avantage par rapport à la précédente : elle ne nécessite pas le calcul de calcul de la matrice des corrélations ; elle montre bien le caractère dual de l'analyse.

La décomposition en valeurs singulières (*Singular Value Decomposition*) d'une matrice Z de taille $I \times K$ consiste à claucler les valeurs singulières $(\delta_1, \dots, \delta_K)$ (triées par ordre décroissante) et les matrices unitaires $U = (u_1, \dots, u_I)$ et $V = (v_1, \dots, v_K)$ tels que :

$$Z = \sum_{\alpha=1}^K \delta_\alpha u_\alpha v_\alpha^T = U \Delta V^t \quad (1.15)$$

avec la relation suivante :

$$\begin{cases} Z v_\alpha = \delta_\alpha u_\alpha \\ Z^T u_\alpha = \delta_\alpha v_\alpha \end{cases}$$

Les colonnes v_α de V sont les axes principaux et les vecteurs $\delta_\alpha u_\alpha$ sont les composantes principales. La matrice V correspond aux vecteurs propres associées aux valeurs propres δ_α . Nous utilisons la fonction `.linalg.svd` pour effectuer cette décomposition. Cette fonction renvoie les valeurs singulières et les matrices unitaires U et V .

```
# Décomposition en valeurs singulières
U, delta ,V = np.linalg.svd(Z)
```

δ représente les valeurs singulières. Pour obtenir les vraies valeurs propres (inerties) du nuage, on applique une correction aux δ obtenues par décomposition :

$$\lambda_\alpha = \frac{\delta_\alpha^2}{n}, \quad \forall \alpha = 1, \dots, K \quad (1.16)$$

```
# Correction des delta
lambd = delta**2/n
# Affichage de delta et lambda
```

```
print(pd.DataFrame(np.array([delta,lambda]),columns = index,
                   index=['delta','lambda']))
```

	Dim.1	Dim.2	Dim.3	Dim.4	Dim.5	Dim.6	Dim.7	Dim.8	Dim.9	Dim.10	Dim.11	Dim.12
delta	11.99	5.84	1.02	0.77	0.46	0.35	0.30	0.16	0.15	0.09	0.07	0.01
lambda	9.58	2.28	0.07	0.04	0.01	0.01	0.01	0.00	0.00	0.00	0.00	0.00

TABLE 1.10 – Delta et lambda

Notons que la fonction `PCA` de `Sklearn` se base sur une décomposition en valeurs singulières dans son calcul et nous renvoie les δ grâce à son attribut `.singular_values_`. Il en est de même pour la fonction `PCA` du package `fanalysis` créée par Olivier Garcia³.

Nous créons une fonction de visualisation des valeurs propres.

```
# Visualisation des valeurs propres
def screeplot(data,choice=None,figsize=None):
    #
    p = data.shape[0]
    fig,axes = plt.subplots(figsize = figsize); axes.grid()
    axes.set_xlabel('Dimensions',fontsize=14)
    axes.set_title('Scree plot',fontsize=14)
    axes.set_xticks([x for x in range(1,p+1)])
    if choice is None or choice=='scree plot':
        eigen = data.iloc[:,0].round(2)
        ylim = np.max(eigen)+1
        axes.set_xlim(0,ylim)
        axes.bar(np.arange(1,p+1),eigen.values,width=0.9)
        axes.plot(np.arange(1,p+1),eigen.values,c="black")
        axes.set_ylabel('Eigenvalue',fontsize=13)
        ## Add text
        for i in range(p):
            axes.scatter(i+1,eigen.values[i],color='black',alpha=1)
            axes.text(i+.75,0.10+eigen.values[i],str(eigen.values[i]),
                      color = 'black')
    elif choice == "percentage":
        percent = data.iloc[:,1].round()
        axes.set_xlim(0,100)
        axes.bar(np.arange(1,p+1),percent.values,width=0.9)
        axes.plot(np.arange(1,p+1),percent.values,c="black")
        axes.set_ylabel('Percentage of variance',fontsize=13)
        ## Add text
        for i in range(p):
            axes.scatter(i+1,percent.values[i],color='black',alpha=1)
            axes.text(i+.6,0.10+percent.values[i],f'{percent.values[i]}%',color = 'black',fontweight='bold',fontsize=12)
    elif choice == "cumulative":
        cumul = data.iloc[:,2].round()
```

3. Olivier Garcia est un ancien étudiant du Master SISE de l'Université de Lyon-2 (o.garcia.dev@gmail.com).

```

    axes.set_ylim(0,105)
    axes.bar(np.arange(1,p+1),cumul.values,width=0.9)
    axes.set_ylabel('Cumulative percentage of variance',fontsize=13)
    plt.show()
# Affichage
screeplot(data=Eigen,figsize=(8,6))

```

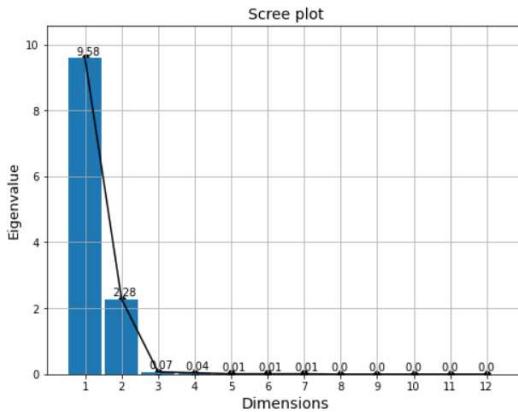


FIGURE 1.4 – Scree plot

1.2.3 Choix du nombre d'axes

Une question qui se pose à la suite d'une ACP est celle de savoir : Combien d'axes interpréter ? En effet, le nombre d'axes factoriels à la suite d'une ACP est égal au nombre de variables actives et chacun de ces axes restitue une partie de l'information contenue dans le nuage. Puisque l'ACP cherche à résumer l'information, alors il est important d'analyser les axes qui sont les plus riches en information.

Pour [Kai91] les premiers axes factoriels fournissent en projection un étalement maximal du nuage. Lorsque les variables sont centrées et réduites, il est justifié de retenir les axes correspondants à des valeurs propres supérieures à 1. En effet, les composantes principales étant des combinaisons linéaires des variables centrées réduites et de variance égale à la valeur propre associée, seules présentent un intérêt pour l'analyse des composantes de variance supérieure à celle des variables initiales (qui vaut 1!).

Rappelons que dans une Analyse en Composantes Principales normée, l'inertie totale de chacun des nuages (celui des vielles et celui des mois) est égale au nombre de variables actives, soit 12 ici. Avec une inertie de 9.58, qui représente 80% de l'inertie des nuages dans l'espace tout entier, le premier facteur est largement prépondérant. L'inertie du deuxième facteur vaut 2.28 et celle du troisième 0.07 ; les deux premiers facteurs totalisent 98.8% de l'inertie totale. Les deux nuages de points (individus et variables) sont donc pratiquement bidimensionnels : leur projection sur le premier plan factoriel en donne une représentation quasiment parfaite. On se limitera dans l'interprétation à l'étude de ces deux premiers facteurs et du plan qu'ils engendrent.

1.3 Représentation des individus et des variables

1.3.1 Nuage des individus

Afin d'avoir une représentation du nuage des individus, il faut au préalable calculer leurs coordonnées factorielles. Pour calculer les coordonnées factorielles des individus, il suffit d'effectuer le produit de la matrice des vecteurs propres (calculés sous décomposition spectrale) et la matrice des données centrées et réduite. Les coordonnées des I points-individus sur l'axe factoriel u_α ($\alpha^{\text{ième}}$ vecteur propre de la matrice \mathbf{C} associé à la valeur propre λ_α) sont les I composantes du vecteur :

$$F_\alpha = Z u_\alpha \quad (1.17)$$

Le facteur F_α est une combinaison linéaire des variables initiales. Puisque le nuage des individus est centré sur le centre de gravité, la moyenne du facteur est nulle :

$$\sum_{i=1}^I F_\alpha(i) = 0 \quad (1.18)$$

et sa variance vaut

$$\text{Var}(F_\alpha) = \lambda_\alpha \quad (1.19)$$

La coordonnée du point - individu i sur cet axe s'écrit explicitement :

$$F_\alpha(i) = \sum_{k=1}^K u_{\alpha k} z_{ik} \quad (1.20)$$

```
# coordonnées factorielles des individus
rowcoord = pd.DataFrame(np.dot(Z,eigenvector),index = donnee.index,
                        columns = index)
# Affichage
print(rowcoord.loc[:,['Dim.1','Dim.2']].T.round(2))
```

Afin d'avoir une représentation plane du nuage des individus, nous créons une fonction de visualisation des individus.

```
# Fonction de visualisation en 2D
def pca_row_plot(data,eigen,axe1,axe2,figsize=None):
    try:
        if axe1==axe2:
            raise ValueError('Erreur: axe1 doit être différent de axe2.')
        elif axe1>axe2:
            raise ValueError('Erreur: axe1 doit être inférieur à axe2.')
        elif axe1<0 or axe2<0:
            msg = 'Erreur: les valeurs des axes doivent être positives ou nulles.'
            raise ValueError(msg)
    else:
        # set limite
        n = data.shape[0]
        # Valeurs propres
```

```

percent = np.array([100*x/sum(eigen) for x in eigen])
dimi = round(percent[axe1],2); dimj = round(percent[axe2],2)

# Graphique
fig, axes = plt.subplots(figsize = figsize); axes.grid()
axes.axis([-8,8,-6,6])
axes.set_title("Projection des individus")
axes.set_xlabel(f"Dim.{1+axe1} ({dimi}%)")
axes.set_ylabel(f"Dim.{1+axe2} ({dimj}%)")
for i in range(n):
    plt.scatter(data.iloc[i,axe1], data.iloc[i,axe2],
                c = "black", alpha = 1)
    axes.text(data.iloc[i,axe1], data.iloc[i,axe2], data.index[i],
              color = "black", fontsize = 11)
plt.axhline(0, color='blue', linestyle="--", linewidth=0.5)
plt.axvline(0, color='blue', linestyle="--", linewidth=0.5)
plt.show()

# if false then raise the value error
except ValueError as e:
    print(e)

# Nuage des individus sur les axes 1 et 2
pca_row_plot(data=rowcoord,eigen=eigenvalue,axe1=0,axe2=1,figsize=(12,8))

```

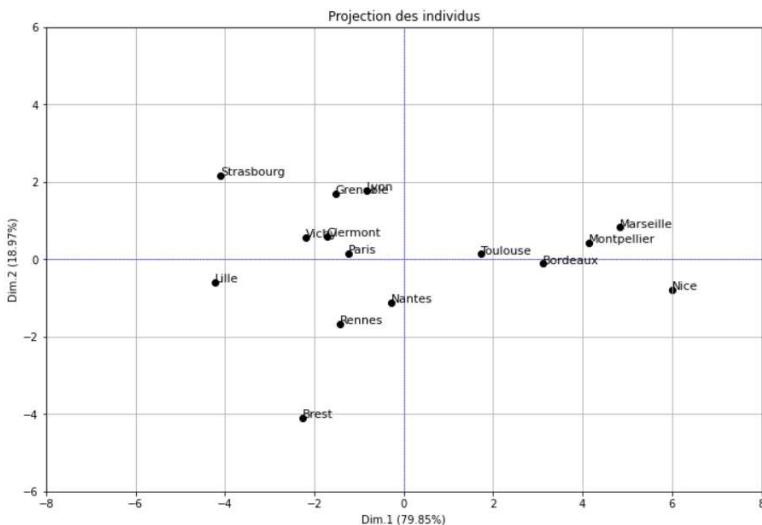


FIGURE 1.5 – Représentation plane du nuage des individus

La figure 1.5 est obtenue à partir des deux premières composantes de l'ACP normée et correspond donc au meilleur plan de représentation du nuage de points au sens de l'inertie projetée. L'inertie projetée sur ce plan correspond à la somme des deux premières valeurs propres divisées

par la somme des valeurs propres, autrement dit 98.82% ($= 79.85\% + 18.97\%$) de l'inertie totale du nuage de points. On peut y voir que Montpellier et Marseille sont très proches. Cela signifie que les températures moyennes à Montpellier et Marseille sont à peu près les mêmes et ceux quel que soit le mois de l'année. Les valeurs sont à peu près les mêmes et donc les points sont proches. De même, Rennes et Nantes sont deux villes ayant des températures proches pour les 12 mois de l'année. Par contre, Nice et Lille ont des comportements très différents. Ces deux villes sont complètement opposés sur le premier axe. Cela signifie que ce sont des villes très différentes parce que le premier axe est celui qui sépare au mieux les points.

1.3.2 Nuage des variables

1) Distances entre variables

La distance euclidienne usuelle entre deux variables j et k est donnée par :

$$d^2(j,k) = \frac{1}{I} \sum_{i=1}^I (z_{ij} - z_{ik})^2 \quad (1.21)$$

soit :

$$d^2(j,k) = \frac{1}{I} \sum_{i=1}^I z_{ij}^2 + \frac{1}{I} \sum_{i=1}^I z_{ik}^2 - \frac{2}{I} \sum_{i=1}^I z_{ij} z_{ik}$$

On rappelle que :

$$\begin{cases} \frac{1}{I} \sum_{i=1}^I z_{ij}^2 = \frac{1}{I} \sum_{i=1}^I z_{ik}^2 = 1 \\ \frac{1}{I} \sum_{i=1}^I z_{ij} z_{ik} = \text{Corr}(j,k) \end{cases}$$

d'où la relation liant la distance dans \mathbb{R}^I entre deux points-variables j et k et le coefficient de corrélation linéaire entre ces variables :

$$d^2(j,k) = 2(1 - \text{Corr}(j,k)) \quad (1.22)$$

ce implique que :

$$0 \leq d^2(j,k) \leq 4$$

Dans l'espace \mathbb{R}^I , le cosinus de l'angle de deux vecteurs-variables est le coefficient de corrélation entre ces deux variables :

$$\text{Corr}(j,k) = \cos(j,k) \quad (1.23)$$

Si ces deux variables sont à la distance 1 de l'origine (i.e. si elles sont de variance unité), le cosinus n'est autre que leur produit scalaire. Par conséquent, les proximités entre variables s'interprètent donc en termes de corrélations.

Distance en variables

```
vardist = pd.DataFrame(np.zeros(shape=(p,p), dtype=float), index = donnee.columns,
                      columns = donnee.columns)
for j in range(p):
```

```

for k in range(j+1,p):
    vardist.iloc[j,k] = 2*(1-corr.iloc[j,k])
# Visualisation - heatmap
plt.figure(figsize = (10,8))
sns.heatmap(vardist,xticklabels=dist.columns,yticklabels = dist.columns,
            vmin = 0, vmax = 4,cmap = sns.color_palette("Blues",12),linewidths=0.5)
plt.title('Heatmap des distances entre mois')
plt.show()

```

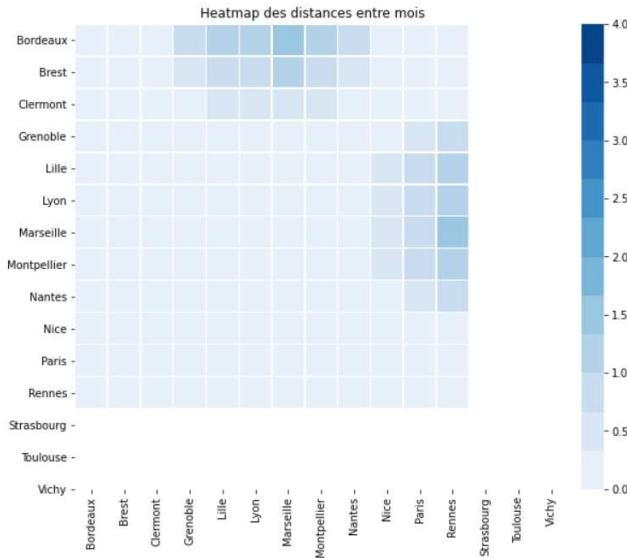


FIGURE 1.6 – Heatmap des distance entre mois

2) Coordonnées des variables

Comme pour les individus, pour avoir une représentation du nuage des variables (ou cercle de corrélation), il faut au préalable calculer leurs coordonnées factorielles. Pour calculer les coordonnées factorielles des variables, nous avons besoin des vecteurs propres et des valeurs propres.

Le vecteur propre

$$v_\alpha = \frac{1}{\sqrt{\lambda_\alpha}} Z u_\alpha \quad (1.24)$$

est un vecteur propre unitaire de ZZ' relativement à la même valeur propre λ_α . Le $\alpha^{\text{ième}}$ facteur dans \mathbb{R}^I s'écrit :

$$G_\alpha = Z' v_\alpha = \frac{1}{\sqrt{\lambda_\alpha}} Z' Z u_\alpha = u_\alpha \sqrt{\lambda_\alpha} \quad (1.25)$$

Remarque :

Comme $F_\alpha = X\mathbf{u}_\alpha$, on a :

$$G_\alpha = \frac{1}{\sqrt{\lambda_\alpha}} Z' F_\alpha \quad (1.26)$$

Ainsi, la coordonnée de la variable k sur l'axe α est donnée par :

$$G_\alpha(k) = \frac{1}{\sqrt{\lambda_\alpha}} \sum_{i=1}^I z_{ik} F_\alpha(i) \quad (1.27)$$

et l'on a :

$$G_\alpha(k) = \text{Corr}(k, F_\alpha) \quad (1.28)$$

La coordonnée d'un point-variable sur un axe n'est autre que le *coefficient de corrélation* de cette variable avec le facteur F_α (combinaison linéaire des variables).

```
# coordonnées des variables
varcoord = pd.DataFrame(eigenvector*np.sqrt(eigenvalue),columns = index,
                        index = donnee.columns)
varcoord.index.name = 'mois'
# Affichage
print(varcoord.loc[:,['Dim.1','Dim.2']].T.round(2))
```

Afin d'avoir une représentation plane du nuage des variables, nous créons une fonction de visualisation du cercle des corrélations.

```
# Visualisation du nuage des variables (=Cercle de corrélation)
def pca_var_plot(data,eigen,axe1,axe2,figsize=None):
    try:
        if axe1==axe2:
            raise ValueError('Erreur: axe1 doit être différent de axe2.')
        elif axe1>axe2:
            raise ValueError('Erreur: axe1 doit être inférieur à axe2.')
        elif axe1<0 or axe2<0:
            msg = 'Erreur: les valeurs des axes doivent être positives ou nulles.'
            raise ValueError(msg)
        else:
            p = data.shape[0]
            # Valeurs propres
            percent = np.array([100*x/sum(eigen) for x in eigen])
            dim1 = round(percent[axe1],2); dim2 = round(percent[axe2],2)

            # Graphique
            fig, axes = plt.subplots(figsize = figsize)
            axes.grid()
            axes.axis([-1.5,1.5,-1.5,1.5])
            axes.set_title("Cercle de corrélation")
            axes.set_xlabel(f"Dim.{1+axe1} ({dim1}%)")
            axes.set_ylabel(f"Dim.{1+axe2} ({dim2}%)")
```

```

for j in range(p):
    axes.arrow(0,0,data.iloc[j,axe1],data.iloc[j,axe2],
               head_width = 0.02,head_length = 0.02,color='black')
    axes.text(data.iloc[j,axe1],data.iloc[j,axe2],data.index[j],
              color = "black")
# cercle
from matplotlib.patches import Ellipse
ellipse = Ellipse((0,0),width = 2, height = 2,facecolor = "none",
                   edgecolor = "blue")
axes.add_patch(ellipse)
plt.axhline(0, color='blue',linestyle="--", linewidth=0.5)
plt.axvline(0, color='blue',linestyle="--", linewidth=0.5)
plt.show()

# if false then raise the value error
except ValueError as e:
    print(e)

# Nuage des variables
pca_var_plot(data=varcoord,eigen=eigenvalue,axe1=0,axe2=1,figsize=(10,10))

```

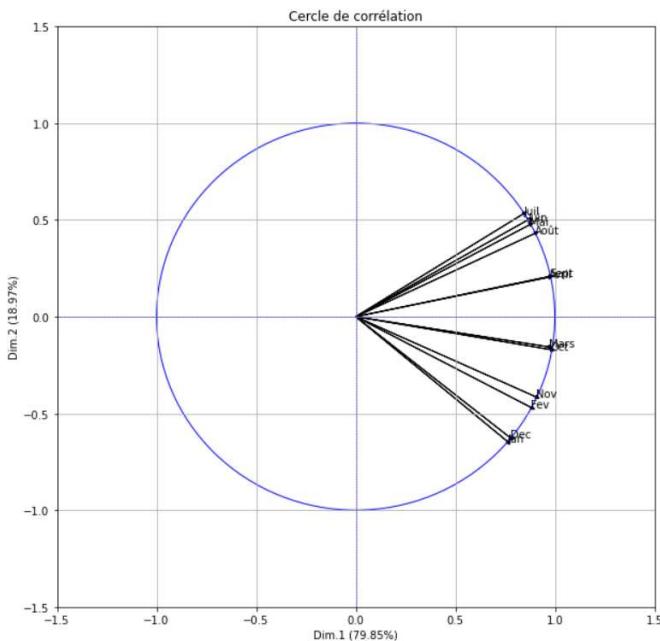


FIGURE 1.7 – Visualisation des coefficients de corrélation entre les variables et les composantes principales G_1 et G_2 .

[Ras18] propose une fonction pour la visualisation de la matrice des corrélations des variables.

i) Analyse du premier axe

On voit que toutes les variables sont corrélées à l'axe 1 (au facteur 1). On a une corrélation à l'axe qui est positive et supérieur à 0,6 pour toutes les variables. On a aussi des corrélations très élevées comme entre Octobre et Mars par exemple (des corrélations très proches de 1). Cela signifie que les températures en Octobre sont très liées aux coordonnées sur l'axe 1. Autrement dit, les villes qui sont à gauche avec une coordonnée sur l'axe 1 ont les températures faibles au mois d'Octobre. Les villes situées au milieu ont des températures moyennes en Octobre et les villes à droite ont des températures élevées en Octobre (température comparée par rapport aux autres villes durant le même mois de l'année).

On a toutes les variables qui sont très liées à l'axe 1. Par conséquent, à droite du graphique, on a des villes où il fait un peu plus chaud tout au long de l'année (par rapport à la moyenne). A gauche par contre, on a des villes où il fait plutôt froid tous les mois de l'année. Ceci représente le principal facteur de variabilité. Ce qui différencie le plus les villes, c'est qu'il y a des villes où il fait un peu plus froid tout le temps et d'autres qui enregistrent des températures plus élevées toute l'année durant.

ii) Analyse du deuxième axe

Pour cet axe, les corrélations sont un peu moins fortes. En haut du graphe des individus, on a des villes où il fait plutôt chaud en Janvier, Décembre et plutôt froid en Mai, Juin, Juillet. Il fait plutôt chaud en Janvier et Décembre parce que la corrélation avec l'axe 2 est positive; bien que n'étant pas très proche de 1. Tandis que la corrélation est négative avec les variables Mai, Juin, Juillet. On peut donc dire que les villes qui ont des coordonnées plutôt élevées pour l'axe 2 vont prendre des valeurs plutôt faibles en Mai, Juin, Juillet. Autrement dit, en haut du graphique, on va avoir des villes où il fait plutôt chaud l'hiver et froid l'Eté. Et au contraire, les villes situées en bas du graphe sont des villes où il fait plutôt froid l'Eté et plutôt chaud L'hiver.

On peut dire que le premier axe sépare les villes chaudes des villes froides, tandis que le deuxième axe, orthogonal au premier, différencie les villes à une température moyenne annuelle constante. Ce dernier sépare à haut du graphe les villes où il fait plutôt chaud l'hiver et froid l'été; donc ayant une petite amplitude thermique. Aux villes en bas du graphe où il fait plutôt froid l'hiver et chaud l'été, donc les villes ayant une forte amplitude thermique.

1.3.3 Relation entre les représentations des nuages N_I et N_K

L'analyse du nuage des observations et celle du nuage des variables s'intéressent aux lignes et respectivement aux colonnes d'une même matrice de données. La question est de savoir : quels sont alors les liens entre ces analyses ?

Rappelons que l'analyse en composantes principales est duale par nature, c'est-à-dire qu'il existe une relation entre les coordonnées factorielles des individus et celles des variables (les corrélations). Nous pouvons constater qu'en (pré-)multipliant $Z^T Z u_\alpha = \lambda_\alpha u_\alpha$ par Z , nous obtenons $Z Z^T (Z u_\alpha) = \lambda_\alpha (Z u_\alpha)$. Ainsi, à tout vecteur propre u_α de $Z^T Z$ associé à une valeur propre λ_α correspond donc un vecteur propre $(Z u_\alpha)$ de $Z Z^T$ associé à la **même** valeur propre. Notons que la norme de ce vecteur n'est pas 1 mais plutôt $\sqrt{\lambda_\alpha}$ ⁴.

4. $\|Z u_\alpha\|^2 = (Z u_\alpha)^T (Z u_\alpha) = u_\alpha^T Z^T Z u_\alpha = u_\alpha^T \lambda_\alpha u_\alpha = \lambda_\alpha$

Aussi, en (pré-)multipliant $ZZ^T v_\alpha = \mu_\alpha v_\alpha$ par Z^T , nous obtenons $Z^T Z (Z^T v_\alpha) = \mu_\alpha (Z^T v_\alpha)$. A tous vecteur propre v_α de ZZ^T associé à une valeur propre μ_α correspond alors un vecteur propre $(Z^T v_\alpha)$ de $Z^T Z$ associé à la même valeur propre. Ce vecteur propre est aussi de norme $\sqrt{\lambda_\alpha}$ et non 1. Nous pouvons conclure que les valeurs propres non nulles de même ordre ZZ^T et $Z^T Z$ sont toutes égales, $\lambda_\alpha = \mu_\alpha$. Aussi, que les relations suivantes sont valables entre vecteurs propres :

$$v_\alpha = \frac{1}{\sqrt{\lambda_\alpha}} Z u_\alpha \quad (1.29)$$

$$u_\alpha = \frac{1}{\sqrt{\lambda_\alpha}} Z^T u_\alpha \quad (1.30)$$

De ce constat résultent les relations de transition ou de dualité suivantes qui permettent de passer des projections des variables sur l'axe factoriel d'ordre α aux projections des observations sur l'axe factoriel correspondant de même ordre.

Pour obtenir les coordonnées de l'individu i sur le facteur (F_α) à partir des coordonnées des variables, la transition s'écrit :

$$F_\alpha(i) = \frac{1}{\sqrt{\lambda_\alpha}} \sum_{j=k}^K z_{ik} \times G_\alpha(k) \quad (1.31)$$

```
# relation de transition - coordonnées des individus
transition1 = Z.dot(varcoord)/np.sqrt(eigenvalue)
print(transition1.iloc[:,[0,1]].round(2))
```

Inversement, il est possible d'obtenir les coordonnées des variables $G_\alpha(k)$ à partir de celles des individus $(F_\alpha(i))$:

$$G_\alpha(k) = \frac{1}{\sqrt{\lambda_\alpha}} \frac{1}{I} \sum_{i=1}^I z_{ik} \times F_\alpha(i) \quad (1.32)$$

```
# relation de transition - coordonnées des variables
transition2 = Z.T.dot(rowcoord)/(n*np.sqrt(eigenvalue))
print(transition2.iloc[:,[0,1]].T.round(2))
```

Ces deux relations sont importantes car elles permettent de passer des résultats d'une des analyses directement aux résultats de l'autre. En général, le nombre I d'individus est nettement supérieur au nombre K de variables initiales, il est donc préférable de traiter la matrice $Z^T Z$, de dimension $K \times K$, plutôt que ZZ^T , de dimension $I \times I$. Les résultats de l'autre analyse sont ensuite facilement obtenus grâce aux relations de transition.

Par ailleurs, les relations de transition rendent possible une **interprétation simultanée** des projections des observations et des projections des variables sur les axes factoriels de même ordre. En effet, si seules quelques observations (quelques valeurs de i) présentent des valeurs très élevées z_{ik} pour une variable k , alors de (1.32), nous obtenons que la projection de cette variable $G_\alpha(k)$ sur un axe est, à un facteur multiplicatif près ($1/\sqrt{\lambda_\alpha}$), la moyenne pondérée des projections sur l'axe de même ordre ($F_\alpha(i)$) de ces observations.

1.3.4 Représentation simultanée : graphe biplot

Pour aller plus loin, certains outils proposent un graphique « **biplot** ». Le biplot est un graphique dans lequel on représente deux ensembles d'objets de nature différente. Lorsque le nombre d'individus et le nombre de variables est assez faible, il peut être intéressant, assez séduisant au demeurant, de représenter simultanément le nuage des individus et le nuage des variables. Cependant, il ne doit pas nous induire en erreur. Les deux nuages n'évoluent pas dans le même espace (l'un appartient à \mathbb{R}^K et l'autre à \mathbb{R}^I . Pour les individus, nous raisonnons en positions relatives dans le repère ; pour les variables, nous raisonnons en directions. **Mais les proximités entre individus et variables n'ont aucun sens** : un individu est du côté des variables pour lequel il prend de grandes valeurs.

```
# Nuage simultané
def biplot(data1,data2,eigen,axe1,axej,figsize=None):
    # Représentation simultanée des individus et des variables
    try:
        if axe1==axej:
            raise ValueError('Erreur: axe1 doit être différent de axej.')
        elif axe1>axej:
            raise ValueError('Erreur: axe1 doit être inférieur à axej.')
        elif axe1<0 or axej<0:
            msg = 'Erreur: les valeurs des axes doivent être positives ou nulles.'
            raise ValueError(msg)
        else:
            n,p = data1.shape
            # Valeurs propres
            percent = np.array([100*x/sum(eigen) for x in eigen])
            dimi = round(percent[axe1],2); dimj = round(percent[axej],2)

            # Biplot
            fig = plt.figure(figsize=(12,8))
            axes1 = fig.add_subplot(111)
            axes2 = axes1.twiny()
            axes2 = axes2.twinx()
            axes1.grid()
            axes2.axis([-1.2,1.2,-1.2,1.2])
            axes1.axis([-8,8,-6,6])
            axes1.set_title("Biplot")
            axes1.set_xlabel(f"Dim.{1+axe1} ({dimi}%)")
            axes1.set_ylabel(f"Dim.{1+axej} ({dimj}%)")
            # Affichage des individus
            for i in range(n):
                axes1.scatter(data1.iloc[i,0],data1.iloc[i,1],c = "black",
                              alpha = 1)
                axes1.text(data1.iloc[i,0],data1.iloc[i,1],data1.index[i],
                           color = "black", fontsize=12)
            # Affichage des variables
            for k in range(p):
                axes2.arrow(0,0,data2.iloc[k,0],data2.iloc[k,1],
                           head_width = 0.02,head_length = 0.02,
```

```

        color='blue', linestyle='--')
    axes2.text(data2.iloc[k,0],data2.iloc[k,1],data2.index[k],
               color = "blue", fontsize=12)
    plt.axhline(0, color='blue',linestyle="--", linewidth=0.5)
    plt.axvline(0, color='blue',linestyle="--", linewidth=0.5)
    plt.show()

except ValueError as e:
    print(e)

# Affichage
biplot(data1=rowcoord,data2=varcoord,eigen=eigenvalue,axe1=0,axej=1,
       figsize=(10,10))

```

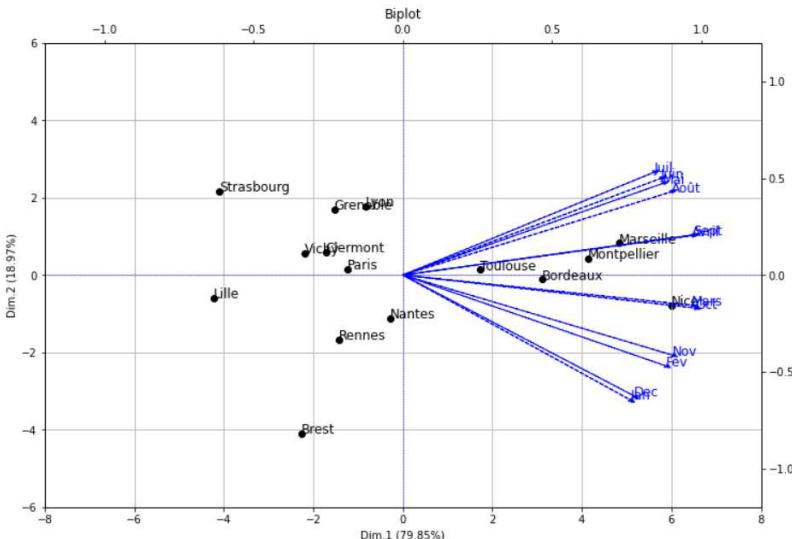


FIGURE 1.8 – Représentation “Biplot” - Données ”températures”

1.4 Outils pour l'aide à l'interprétation

Un des objectifs de l'ACP est d'aider à une meilleure compréhension des données à travers les possibilités d'interprétation des relations entre multiples variables, des positionnement relatifs de certains individus ou classes d'individus par rapport aux variables. Les aides à l'interprétation permettent de savoir jusqu'où aller dans l'interprétation, en identifiant les individus et variables mal représentées par leurs projections sur les axes et les éventuels individus dont l'impact sur l'analyse est excessif.

1.4.1 Analyse du point de vue des individus

1) Contribution absolues des individus

La contribution d'un individu à la formation d'un axe est la part d'information apportée par cet individu pour former l'axe. Elle permet de déterminer les individus qui pèsent le plus dans la définition de chaque facteur. La contribution d'un individu à la variance expliquée par un axe factoriel est le rapport entre le carré de la projection de l'observation sur l'axe et la variance expliquée par cet axe, c'est-à-dire la valeur propre correspondant à l'axe :

$$CTR_\alpha(i) = 100 \times \frac{F_\alpha^2(i)}{n \times \lambda_\alpha} \quad (1.33)$$

```
# contributions factorielles des individus
rowcontrib = rowcoord.apply(lambda x: 100*x**2/(n*eigenvalue),axis=1)
# Affichage
print(rowcontrib.iloc[:,[0,1]].round(2))
```

Les sommes en lignes (sur chaque facteur) sont égales à 100 ici :

$$\sum_{i=1}^I CTR_\alpha(i) = 100, \quad \forall \alpha = 1, \dots, K \quad (1.34)$$

Nous créons un outil d'affichage graphique des contributions les plus importantes selon les facteurs.

```
# Affichage graphique des contributions
def plot_graph(data,axis,xlabel,title,figsize=None):
    p = data.shape[1]
    try:
        if axis<0 or axis>p:
            raise ValueError(f'axis doit être compris entre {0} et {p-1}.')
        else:
            sort = data.sort_values(by=f'Dim.{1+axis}', ascending=True)
            sort.iloc[:,axis].plot.barh(figsize=figsize)
            plt.xlabel(xlabel)
            plt.title(f'{title} in axis {1+axis}')
            plt.show()
    except ValueError as f:
        print(f)

# Contribution axe 1
plot_graph(data=rowcontrib, axis=0, xlabel = 'Contribution (%)',
            title = 'Rows contributions', figsize=(10,8))

# Contribution axe 2
plot_graph(data=rowcontrib, axis=1, xlabel = 'Contribution (%)',
            title = 'Rows contributions', figsize=(10,8))
```

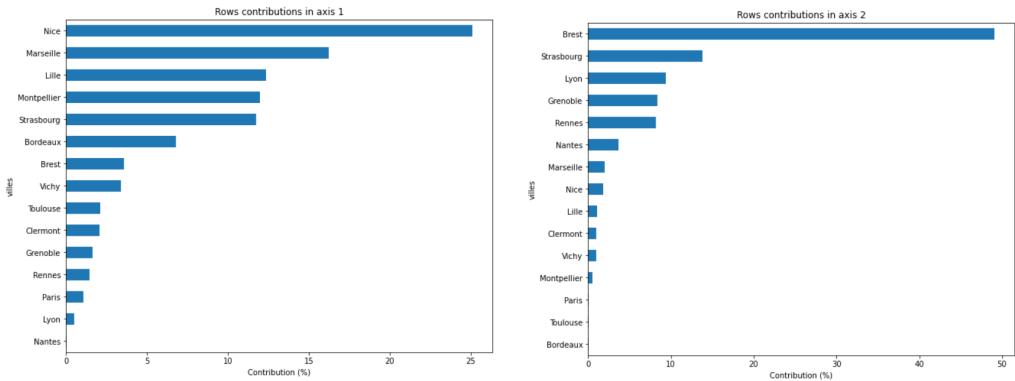


FIGURE 1.9 – Contributions des villes sur les facteurs 1 et 2

Le graphe (1.9) nous présente les contributions des villes au premier facteur. On voit que ce premier facteur est dû essentiellement à 5 villes (Nice, Marseille, Lille, Montpellier et Strasbourg) qui totalisent 77.4% de son inertie.

Grâce au graphique (1.9), on voit que le deuxième facteur est dû pour moitié (49.1%) à la ville de Brest, qui est donc assez particulière du point de vue climatique. Cependant, remarquons que la différence d'inertie entre le deuxième axe et le troisième axe ($2.28 - 0.08 = 2.20$) est beaucoup plus grande que l'inertie de Brest le long du deuxième axe ($2.28 \times 0.49 = 1.12$). Donc, même sans la ville Brest, ce deuxième axe serait donc apparu.

2) Qualité de représentation - les COS² (cosinus carré)

La qualité de représentation de l'individu i sur l'axe α avec :

$$COS_{\alpha}^2(i) = \frac{F_{\alpha}^2(i)}{d^2(i, G_I)} = \frac{F_{\alpha}^2(i)}{\sum_{k=1}^K z_{ik}^2} \quad (1.35)$$

```
# Cosinus carré des individus
rowcos2 = rowcoord.apply(lambda x : x**2/rowdisto, axis=0)
#Affichage
print(rowcos2.iloc[:, [0, 1]].T.round(2))
```

Conformément à la théorie, pour chaque individu i , la somme des COS^2 sur l'ensemble des facteurs est égale à 1.

$$\sum_{\alpha=1}^K COS_{\alpha}^2(i) = 1, \quad \forall i \quad (1.36)$$

```
# Cosinus carré axe 1
plot_graph(data=rowcos2, axis=0, xlabel = 'Cosinus 2', title = 'Rows cosinus 2',
            figsize=(10,8))
```

```
# Cosinus carré axe 2
```

```
plot_graph(data=rowcos2,axis=1,xlabel = 'Cosinus 2',title = 'Rows cosinus 2',
figsize=(10,8))
```

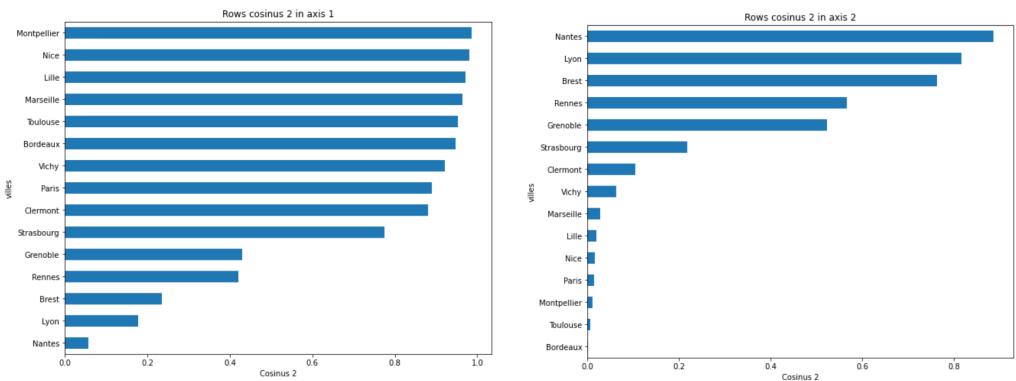


FIGURE 1.10 – Cosinus carré des villes sur les facteurs 1 et 2

1.4.2 Analyse du point de vue des variables

1) Contributions absolues des variables

La contribution d'une variable à la formation d'un axe est la part d'information apportée par cette variable pour former l'axe. Elle est basée sur le carré de la corrélation, mais relativisée par l'importance de l'axe (valeur propre). En d'autres termes, la contribution d'une variable à la variance appliquée par un axe factoriel est le carré de la projection de la variable sur l'axe et la variance totale expliquée par cet axe (la valeur propre correspondante) :

$$CTR_\alpha(k) = 100 \times \frac{G_\alpha^2(k)}{\lambda_\alpha} \quad (1.37)$$

```
# contributions factorielles des variables
varcontrib = varcoord.apply(lambda x : 100*x**2/eigenvalue, axis=1)
# Affichage
print(varcontrib.iloc[:, [0,1]].T.round(2))
```

Les sommes en colonnes sont égales à 100. Nous faisons appel à la fonction `plot_graph` créée plus haut.

```
# Contribution variable axe 1
plot_graph(data=varcontrib, axis=0, xlabel = 'Contribution (%)',
           title = 'Columns contributions', figsize=(10,8))

# Contribution variable axe 2
plot_graph(data=varcontrib, axis=1, xlabel = 'Contribution (%)',
           title = 'Columns contributions', figsize=(10,8))
```

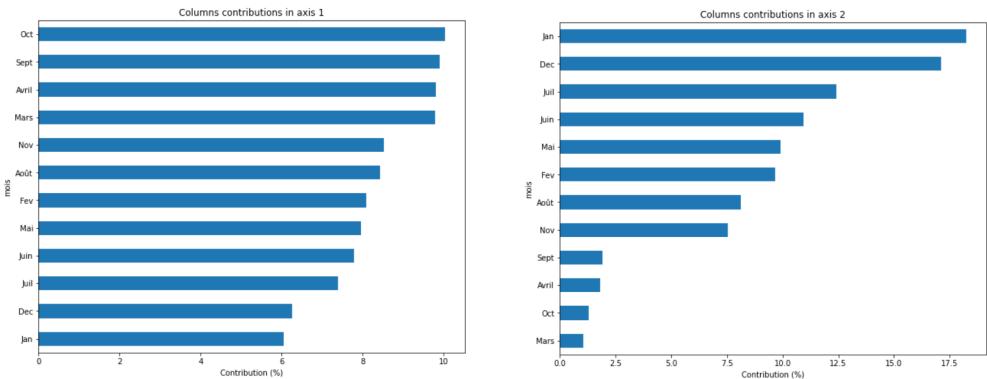


FIGURE 1.11 – Contributions des mois sur les facteurs 1 et 2

2) Qualité de représentation des variables (COS^2)

La qualité de la représentation d'une variable est mesurée par le Cos^2 compris entre 0 et 1. Il représente le cosinus de l'angle formé entre une variable et sa projection. Le Cos^2 est déterminé en éllevant au carré la corrélation.

$$COS_{\alpha}^2(k) = G_{\alpha}^2(k) \quad (1.38)$$

```
# Cosinus carré des variables
varcos2 = varcoord.apply(lambda x : x**2)
# Affichage
print(varcos2.iloc[:,[0,1]].T.round(2))

# Cosinus carré variable axe 1
plot_graph(data=varcos2, axis=0, xlabel = 'Cosines 2', title = 'Columns cosines 2',
            figsize=(10,8))

# Cosinus carré variable axe 2
plot_graph(data=varcos2, axis=1, xlabel = 'Cosines 2', title = 'Columns cosines 2',
            figsize=(10,8))
```

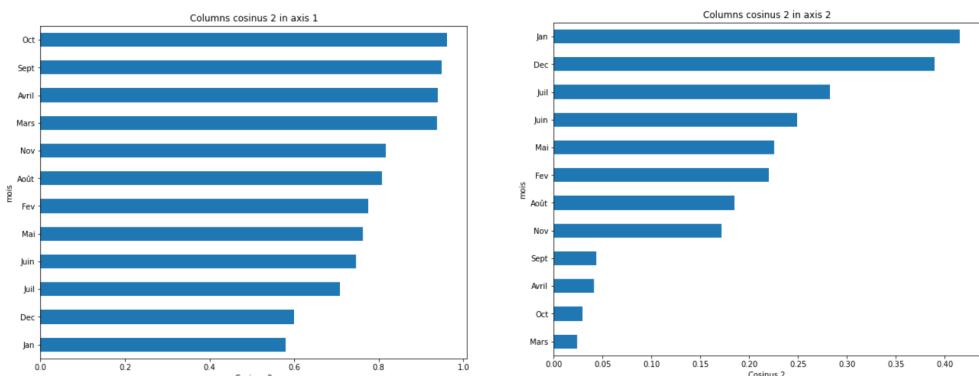


FIGURE 1.12 – Cosinus carrés des mois sur les facteurs 1 et 2

1.5 Éléments supplémentaires

Les données du tableau (1.2) ont servi à déterminer les bases factorielles (formation des axes) des espaces directs et duals. Néanmoins, on peut représenter dans celles – ci des données ne figurant pas dans le tableau, c'est-à-dire n'ayant pas participé à la détermination des axes factoriels. On espère de ces variables dites « supplémentaires » un complément d'éclairage dans l'analyse.

1.5.1 Individus supplémentaires

On utilise souvent les individus supplémentaires en cas de mélange de la population. On a observé au courant d'une année, les températures en mi-journée de plusieurs villes européennes. Le tableau (1.11) donne les valeurs obtenues.

Villes	Jan	Fev	Mars	Avril	Mai	Juin	Juil	Août	Sept	Oct	Nov	Dec
Amsterdam	2,9	2,5	5,7	8,2	12,5	14,8	17,1	17,1	14,5	11,4	7,0	4,4
Anvers	3,1	2,9	6,2	8,9	12,9	15,5	17,9	17,6	14,7	11,5	6,8	4,7
Athènes	9,1	9,7	11,7	15,4	20,1	24,5	27,4	27,2	23,8	19,2	14,6	11,0
Barcelone	9,1	10,3	11,8	14,1	17,4	21,2	24,2	24,1	21,7	17,5	13,1	10,0
Berlin	-0,2	0,1	4,4	8,2	13,8	16,0	18,3	18,0	14,4	10,0	4,2	1,2
Bruxelles	3,3	3,3	6,7	8,9	12,8	15,6	17,8	17,8	15,0	11,1	6,7	4,4
Budapest	-1,1	0,8	5,5	11,6	17,0	20,2	22,0	21,3	16,9	11,3	5,1	0,7
Copenhague	-0,4	-0,4	1,3	5,8	11,1	15,4	17,1	16,6	13,3	8,8	4,1	1,3
Cracovie	-3,7	-2,0	1,9	7,9	13,2	16,9	18,4	17,6	13,7	8,6	2,6	-1,7
Dublin	4,8	5,0	5,9	7,8	10,4	13,3	15,0	14,6	12,7	9,7	6,7	5,4

TABLE 1.11 – Individus supplémentaires

On souhaite positionner ces villes supplémentaires, par rapport à celles existantes. Au préalable, importons cette base supplémentaire.

```
# individus supplémentaires
ind_sup = pd.read_excel('temperature.xlsx', sheet_name=3, header=0, index_col=0)
```

On calcule quelques statistiques sur cette nouvelle base de données.

```
# statistiques descriptives
print(ind_sup.describe(include='all').round(2))
```

stats	Jan	Fev	Mars	Avril	Mai	Juin	Juil	Août	Sept	Oct	Nov	Dec
count	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00
mean	2.69	3.22	6.11	9.68	14.12	17.34	19.52	19.19	16.07	11.91	7.09	4.14
std	4.22	4.11	3.47	3.04	3.07	3.48	3.81	3.85	3.73	3.57	3.86	4.03
min	-3.70	-2.00	1.30	5.80	10.40	13.30	15.00	14.60	12.70	8.60	2.60	-1.70
25%	-0.35	0.28	4.68	7.98	12.58	15.42	17.28	17.23	13.88	9.77	4.42	1.23
50%	3.00	2.70	5.80	8.55	13.05	15.80	18.10	17.70	14.60	11.20	6.70	4.40
75%	4.42	4.58	6.58	10.92	16.20	19.38	21.10	20.48	16.42	11.48	6.95	5.22
max	9.10	10.30	11.80	15.40	20.10	24.50	27.40	27.20	23.80	19.20	14.60	11.00

TABLE 1.12 – Statistiques sur les données supplémentaires

On stocke dans deux variables `m` et `p` les dimensions de notre tableau.

```

# dimensions
m,p = ind_sup.shape
print(f'm = {m} et p = {p}!')

m = 10 et p = 12

```

Afin de pouvoir calculer les coordonnées factorielles des individus supplémentaires, nous devons au préalable centrer et réduire nos données. Précisons que le centrage et la réduction se font par rapport aux moyennes et aux écarts-types calculés sur les individus actifs uniquement (ceux qui ont servi à la détermination des axes factoriels).

```

# Normalisation des données supplémentaires
A = ind_sup.apply(lambda x : (x-donnee.mean(axis=0))/donnee.std(ddof=0, axis=0),
                  axis=1)
print(A.round(2))

```

Villes	Jan	Fev	Mars	Avril	Mai	Juin	Juil	Août	Sept	Oct	Nov	Dec
Amsterdam	-0.55	-1.29	-1.71	-2.03	-1.33	-1.75	-1.33	-1.27	-1.39	-0.52	-0.53	-0.24
Anvers	-0.45	-1.07	-1.38	-1.52	-1.05	-1.35	-0.94	-1.01	-1.28	-0.46	-0.65	-0.08
Athènes	2.64	2.70	2.35	3.23	3.90	3.85	3.68	3.93	3.82	3.89	3.83	3.25
Barcelone	2.64	3.03	2.41	2.28	2.04	1.94	2.12	2.33	2.64	2.93	2.97	2.72
Berlin	-2.15	-2.62	-2.59	-2.03	-0.44	-1.06	-0.75	-0.81	-1.45	-1.31	-2.14	-1.93
Bruxelles	-0.35	-0.85	-1.04	-1.52	-1.12	-1.29	-0.99	-0.91	-1.11	-0.69	-0.70	-0.24
Budapest	-2.62	-2.23	-1.85	0.45	1.77	1.37	1.05	0.89	-0.05	-0.58	-1.62	-2.19
Copenhague	-2.26	-2.90	-4.69	-3.79	-2.29	-1.40	-1.33	-1.53	-2.06	-1.99	-2.20	-1.87
Cracovie	-3.96	-3.79	-4.29	-2.25	-0.85	-0.54	-0.70	-1.01	-1.84	-2.10	-3.06	-3.46
Dublin	0.43	0.09	-1.58	-2.33	-2.77	-2.62	-2.35	-2.56	-2.40	-1.48	-0.70	0.29

TABLE 1.13 – Données supplémentaires centrées et réduites

On peut à présent calculer les coordonnées factorielles de ces nouveaux individus en utilisant l'équation (1.31) qui nous donne la relation entre les coordonnées des variables et celles des individus. Si on note \mathbf{A} la matrice des données centrées et réduites pour les individus supplémentaires, alors la coordonnée d'un individu supplémentaire i' sur l'axe de rang α est :

$$F_\alpha(i') = \frac{1}{\sqrt{\lambda_\alpha}} \sum_{k=1}^K a_{i'k} G_\alpha(k) \quad (1.39)$$

Ainsi, le calcul de la coordonnée de i' se fait uniquement à partir des variables actives. Il n'est donc pas nécessaire de disposer des valeurs prises par les individus supplémentaires pour les variables supplémentaires.

```

# coordonnées factorielles des villes supplémentaires
rowsupcoord = pd.DataFrame(np.zeros(shape=(m,p), dtype=float), columns = index,
                           index = ind_sup.index)
for j in range(p):
    rowsupcoord.iloc[:,j] = 1/np.sqrt(eigenvalue[j])*A.dot(varcoord.iloc[:,j])

# Affichage
rowsupcoord.index.name = 'villes'
print(rowsupcoord.iloc[:,[0,1]].T.round(2))

```

villes	<i>Amsterdam</i>	<i>Anvers</i>	<i>Athènes</i>	<i>Barcelone</i>	<i>Berlin</i>	<i>Bruxelles</i>	<i>Budapest</i>	<i>Copenhague</i>	<i>Cracovie</i>	<i>Dublin</i>
Dim.1	-4.09	-3.30	11.84	8.66	-5.56	-3.17	-1.51	-8.27	-7.97	-5.34
Dim.2	-1.18	-0.85	0.83	-1.22	2.08	-0.86	5.11	1.07	4.28	-3.75

TABLE 1.14 – Coordonnées factorielles des individus supplémentaires sur les axes 1 et 2

Il ne reste plus qu'à les représenter dans le premier plan factoriel parmi les observations actives.

```
# Fonction de visualisation en 2D
def pca_rowsup_plot(data1,data2,eigen,axe1,axe2,figsize=None):
    try:
        if axe1==axe2:
            raise ValueError('Erreur: axe1 doit être différent de axe2.')
        elif axe1>axe2:
            raise ValueError('Erreur: axe1 doit être inférieur à axe2.')
        elif axe1<0 or axe2<0:
            msg = 'Erreur: les valeurs des axes doivent être positives ou nulles.'
            raise ValueError(msg)
        else:
            n = data1.shape[0]
            m = data2.shape[0]
            # Valeurs propres
            percent = np.array([100*x/sum(eigen) for x in eigen])
            dim1 = round(percent[axe1],2); dim2 = round(percent[axe2],2)

            # Graphique
            fig, axes = plt.subplots(figsize = figsize); axes.grid()
            axes.axis([-15,15,-8,8])
            axes.set_title("Projection des individus")
            axes.set_xlabel(f"Dim.{1+axe1} ({dim1}%)")
            axes.set_ylabel(f"Dim.{2+axe2} ({dim2}%)")
            # Positionnement des individus actifs
            for i in range(n):
                plt.scatter(data1.iloc[i,axe1], data1.iloc[i,axe2],
                            c = "black", alpha = 1)
                axes.text(data1.iloc[i,axe1],data1.iloc[i,axe2],data1.index[i],
                          color = "black", fontsize = 11)
            # positionnement des individus illustratifs
            for j in range(m):
                plt.scatter(data2.iloc[j,axe1], data2.iloc[j,axe2],
                            c = "blue", alpha = 1)
                axes.text(data2.iloc[j,axe1],data2.iloc[j,axe2],data2.index[j],
                          color = "blue", fontsize = 11)
            plt.axhline(0, color='blue', linestyle="--", linewidth=0.5)
            plt.axvline(0, color='blue', linestyle="--", linewidth=0.5)
            plt.show()
```

```

# if false then raise the value error
except ValueError as e:
    print(e)

# Nuage des individus
pca_rowsup_plot(data1 = rowcoord,data2 = rowsupcoord,eigen = eigenvalue,
axe1 = 0,axej = 1,figsize=(10,8))

```

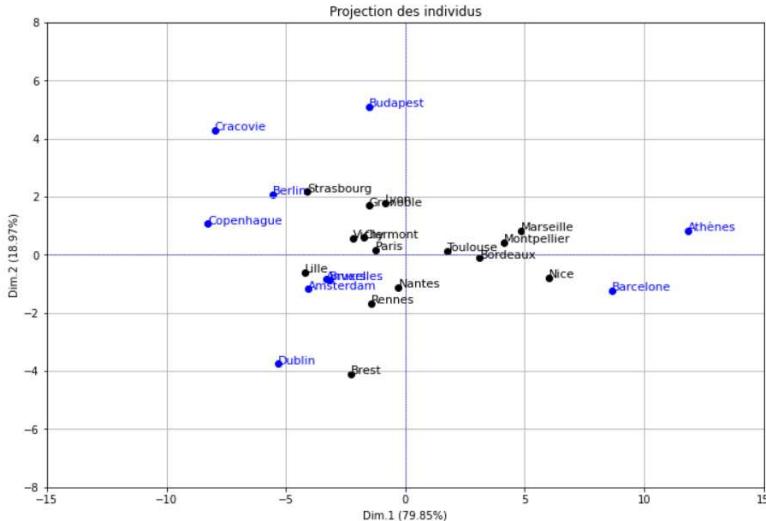


FIGURE 1.13 – Positionnement des individus supplémentaires dans le premier plan factoriel

L’observation du graphique (1.13) montre que les villes Bruxelles et Amsterdam sont proches de Lille. Ce qui signifie que, globalement, les températures dans ces villes au courant de l’année sont presqu’identiques à celles qu’on observe à Lille. Ces villes s’opposent à Athènes et Barcelone. Nous avons également une opposition entre Budapest et Dublin qui sont éloignées du plan.

1.5.2 Variables supplémentaires

Les variables supplémentaires ont pour vocation de renforcer l’interprétation des composantes. Elles ne sont pas utilisées pour leur construction, elles interviennent seulement après coup pour mieux comprendre et commenter les résultats.

1) Variables supplémentaires quantitatives

Par définition, une variable quantitative supplémentaire n’intervient pas dans le calcul des distances entre individus. On la représente de la même façon que les variables actives en tant qu’aide à l’interprétation du nuage des individus.

Rappelons que les variables Latitude et Longitude présentent dans le tableau (1.2) n’ont pas été utilisées tout au long de notre étude. Elles ont été mises en supplémentaires. A ces variables, nous

avons ajouter les moyennes et les amplitudes (maximum - minimum) de température calculées sur chaque ville. Nous chargeons notre jeu de données.

```
# variables supplémentaires quantitatives
quant_sup = pd.read_excel('temperature.xlsx', sheet_name=3, header=0, index_col=0)
```

Villes	moy	amp	Lat	Long
Bordeaux	13,3	15,4	44,5	-0,34
Brest	10,8	10,2	48,24	-4,29
Clermont	10,9	16,8	45,47	3,05
Grenoble	11,0	18,6	45,1	5,43
Lille	9,7	14,7	50,38	3,04
Lyon	11,4	18,6	45,45	4,51
Marseille	14,2	17,8	43,18	5,24
Montpellier	13,9	17,1	43,36	3,53
Nantes	11,7	13,8	47,13	-1,33
Nice	14,8	15,2	43,42	7,15
Paris	11,2	15,7	48,52	2,2
Rennes	11,1	13,1	48,05	-1,41
Strasbourg	9,7	18,6	48,35	7,45
Toulouse	12,7	16,2	43,36	1,26
Vichy	10,7	16,9	46,08	3,26

TABLE 1.15 – Données sur les variables supplémentaires

Il s'agit des variables variables géographiques (Latitude et Longitude) et les indicateurs statistiques (moyenne et amplitude). Nous stockons dans deux variables `n` et `q` les dimensions de notre tableau.

```
# Dimensions des variables supplémentaires
n, q = vsQuant.shape
print(f'm = {n} et q = {q}')
```

`m = 15 et q = 4`

Afin de positionner ces variables dans le plan factoriel, nous devons au préalable calculer les corrélations avec les axes factoriels. La coordonnée de la variable supplémentaire k' sur l'axe α correspond au coefficient de corrélation linéaire entre k' et F_α l'axe principal α . Formellement, on peut utiliser les relations de transition, équation (1.32), pour calculer la coordonnée de la variable supplémentaire k' sur l'axe de rang α :

$$G_\alpha(i') = \frac{1}{\sqrt{\lambda_\alpha}} \frac{1}{I} \sum_{i \in \text{actif}} x_{ik'} F_\alpha(i) = \text{Corr}(k', F_\alpha) \quad (1.40)$$

avec `{actif}` l'ensemble des individus actifs : le calcul de la coordonnée se fait uniquement à partir des individus actifs.

```
# corrélation avec les axes factorielles
varsupcoord = pd.DataFrame(np.zeros(shape=(q, p), dtype=float), columns = index,
                           index = vsQuant.columns)
```

```

for k in range(q):
    for l in range(p):
        varsupcoord.iloc[k,l] = np.corrcoef(vsQuant.iloc[:,k],
                                             rowcoord.iloc[:,l])[1,0]

# Affichage
print(varsupcoord.iloc[:,[0,1]].T.round(2))

```

	dimension	moy	amp	Lati	Long
Dim.1		1.00	0.10	-0.84	0.17
Dim.2		-0.02	0.99	-0.31	0.79

TABLE 1.16 – Corrélation des variables supplémentaires avec les axes

```

# Visualisation du nuage des variables (=Cercle de corrélation)
def pca_varsup_plot(data1,data2,eigen,axe1,axeJ,figsize=None):
    try:
        if axe1==axeJ:
            raise ValueError('Erreur: axe1 doit être différent de axeJ.')
        elif axe1>axeJ:
            raise ValueError('Erreur: axe1 doit être inférieur à axeJ.')
        elif axe1<0 or axeJ<0:
            msg = 'Erreur: les valeurs des axes doivent être positives ou nulles.'
            raise ValueError(msg)
        else:
            p = data1.shape[0]
            q = data2.shape[0]
            # Valeurs propres
            percent = np.array([100*x/sum(eigen) for x in eigen])
            dimi = round(percent[axe1],2); dimj = round(percent[axeJ],2)

            # Graphique
            fig, axes = plt.subplots(figsize = figsize)
            axes.grid()
            axes.axis([-1.5,1.5,-1.5,1.5])
            axes.set_title("Cercle de corrélation")
            axes.set_xlabel(f"Dim.{1+axe1} ({dimi}%)")
            axes.set_ylabel(f"Dim.{1+axeJ} ({dimj}%)")
            for j in range(p):
                axes.arrow(0,0,data1.iloc[j,axe1],data1.iloc[j,axeJ],
                           head_width = 0.02,head_length = 0.02,color='black')
                axes.text(data1.iloc[j,axe1],data1.iloc[j,axeJ],data1.index[j],
                          color = "black")
            for k in range(q):
                axes.arrow(0,0,data2.iloc[k,axe1],data2.iloc[k,axeJ],
                           head_width = 0.02, head_length = 0.02,
                           color='blue',linestyle='--')
                axes.text(data2.iloc[k,axe1],data2.iloc[k,axeJ],data2.index[k],
                          color = "blue")

```

```

# cercle
from matplotlib.patches import Ellipse
ellipse = Ellipse((0,0),width = 2, height = 2, facecolor = "none",
                  edgecolor = "blue")
axes.add_patch(ellipse)
plt.axhline(0, color='blue',linestyle="--", linewidth=0.5)
plt.axvline(0, color='blue',linestyle="--", linewidth=0.5)
plt.show()

# if false then raise the value error
except ValueError as e:
    print(e)

# Nuage des variables
pca_varsup_plot(data1 = varcoord,data2 = varsupcoord,eigen=eigenvalue,
                 axei=0,axej=1,figsize=(10,10))

```

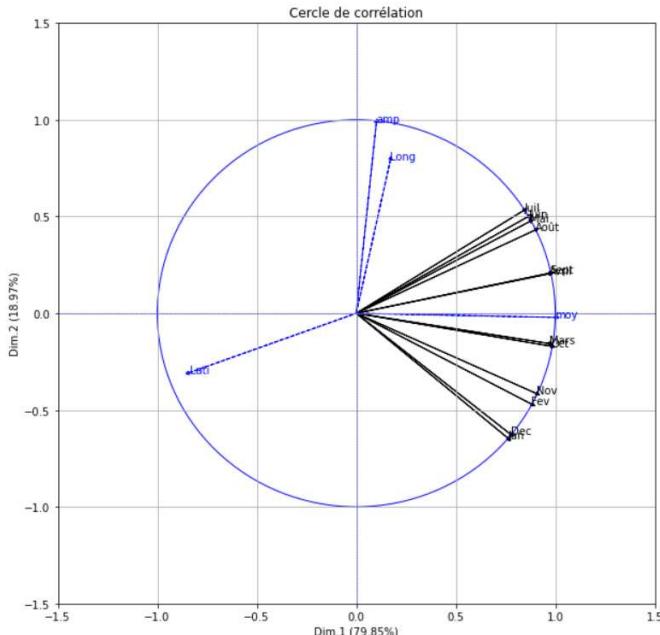


FIGURE 1.14 – Cercle des corrélations incluant les variables illustratives

Le cercle des corrélations (cf. Figure 1.14) permet une représentation conjointe des variables actives et supplémentaires.

2) Variables supplémentaires qualitatives

Tout comme les variables quantitatives supplémentaires, elles n'ont pas participé à la construction des axes. Les variables qualitatives ne peuvent pas être représentées de la même façon que les variables quantitatives supplémentaires puisqu'il est impossible de calculer la corrélation entre une variable qualitative et F_α . L'information d'une variable qualitative se situe au niveau de ses

modalités. Il est donc naturel de représenter une modalité au barycentre de l'ensemble des individus qui la possèdent. Une modalité peut donc ainsi considérée comme l'individu moyen obtenu à partir de l'ensemble des individus qui la possèdent. En ce sens, on la représente sur le graphique des individus.

```
# variables supplémentaires qualitatives
vsQual = pd.read_excel('temperature.xlsx',sheet_name=2,header=0,index_col=0)

# Dimensions des variables supplémentaires
n, r = vsQual.shape
print(f'm = {n} et r = {r}')

m = 15 et r = 1
```

Nous disposons d'une variable qualitative qui traduit le groupe d'appartenance des villes. On récupère ensuite la liste des modalités contenues par cette dernière.

```
# Modalités de la variable
modalites = np.unique(vsQual)
print('modalite :',modalites)

modalite : ['A' 'B' 'C']
```

L'information issue d'une variable qualitative supplémentaire peut être également représentée par l'intermédiaire d'un code couleur : l'ensemble des individus qui possèdent une modalité sont coloriés suivant une même couleur. Cela permet de visualiser la dispersion autour des barycentres associés aux modalités.

Nous représentons les individus dans le plan factoriel coloriés selon la modalité (le groupe) d'appartenance de la variable qualitative.

```
# Fonction de visualisation en 2D
def pca_rowqual_plot(data,vsqual,eigen,axe1,axe2,figsize=None):
    try:
        if axe1==axe2:
            raise ValueError('Erreur: axe1 doit être différent de axe2.')
        elif axe1>axe2:
            raise ValueError('Erreur: axe1 doit être inférieur à axe2.')
        elif axe1<0 or axe2<0:
            msg = 'Erreur: les valeurs des axes doivent être positives ou nulles.'
            raise ValueError(msg)
        else:
            # set limite
            n = data.shape[0]
            # Valeurs propres
            percent = np.array([100*x/sum(eigen) for x in eigen])
            dim1 = round(percent[axe1],2); dim2 = round(percent[axe2],2)

            # Graphique
            fig, axes = plt.subplots(figsize = figsize); axes.grid()
            axes.axis([-8,8,-6,6])
```

```

axes.set_title("Projection des individus")
axes.set_xlabel(f"Dim.{1+axe1} ({dimi}%)")
axes.set_ylabel(f"Dim.{1+axe2} ({dimj}%)")
cdict = {'A': 'green', 'B': 'blue', 'C': 'red'}
for group in np.unique(vsqual):
    idx = np.where(vsqual==group)
    axes.scatter(data.iloc[:,axe1].values[idx[0]],
                 data.iloc[:,axe2].values[idx[0]],
                 label = group, c = cdict[group])
    for i in idx[0]:
        axes.text(data.iloc[:,0].values[i],data.iloc[:,1].values[i],
                  s=data.index[i],c = cdict[group], fontsize = 11)
axes.legend()
plt.axhline(0, color='blue', linestyle="--", linewidth=0.5)
plt.axvline(0, color='blue', linestyle="--", linewidth=0.5)
plt.show()

# if false then raise the value error
except ValueError as e:
    print(e)

# Nuage des individus sur les axes 1 et 2
pca_rowqual_plot(data=rowcoord,vsqual = vsQual,eigen=eigenvalue,
                   axe1=0,axe2=1,figsize=(12,8))

```

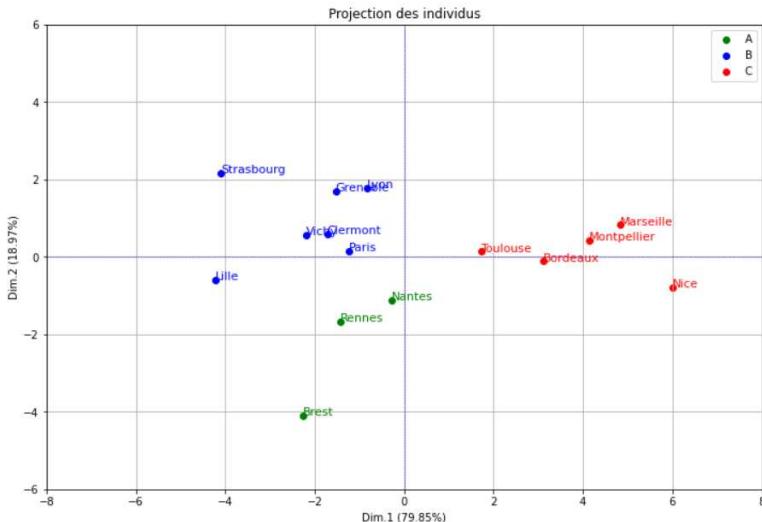


FIGURE 1.15 – Nuage des individus

Nous calculons ensuite les positions des barycentres conditionnels dans le plan factoriel. Nous le faisons uniquement pour les axes 1 et 2.

```

# Barycentre des modalités
df = pd.concat([rowcoord,vsQual],axis=1)
# Moyennes conditionnelles
condmean = df.pivot_table(index = 'groupe',values = ['Dim.1','Dim.2'],
                           aggfunc = pd.Series.mean)
print(condmean.T.round(2))

```

groupe	A	B	C
Dim.1	-1.33	-2.27	3.97
Dim.2	-2.29	0.91	0.10

TABLE 1.17 – Barycentre des modalités

Ces barycentres correspondent également aux coordonnées factorielles. Plaçons-les dans le repère pour disposer d'une vision d'ensemble.

```

# Fonction de visualisation en 2D
def pca_rowsupqual_plot(data1,data2,data3,eigen,axe1,axeJ,figsize=None):
    try:
        if axe1==axeJ:
            raise ValueError('Erreur: axe1 doit être différent de axeJ.')
        elif axe1>axeJ:
            raise ValueError('Erreur: axe1 doit être inférieur à axeJ.')
        elif axe1<0 or axeJ<0:
            msg = 'Erreur: les valeurs des axes doivent être positives ou nulles.'
            raise ValueError(msg)
    else:
        n = data1.shape[0];m = data2.shape[0];r = data3.shape[0]
        # Valeurs propres
        percent = np.array([100*x/sum(eigen) for x in eigen])
        dimi = round(percent[axe1],2); dimj = round(percent[axeJ],2)

        # Graphique
        fig, axes = plt.subplots(figsize = figsize); axes.grid()
        axes.axis([-15,15,-8,8])
        axes.set_title("Projection des individus")
        axes.set_xlabel(f"Dim.{1+axe1} ({dimi}%)")
        axes.set_ylabel(f"Dim.{1+axeJ} ({dimj}%)")
        # Positionnement des individus actifs
        for i in range(n):
            plt.scatter(data1.iloc[i,axe1], data1.iloc[i,axeJ],
                        c = "black", alpha = 1)
            axes.text(data1.iloc[i,axe1],data1.iloc[i,axeJ],data1.index[i],
                      color = "black", fontsize = 11)
        # positionnement des individus illustratifs
        for j in range(m):
            plt.scatter(data2.iloc[j,axe1], data2.iloc[j,axeJ],
                        c = "blue", alpha = 1)

```

```

        axes.text(data2.iloc[j,axe1],data2.iloc[j,axe2],data2.index[j],
                   color = "blue", fontsize = 11)
    # positionnement des modalités
    for k in range(r):
        plt.scatter(data3.iloc[k,axe1], data3.iloc[k,axe2],
                    c = "red", alpha = 1)
        axes.text(data3.iloc[k,axe1],data3.iloc[k,axe2],data3.index[k],
                   color = "red", fontsize = 12)
    plt.axhline(0, color='blue',linestyle="--", linewidth=0.5)
    plt.axvline(0, color='blue',linestyle="--", linewidth=0.5)
    plt.show()

# if false then raise the value error
except ValueError as e:
    print(e)

# Nuage des individus
pca_rowsupqual_plot(data1 = rowcoord,data2 = rowsupcoord,data3 = condmean,
                     eigen = eigenvalue,axe1 = 0,axe2 = 1,figsize=(12,8))

```

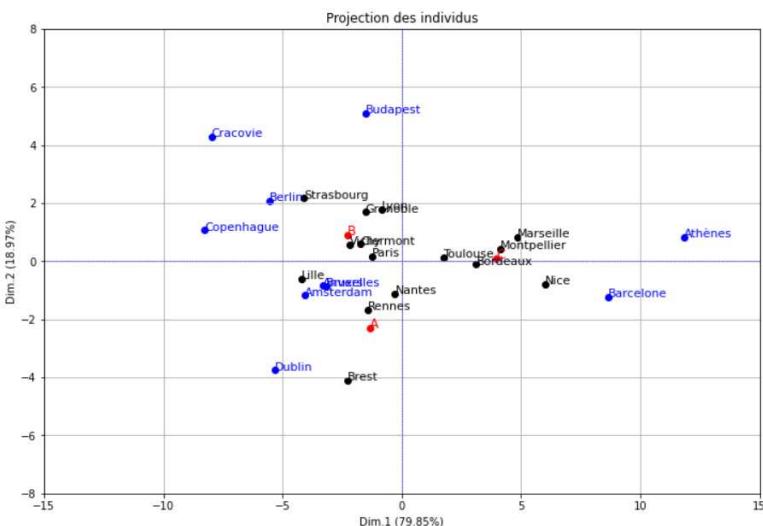


FIGURE 1.16 – Nuage des individus

1.6 Les variantes de l'analyse en composantes principales

1.6.1 ACP avec rotation

La force de l'ACP [voir [Ste12]], qui est de projeter le maximum d'inertie sur le premier axe, peut se révéler une faiblesse quand on veut repérer des groupes de variables. Le plan factoriel montre bien quelles sont les corrélations entre variables, mais les variables ont tendance à être

toutes orientées dans la direction du premier axe (en vertu de l'effet « taille », qui oppose les valeurs élevées aux valeurs faibles des variables), certaines sont entre plusieurs axes, et on ne voit pas toujours de groupes naturels de variables. Un moyen d'y remédier consiste à faire pivoter les axes de l'ACP, de façon à rapprocher au mieux les variables sur les différents axes, en remplaçant le critère de maximisation de l'inertie restituée sur le premier axe par un autre critère facilitant l'interprétation. Ce critère dépend de la méthode, mais de toute façon l'inertie totale ne change pas après rotation : seule change la décomposition.

Les rotations sont des rotations orthogonales ou des rotations obliques. Dans les premières, les facteurs ne sont pas corrélés, ce qui permet une meilleure interprétation. Dans les secondes, les facteurs ne sont plus orthogonaux, ce qui rend l'interprétation plus difficile, mais présente l'avantage que les valeurs propres sont plus fortes et que la corrélation des facteurs avec les variables est plus forte.

- Les principales ACP obliques sont les ACP oblimin et promax, la seconde étant plus rapide et utilisée sur de gros volumes de données.
- Les principales ACP orthogonales sont les ACP varimax, quartimax et equamax, cette dernière étant un compromis entre les deux premières (et l'ACP orthomax généralise les trois précédentes).

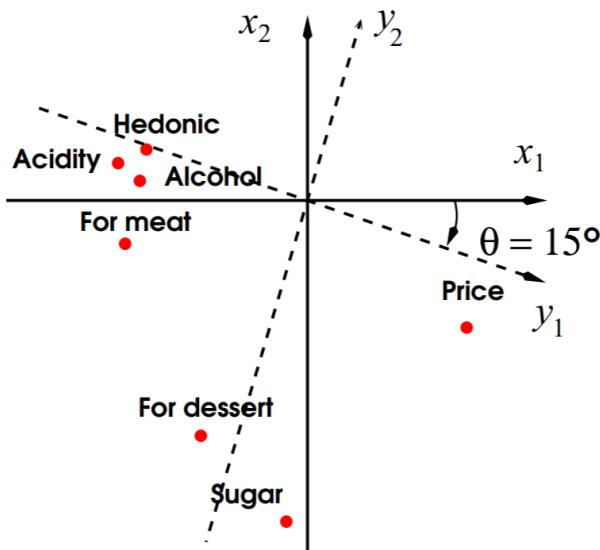


FIGURE 1.17 – Exemple de rotation VARIMAX

La plus répandue des ACP avec rotation est l'ACP varimax. Son principe est de maximiser pour chaque facteur, non pas la somme des carrés des coefficients de corrélation de ce facteur avec l'ensemble des variables, mais la variance de ces coefficients de corrélation, ce qui fait que chaque facteur est fortement corrélé à quelques variables et faiblement corrélé aux autres. Donc sur chaque axe, certaines variables ont une contribution élevée, les autres une contribution très

faible, et les axes sont facilement interprétables.

[Abd03] donne ainsi l'exemple de cinq vins décrits par leur acidité, leur teneur en sucre et en alcool, leur alliance avec la viande et les desserts, leur dimension hédonique et leur prix. Il montre (Figure 1.17) que l'ACP varimax permet de mieux expliquer les axes « prix » et « sucre ».

L'ACP promax est une méthode mixte puisqu'elle résulte d'une rotation varimax suivie d'une rotation oblique telle que les coordonnées factorielles élevées et faibles de l'espace des variables correspondent aux mêmes variables, mais avec des valeurs faibles de coordonnées qui sont encore faibles.

L'ACP quartimax est en quelque sorte dual de l'ACP varimax, puisqu'elle maximise la variance des coefficients de corrélation, non pas de chaque facteur avec l'ensemble des variables, mais de chaque variable avec l'ensemble des facteurs. Le résultat est que chaque variable est proche d'un petit nombre de facteurs. Mais contrairement à l'ACP varimax, l'ACP quartimax peut générer un facteur général avec lequel la plupart des variables sont fortement corrélées, ce qui n'est généralement pas le résultat souhaité dans de type d'analyse. Pour cette raison, ce type d'ACP est moins répandu.

Ces variantes de l'ACP se trouvent dans le package [factor_analyzer](#).

1.6.2 ACP des rangs

En présence de valeurs hors norme ou de distributions complètement asymétriques, la réduction des variables opérée dans l'ACP peut ne pas suffire à assurer de bons résultats, et on peut avoir intérêt à travailler sur les rangs des variables plutôt que sur les variables elles-mêmes. On obtient ainsi une ACP des rangs (ACP non paramétrique) plus robuste que l'ACP habituelle. Elle se calcule en remplaçant la matrice des corrélations de Pearson par la matrice des corrélations des rangs de Spearman. En effet, le coefficient de corrélation ρ de Spearman se calcule comme le coefficient de Pearson après avoir remplacé les valeurs des variables par leurs rangs :

$$\rho = \frac{\text{Cov}(r_x, r_y)}{\sigma_{r_x} \sigma_{r_y}} \quad (1.41)$$

L'interprétation de l'ACP des rangs est aussi simple que celle de l'ACP habituelle. Deux variables sont proches sur le plan factoriel si elles classent de la même façon l'ensemble des individus. Deux individus sont proches s'ils ont des rangs similaires pour l'ensemble des variables.

1.6.3 ACP sur les variables qualitatives

Cette ACP s'applique à des variables qualitatives ou numériques avec des liaisons non-linéaires, et consiste en une ACP appliquée à des variables préalablement transformées en variables numériques de façon optimale. On associe à chaque modalité d'une variable qualitative un score déterminé de façon à maximiser l'importance des premières valeurs propres, et on effectue ensuite l'ACP sur ces scores numériques plutôt que de l'effectuer sur les indicatrices des modalités, comme dans l'analyse des correspondances multiples. Cette méthode a été décrite dans [Gif90]⁵.

5. Albert Gifi est le pseudonyme collectif d'un groupe de chercheurs de l'Université de Leiden, menés par Jan de Leeuw et qui ont réalisé entre 1970 et 1990 des travaux sur l'analyse multivariée non linéaire.

villes	Disto2	Poids	Inertie	Coordonnées			Contributions			Cosinus carré		
				Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3
Bordeaux	10.287	0.067	0.686	3.121	-0.109	-0.721	6.776	0.035	49.476	0.947	0.001	0.051
Brest	21.948	0.067	1.463	-2.268	-4.093	0.115	3.579	49.069	1.257	0.234	0.733	0.001
Clermont	3.386	0.067	0.226	-1.726	0.593	-0.019	2.073	1.028	0.035	0.880	0.104	0.000
Grenoble	5.452	0.067	0.363	-1.529	1.688	-0.137	1.627	8.344	1.800	0.429	0.523	0.003
Lille	18.303	0.067	1.220	-4.217	-0.595	0.356	12.372	1.037	12.060	0.972	0.019	0.007
Lyon	3.913	0.067	0.261	-0.835	1.788	-0.023	0.485	9.365	0.050	0.178	0.817	0.000
Marseille	24.222	0.067	1.615	4.833	0.829	0.357	16.250	2.012	12.107	0.964	0.028	0.005
Montpellier	17.449	0.067	1.163	4.147	0.435	0.183	11.967	0.555	3.198	0.986	0.011	0.002
Nantes	1.402	0.067	0.093	-0.281	-1.115	-0.228	0.055	3.638	4.958	0.056	0.886	0.037
Nice	36.819	0.067	2.455	6.007	-0.789	0.205	25.106	1.825	3.994	0.980	0.017	0.001
Paris	1.734	0.067	0.116	-1.242	0.156	-0.218	1.073	0.072	4.533	0.889	0.014	0.027
Rennes	4.930	0.067	0.329	-1.439	-1.671	-0.152	1.440	8.178	2.209	0.420	0.567	0.005
Strasbourg	21.731	0.067	1.449	-4.106	2.172	0.035	11.728	13.819	0.114	0.776	0.217	0.000
Toulouse	3.164	0.067	0.211	1.736	0.136	0.043	2.007	0.054	0.179	0.953	0.006	0.001
Vichy	5.259	0.067	0.351	-2.201	0.575	0.206	3.372	0.969	4.031	0.922	0.063	0.008

mois	Coordonnées			Contributions			Cosinus carré		
	Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3
Jan	0.761	-0.644	-0.021	6.048	18.238	0.655	0.579	0.415	0.000
Fev	0.880	-0.469	-0.034	8.090	9.666	1.609	0.775	0.220	0.001
Mars	0.969	-0.156	-0.154	9.795	1.069	34.028	0.939	0.024	0.024
Avril	0.969	0.204	-0.122	9.806	1.822	21.169	0.940	0.041	0.015
Mai	0.873	0.475	-0.039	7.950	9.899	2.123	0.762	0.225	0.001
Juin	0.864	0.499	-0.012	7.783	10.953	0.220	0.746	0.249	0.000
Juil	0.842	0.531	0.082	7.391	12.406	9.632	0.708	0.282	0.007
Août	0.899	0.430	0.062	8.427	8.120	5.409	0.807	0.185	0.004
Sept	0.974	0.208	0.041	9.901	1.902	2.430	0.949	0.043	0.002
Oct	0.980	-0.170	0.074	10.026	1.276	7.860	0.961	0.029	0.006
Nov	0.904	-0.414	0.085	8.524	7.527	10.340	0.817	0.171	0.007
Dec	0.774	-0.624	0.056	6.258	17.121	4.524	0.600	0.390	0.003

CHAPITRE 2

ANALYSE FACTORIELLE DES CORRESPONDANCES (AFC)

Sommaire

2.1	Données et notations	47
2.2	Les nuages et leur ajustement	60
2.3	Aides à l'interprétation	74
2.4	Éléments supplémentaires	80

L'analyse factorielle des correspondances (AFC) a été proposée par [B⁺73]. Elle permet d'étudier la liaison (dite encore correspondances) entre deux variables qualitatives et détermine les modalités des variables qui interviennent dans la liaison ainsi que la manière dont elles interviennent (attraction répulsion, indépendance etc.). L'AFC est adaptée au traitement des données se présentant sous la forme d'un tableau de contingence (tableau de dépendance ou tableau croisé) ou d'un tableau formé d'individu décrits par deux caractères qualitatifs. La notion cruciale en analyse factorielle des correspondances est celle de la liaison entre deux variables qualitatives. Étudier la liaison entre deux variables qualitatives, c'est examiner l'écart entre les données et une situation d'indépendance.

L'intérêt de l'analyse factorielle des correspondances réside dans la symétrie qu'elle fait jouer aux variables ; dans le principe d'équivalence distributionnelle qui rend la méthode peut sensible au découpage adopté pour décrire les caractères X et Y , si ces dernières sont quantitatives et ont été découpées en classes ; dans le fait qu'elle permet grâce au découpage en classes des variables quantitatives d'analyser n'importe quel lot de variables hétérogènes ; dans le fait due raisonnant sur des profils, elle supprime l'effet taille en tant que facteur d'intensité, mais cet effet peut subsister en tant que facteur de forme (effet Gutmann par exemple).

2.1 Données et notations

Les données sur lesquelles on travaille sont constituées par n individus pour lesquels on dispose de deux variables qualitatives. Ces données sont regroupées dans un tableau dit de contingence dans lequel on trouve en lignes, les modalités de la variables V1 et en colonnes, les modalités de

la variable V2. A l'intersection de la ligne i avec la colonne j , on trouve x_{ij} , le nombre d'individus ayant choisis ou possédant à la fois la modalité i et V1 et la modalité j de V2. Ce tableau de contingence est aussi appelé tableau croisé dans la terminologie des enquêtes.

V1 \ V2	1	\dots	j	\dots	J
i	\dots	\dots	x_{ij}	\dots	\dots
	\vdots	\vdots	\vdots	\vdots	\vdots

TABLE 2.1 – Tableau de contingence

x_{ij} représente le nombre d'individus appartenant à l'élément i de l'ensemble I et à l'élément j de l'ensemble J. on a : $\sum_{i \in I} \sum_{j \in J} x_{ij} = n$.

Un tableau de contingence est généralement obtenu à partir du croisement de deux variables. Donc initialement les données sur lesquelles on travaille sont constituées par n individus pour lesquels on dispose de deux variables qualitatives.

Individus	V1 V2	
	1	
	i	j
l		
n		

TABLE 2.2 – Notation du tableau initial

Dans ce tableau, l'individu l possède la modalité i de la variable V1 et la modalité j de la variable V2. on construit alors un tableau de contingence ou tableau croisé en mettant en regard les modalités de la première variable V1 en lignes par exemple et les modalités de la seconde variable V2 en colonnes. Ce tableau contient donc al distribution des n individus dans les $I \times J$ cases du tableau.

Les données qui vont nous servir à illustrer notre présentation de l'analyse des correspondances sont en quelque sorte des données historiques. Il s'agit d'une ancienne enquête réalisée dans le cadre du CDEROC par Nicole Tabard¹ et publiée en 1974 repris par [HLP16]. Le tableau suivant est une reproduction du tableau publié.

1. Tabard N. (1974). Besoins et aspirations des familles et des jeunes. CREDOC. Paris.

	rester au foyer	trav. à mi-temps	trav. plein temps
2 conj. tr. également	13	142	106
trav. mari + absorbant	30	408	117
seul le mari trav.	241	573	94

TABLE 2.3 – Données d'enquête

```
# Chargement des données
```

```
import pandas as pd
```

```
donnée = pd.read_csv('femme_travail.csv', delimiter=";", encoding = "cp1252",
index_col = 0,usecols=[0,1,2,3])
```

Dans cette enquête, 1724 femmes ont été interrogées et deux questions ont été sélectionnées relatives à l'égard du travail féminin.

1. **Première question :** Quel est pour vous la famille idéale ? avec 3 modalités de réponse :

- Les deux conjoints travaillent également ;
- Le mari a un métier plus absorbant que celui de sa femme ;
- Seul le mari travaille.

2. **Deuxième question :** Quel est l'activité qui convient le mieux à une mère de famille quand les enfants vont à l'école ? Avec toujours trois modalités :

- Rester au foyer ;
- Travailler à mi-temps ;
- Travailler à plein temps ;

Les 3 modalités liées à la première question représentent le nombre de lignes de notre jeu de données et les 3 modalités liées à la deuxième question, le nombre de colonnes. On stocke cela dans deux variables n et p.

```
# Dimensions du tableau
```

```
n = donnée.shape[0]      # Nombre de lignes
p = donnée.shape[1]      # Nombre de colonnes
print(f'n = {n} et p = {p}.!')
```

```
n = 3 et p = 3.
```

Complétons le tableau (2.3) par les totaux sur les lignes et les colonnes. Pour ne pas modifier la structure de notre tableau, nous faisons une copie de ce dernier et effectuons les calculs dessus.

```
# copie des données
```

```
df = donnée.copy()
df.loc['Somme'] = df.sum(axis=0)
df.loc[:, 'Somme'] = df.sum(axis=1)
```

```
# Calcul des totaux en ligne
```

```
rowsum = donnée.sum(axis =1)
```

```
# Calcul des totaux en colonne
colsum = donnee.sum(axis=0)
# Affichage
print(df)
```

On obtient le tableau définitif suivant :

	rester au foyer	trav. à mi-temps	trav. plein temps	Somme
2 conj. tr. également	13	142	106	261
trav. mari + absorbant	30	408	117	555
seul le mari trav.	241	573	94	908
Somme	284	1123	317	1724

TABLE 2.4 – Les effectifs observés

Si on regarde les résultats bruts, on s'aperçoit qu'il y a 261 femmes qui ont répondu, pour la première question, *les deux conjoints travaillent également*, mais 908 qui ont choisi *seul le mari travaille*. On peut dire qu'à l'aune de cette question, à cette époque là, une majorité des femmes étaient hostiles au travail féminin.

A la deuxième question, *travailler à mi-temps* à obtenu 1123 suffrages. C'est donc une réponse largement majoritaire. Au regard de cette question, on peut avoir l'impression qu'à cette époque là les femmes étaient favorables au travail féminin, pas à temps plein certes, mais favorables tout de même.

Ces deux questions semblent donc apporter des réponses un peu contradictoires. Donc on se pose la question suivante : Quel lien y a-t-il entre ces deux questions ?

A partir du tableau de contingence, on va calculer le tableau des probabilités correspondant.

2.1.1 Du tableau de contingence au tableau de probabilités

A partir du tableau de contingence précédent (tableau 2.3), on va calculer le tableau des probabilités correspondant. L'analyse des correspondances va en fait travailler sur le tableau des probabilités (ou de fréquence relation). Pour obtenir une probabilité f_{ij} , la proportion des individus possédant à la fois les modalités i et j , on divise l'effectif x_{ij} par l'effectif total n , on a :

$$f_{ij} = \frac{x_{ij}}{n} = \mathbb{P}(V1 = i, V2 = j) \quad (2.1)$$

```
# Probabilités conjointes
probconj=donnee.apply(lambda x : x/donnee.sum().sum())
print(probconj)
```

	rester au foyer	trav. à mi-temps	trav. plein temps
2 conj. tr. également	0.007541	0.082367	0.061485
trav. mari + absorbant	0.017401	0.236659	0.067865
seul le mari trav.	0.139791	0.332367	0.054524

TABLE 2.5 – Probabilités conjointes

On obtient bien ainsi la probabilité conjointe de posséder à la fois la modalité i de V1 et la modalité j de V2. C'est ce terme que l'on met dans le tableau. On a : $\sum_{i \in I} \sum_{j \in J} f_{ij} = 1$ (i.e la somme des probabilités vaut 1).

```
# Somme
print(f'Somme des probabilités : {probconj.sum().sum()}')

Somme des probabilités : 1.0
```

On complète le tableau par les probabilités marginales en colonnes ou fréquences marginales de la modalité j de V2 $\left(f_{\bullet j} = \sum_{i \in I} f_{ij} \right)$

```
# Marge colonne (probabilité marginale)
rowprob = probconj.sum(axis = 1)
print(rowprob)
```

	2 conj. tr. également	trav. mari + absorbant	seul le mari trav.
Somme	0.151392	0.321926	0.526682

TABLE 2.6 – Marges colonnes

... et les probabilités marginales en lignes (ou fréquence marginale de la modalité i de V1) $\left(f_{i \bullet} = \sum_{j \in J} f_{ij} \right)$

```
# Marge ligne (probabilité marginale)
colprob = probconj.sum(axis = 0)
print(colprob)
```

	rester au foyer	trav. à mi-temps	trav. plein temps
Somme	0.164733	0.651392	0.183875

TABLE 2.7 – Marges lignes

avec $\sum_{i \in I} f_{i \cdot} = \sum_{j \in J} f_{\cdot j} = 1$.

```
# copie des données
df2 = probconj.copy()
df2.loc['Somme'] = df2.sum(axis=0)
df2.loc[:, 'Somme'] = df2.sum(axis=1)
# Affichage
df2
```

	rester au foyer	trav. à mi-temps	trav. plein temps	Somme
2 conj. tr. également	0.007541	0.082367	0.061485	0.151392
trav. mari + absorbant	0.017401	0.236659	0.067865	0.321926
seul le mari trav.	0.139791	0.332367	0.054524	0.526682
Somme	0.164733	0.651392	0.183875	1.000000

TABLE 2.8 – Probabilités conjointes observées

Notre problème est d'étudier la liaison entre V1 et V2, c'est-à-dire l'écart entre les données observées et une situation ou le modèle d'indépendance.

2.1.2 Indépendance entre variables qualitatives

Donnons quelques précisions sur l'indépendance entre 2 variables qualitatives et comment l'analyse des correspondances appréhende *l'écart à l'indépendance*.

1) Modèle d'indépendance

a) Événements indépendants

Soient A et B deux événements. On dit que A et B sont indépendants si et seulement si :

$$\mathbb{P}(A \cap B) = \mathbb{P}(A) \times \mathbb{P}(B) \quad (2.2)$$

Autrement, deux événements sont indépendants si la probabilité de leur intersection est égale au produit des probabilités.

b) Variables aléatoires indépendantes

$$\forall i, \forall j, \quad f_{ij} = f_{i\bullet} \times f_{\bullet j} \quad (2.3)$$

Autrement dit, la probabilité conjointe est égale au produit des probabilités marginales.

Autres écritures :

- $\frac{f_{ij}}{f_{i\bullet}} = f_{\bullet j}$

- $\frac{f_{ij}}{f_{\bullet j}} = f_{i\bullet}$

c'est-à-dire **probabilité conditionnelle = probabilité marginale**

2) Liaisons entre deux variables qualitatives

La liaison entre deux variables qualitatives, c'est l'écart entre les données observées (f_{ij}) d'une part et le modèle d'indépendance d'autre part ($f_{i\bullet}f_{\bullet j}$). Quant on étudie la liaison entre deux variables qualitatives, on réalise généralement en premier lieu un test de significativité de la liaison en mettant en oeuvre un test de khi-deux.

3) Significativité de la liaison (de l'écart) : test du χ^2

Les étapes du test :

Étape 1 :

On calcule d'abord les effectifs théoriques y_{ij} suivant la formule du modèle d'indépendance :

$$y_{ij} = \frac{n_{i\bullet} \times n_{\bullet j}}{n} \quad \forall i, j \quad (2.4)$$

```
# Effectif total
neff = donnee.values.sum()
# Effectif théorique
import numpy as np
efftheorik = pd.DataFrame(np.zeros((n,p), dtype=float), index = donnee.index,
                           columns = donnee.columns)
for i in range(n):
    for j in range(p):
        efftheorik.values[i,j] = (rowsum[i]*colsum[j])/neff
# copie des données
df3 = efftheorik.copy()
df3.loc['Somme'] = df3.sum(axis=0)
df3.loc[:, 'Somme'] = df3.sum(axis=1)
# Affichage
df3.round(2)
```

	rester au foyer	trav. à mi-temps	trav. plein temps	Somme
2 conj. tr. également	43.00	170.01	47.99	261.0
trav. mari + absorbant	91.43	361.52	102.05	555.0
seul le mari trav.	149.58	591.46	166.96	908.0
Somme	284.00	1123.00	317.00	1724.0

TABLE 2.9 – Effectifs théoriques

Étape 2 :

On calcule la statistique de khi-deux :

$$\begin{aligned} \chi_{obs}^2 &= \sum_{i \in I} \sum_{j \in J} \frac{(\text{effectifs observés} - \text{effectifs théoriques})^2}{\text{effectifs théoriques}} \\ &= \sum_{i \in I} \sum_{j \in J} \frac{(x_{ij} - y_{ij})^2}{y_{ij}} = \sum_{i \in I} \sum_{j \in J} \frac{(nf_{ij} - n f_{i\bullet} f_{\bullet j})^2}{n f_{i\bullet} f_{\bullet j}} \\ &= \sum_{i \in I} \sum_{j \in J} n \frac{(\text{probabilités observées} - \text{probabilités théoriques})^2}{\text{probabilités théoriques}} \\ &= n\phi^2 \quad \text{où} \quad \left(\phi^2 = \sum_{i \in I} \sum_{j \in J} \frac{(\text{probabilités observées} - \text{probabilités théoriques})^2}{\text{probabilités théoriques}} \right) \end{aligned}$$

ϕ^2 confronte les probabilités observées aux probabilités théoriques. C'est donc un **indicateur d'intensité de la liaison**. Cette statistique suit une χ^2 à $(I - 1) \times (J - 1)$ degrés de libertés. On rejette l'hypothèse nulle si la pvalue est inférieure au seuil fixé (généralement 5%).

```

# Statistique du Khi-deux
khi2 = np.sum(((donnee.values - efftheorik.values)**2)/efftheorik.values)
print("Khi2 vaut %.2f" %(khi2))

Khi2 vaut 233.43.

```

On calcule le nombre de degré de libertés et la pvalue associée.

```

# Degré de liberté
ddl = (n - 1)*(p - 1)
print("degré de liberté : %.i" %(ddl))
# P-value du test
from scipy import stats as stat
pvalue = 1 - stat.chi2.cdf(khi2,df=ddl)
print('pvalue : %.4f' %(pvalue))

degré de liberté : 4
pvalue : 0.0000

```

Le degré de liberté est 4. Le test conduit au rejet de l'hypothèse nulle (p-value ≈ 0).

Remarque

Un façon assez simple est d'utiliser la fonction `.chi2_contingency` fournit par la librairie `scipy`. Cette fonction retourne 4 éléments dont les 3 premiers sont respectivement la statistique du χ^2 , le pvalue et le nombre de degré de liberté. Le 4^{ème} élément correspond aux effectifs théoriques.

```

# Autre façon
from scipy import stats

scipy_stats = stats.chi2_contingency(donnee)
khistats = pd.DataFrame(scipy_stats[:3],columns=['valeur'],
                         index = ['chisq','p.value','ddl'])
print(khistats.round(2))

```

	chisq	p.value	ddl
valeur	233.43	0.00	4

TABLE 2.10 – Test d'indépendance

2.1.3 Comment l'AFC appréhende l'écart à l'indépendance ?

On va d'abord adopter le point de vue des lignes.

1) Analyse par lignes

Dans cette analyse on se réfère au modèle d'indépendance :

$$\forall i, j \quad \frac{f_{ij}}{f_{i\bullet}} = f_{\bullet j} \quad (2.5)$$

On va diviser chaque élément d'une ligne par sa marge (ou son total). On obtient ainsi ce qu'on appelle un profil ligne i qui n'est rien d'autre qu'une distribution conditionnelle.

V1 \ V2	1	\dots	j	\dots	J	Somme
1	$\frac{f_{11}}{f_{1\bullet}}$		$\frac{f_{1j}}{f_{1\bullet}}$		$\frac{f_{1J}}{f_{1\bullet}}$	1
\vdots						
i	$\frac{f_{i1}}{f_{i\bullet}}$		$\frac{f_{ij}}{f_{i\bullet}}$		$\frac{f_{iJ}}{f_{i\bullet}}$	1
\vdots						
I	$\frac{f_{I1}}{f_{I\bullet}}$		$\frac{f_{IJ}}{f_{I\bullet}}$		$\frac{f_{IJ}}{f_{I\bullet}}$	1

TABLE 2.11 – Distribution conditionnelle de $X|Y$

La quantité $f_{ij}/f_{i\bullet}$ est la probabilité conditionnelle de posséder la modalité j de la variable V2 lorsque l'on possède la modalité i de la variable V1. C'est donc un estimateur empirique de $\mathbb{P}(V2 = j|V1 = i)$. Il y a donc indépendance, lorsque, pour toutes les cases, la probabilité conditionnelle est égale à la probabilité marginale. Il s'agit en effet d'analyser les proportions en ligne du tableau de contingence. Ce point de vue est proche de l'intuition : il y a indépendance si la probabilité de posséder j de la variable V2 ne dépend pas de la modalité possédée par la variable V1. En se basant sur les données du tableau (2.3), les profils lignes sont calculés sous la forme du ratio suivant :

$$\mathbb{P}(V2 = j|V1 = i) = \frac{x_{ij}}{n_{i\bullet}}, \quad \text{avec } n_{i\bullet} = \sum_{j=1}^J x_{ij}. \quad (2.6)$$

```
# Profils lignes
import numpy as np
rowprof = donnee.apply(lambda x:x/np.sum(x), axis=1)
# Copie des données
df4 = rowprof.copy()
df4['Profil moyen'] = df4.sum(axis=1)
print(df4.round(3))
```

	rester au foyer	trav. à mi-temps	trav. plein temps	Profil moyen
2 conj. tr. également	0.050	0.544	0.406	1
trav. mari + absorbant	0.054	0.735	0.211	1
seul le mari trav.	0.265	0.631	0.104	1

TABLE 2.12 – Profils lignes : distribution conditionnelle de $V1|V2$

Dans cette ligne, on a la distribution des réponses pour la variable V2 sachant que l'on possède la modalité i de la variable V1. Représentons graphiquement cela.

```
# Graphique
from matplotlib import pyplot as plt
rowprof.plot.bah(stacked=True, figsize=(16,8))
plt.ylabel('Modalités V1')
plt.xlabel('Distribution conditionnelle')
plt.title('Profils lignes')
plt.show()
```

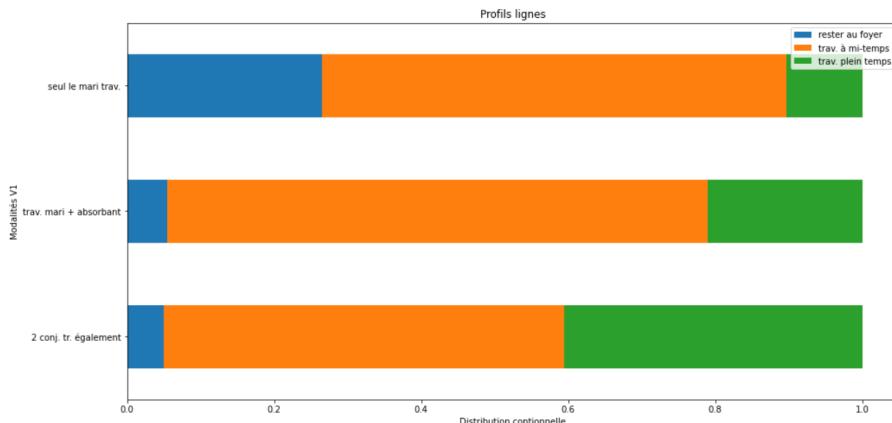


FIGURE 2.1 – Profils lignes

Ces profils lignes, on va le comparer au profit ligne moyen qui est la distribution marginale de la variable V1 (profil de l'ensemble des individus étudiés).

```
# Profil ligne moyen
rowmean = colsum/colsum.sum()
# Copie des données
df5 = rowprof.copy()
df5.loc['Profil moyen',:] = rowmean
df5.loc[:, 'Profil moyen'] = df5.sum(axis=1)
print(df5.round(3))
```

	rester au foyer	trav. à mi-temps	trav. plein temps	Profil moyen
2 conj. tr. également	0.050	0.544	0.406	1
trav. mari + absorbant	0.054	0.735	0.211	1
seul le mari trav.	0.265	0.631	0.104	1
Profil moyen	0.165	0.651	0.184	1

TABLE 2.13 – Profil ligne moyen

Ajoutons cette information au graphique précédent.

```
# Graphique
df5.iloc[:,[0,1,2]].plot.barh(stacked=True, figsize=(16,8))
plt.ylabel('Modalités V1')
plt.xlabel('Distribution conditionnelle')
plt.title('Profils lignes')
plt.show()
```

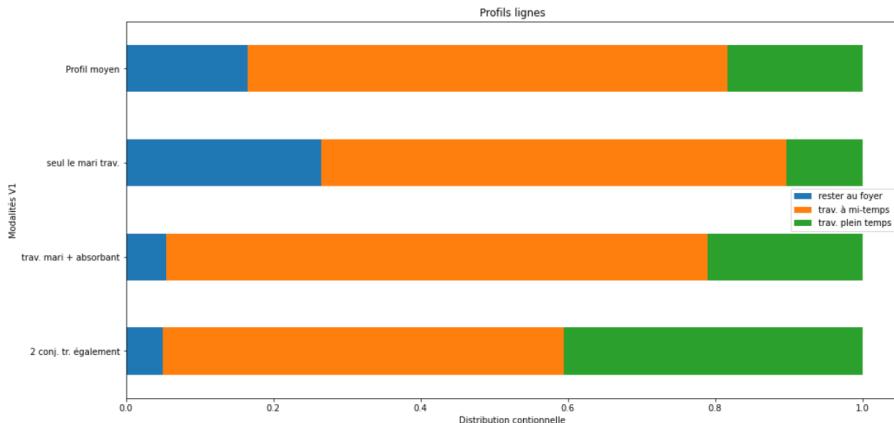


FIGURE 2.2 – Profils lignes complétés du profil ligne moyen

Pour comparer la distribution conditionnelle à la distribution marginale, l’analyse des correspondances va comparer les profils lignes au profil moyen. Il s’agit bien d’une approche multidimensionnelle de l’écart à l’indépendance parce que l’on considère simultanément l’ensemble des profils.

2) Analyse par colonnes

Nous allons à présent étudier le même tableau mais en procédant par une analyse en colonnes. Lorsqu’on réalise une analyse par colonnes, on se réfère au modèle d’indépendance suivant :

$$\frac{f_{ij}}{f_{\bullet j}} = f_{i\bullet} \quad (2.7)$$

On construit le tableau des profils colonnes. On va diviser chaque élément d’une colonne par sa marge (ou son total). On obtient ainsi ce qu’on appelle un profil colonne j qui n’est rien d’autre qu’une distribution conditionnelle.

V1 \ V2	1	\dots	j	\dots	J
1	$\frac{f_{11}}{f_{\bullet 1}}$		$\frac{f_{1j}}{f_{\bullet j}}$		$\frac{f_{1J}}{f_{\bullet J}}$
\vdots					
i	$\frac{f_{i1}}{f_{\bullet 1}}$		$\frac{f_{ij}}{f_{\bullet j}}$		$\frac{f_{iJ}}{f_{\bullet J}}$
\vdots					
I	$\frac{f_{I1}}{f_{\bullet 1}}$		$\frac{f_{IJ}}{f_{\bullet j}}$		$\frac{f_{IJ}}{f_{\bullet J}}$
Somme	1		1		1

TABLE 2.14 – Distribution conditionnelle de $Y|X$

La quantité $f_{ij}/f_{\bullet j}$ est la probabilité conditionnelle de posséder la modalité i de la variables V1 lorsque l'on possède la modalité j de la variable V2. Il s'agit en effet d'analyser les proportions en colonnes du tableau de contingence. Les profils colonnes sont calculés sous la forme du ratio suivant :

$$\mathbb{P}(V2 = j|V1 = i) = \frac{x_{ij}}{n_{\bullet j}}, \quad \text{avec } n_{\bullet j} = \sum_{i \in I} x_{ij} \quad (2.8)$$

```
## Profils colonnes
import numpy as np
colprof = donnee.apply(lambda x:x/np.sum(x), axis=0)
somme = pd.DataFrame(colprof.sum(axis = 0).values.reshape(1,p),
                      index = ["Somme"],
                      columns = donnee.columns)
colprofdf = pd.concat([colprof, somme], axis = 0)
print(colprofdf.round(3))
```

	rester au foyer	trav. à mi-temps	trav. plein temps
2 conj. tr. également	0.046	0.126	0.334
trav. mari + absorbant	0.106	0.363	0.369
seul le mari trav.	0.849	0.510	0.297
Somme	1.000	1.000	1.000

TABLE 2.15 – Profils colonnes

```
# Graphique
colprof.transpose().plot(kind = "bar", stacked=True, figsize=(16,8))
plt.ylabel('Distribution conditionnelle', fontsize=12)
plt.xlabel('Modalités V2', fontsize=12)
plt.title('Profils colonnes', fontsize=12)
plt.show()
```

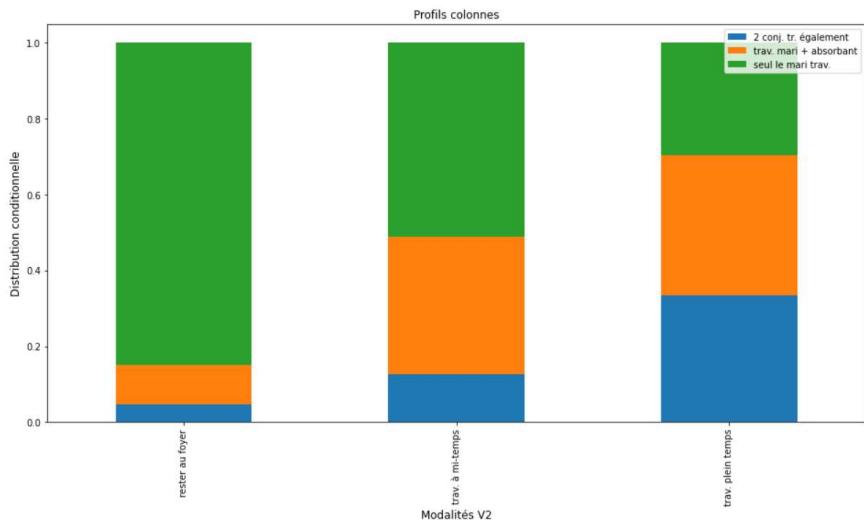


FIGURE 2.3 – Profils colonnes

On compare également les profils colonnes au profil moyen.

```
# Profil ligne moyen
colmean = rowsum/rowsum.sum()
colmeandf = pd.DataFrame(colmean.values.reshape(n,1),
                         columns = ["Profil moyen"], index = donnee.index)
coldata = pd.concat([colprof, colmeandf], axis = 1)
coldata.loc['Somme',:] = coldata.sum(axis=0)
coldata.round(3)
```

	rester au foyer	trav. à mi-temps	trav. plein temps	Profil moyen
2 conj. tr. également	0.046	0.126	0.334	0.151
trav. mari + absorbant	0.106	0.363	0.369	0.322
seul le mari trav.	0.849	0.510	0.297	0.527
Somme	1.000	1.000	1.000	1.000

TABLE 2.16 – Profil colonne moyen

On rajoute cette information au graphique précédent.

```
# Graphique
coldata.transpose().iloc[:,[0,1,2]].plot(kind = "bar", stacked=True, figsize=(16,8))
plt.ylabel('Distribution conditionnelle', fontsize=12)
plt.xlabel('Modalités V2', fontsize=12)
plt.title('Profils colonnes', fontsize=12)
plt.show()
```

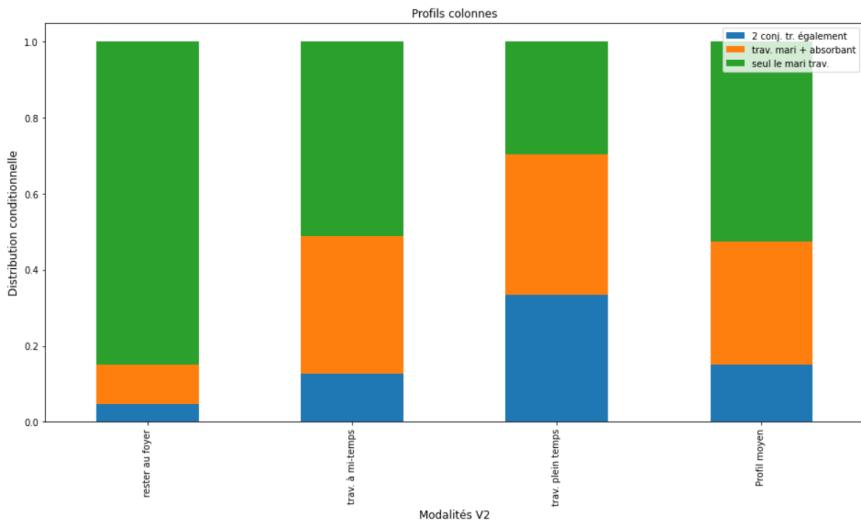


FIGURE 2.4 – Profils colonnes complétés du profil colonne moyen

Le modèle d’indépendance stipule donc que les profils lignes d’une part et les profils colonnes d’autre part sont égaux au profil moyen correspondant.

2.2 Les nuages et leur ajustement

2.2.1 Nuage des profils lignes

1) Distance entre deux profils lignes

Dans l’espace R^I des profils lignes, la distance utilisée en AFC est la distance de khi-deux. Elle ressemble beaucoup à la distance euclidienne usuelle. En effet, on va retrouver une somme des carrés des écarts. En AFC, l’écart entre le profil k et le profil l est mesurée par :

$$d_{\chi^2}^2(k, l) = \sum_{j \in J} \frac{1}{f_{\bullet j}} \left(\frac{f_{kj}}{f_{k \bullet}} - \frac{f_{lj}}{f_{l \bullet}} \right)^2 \quad (2.9)$$

La différence avec la distance euclidienne usuelle est que chaque modalité j de V2 a un poids égal à $1/f_{\bullet j}$.

```
# Distance entre paires de modalités lignes
rowdist = np.zeros(shape = (rowprof.shape[0], rowprof.shape[0]))
value = rowprof.values
for i in range(rowprof.shape[0]-1):
    for j in range(i+1, rowprof.shape[0]):
        rowdist[i,j] = np.sum((value[i,:]-value[j,:])**2/rowmean.values)
# Affichage
rowdistdf = pd.DataFrame(rowdist, index = data.index,
                           columns = data.index)
print(rowdistdf)
```

	trav. mari + absorbant	seul le mari trav.
2 conj. tr. également	0.264	0.792
trav. mari + absorbant		0.350

TABLE 2.17 – Distances entre profils lignes

Nous visualisons cette matrice à l'aide d'un heatmap.

```
# Graphique
import seaborn as sns
plt.figure(figsize = (8,6))
sns.heatmap(rowdistdf, vmin = 0, vmax = np.max(rowdist),
            linewidth = 0.1, cmap = "Blues", xticklabels = donnee.index,
            yticklabels = donnee.index)
plt.title('Distances entre profils lignes', fontsize=12)
plt.xlabel('Modalités V1', fontsize=12)
plt.ylabel('Modalités V1', fontsize=12)
plt.show()
```

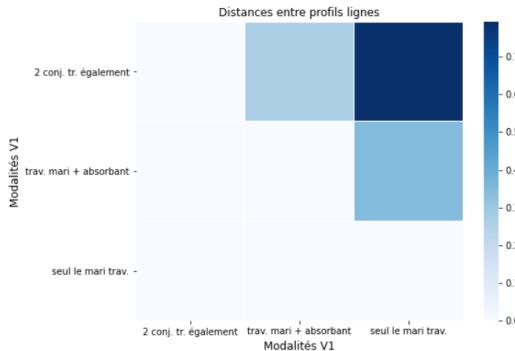


FIGURE 2.5 – Représentation des distances entre profils lignes

2) Distance entre un profil ligne et le profil moyen

Dans l'espace R^I des profils lignes, le centre de gravité correspond au profil moyen. Ce dernier est noté G_I et a pour coordonnées $f_{\bullet j}$. Donc la distance entre le profil i et le profil moyen G_I est :

$$d_{\chi^2}^2(i, G_I) = \sum_{j \in J} \frac{1}{f_{\bullet j}} \left(\frac{f_{ij}}{f_{i\bullet}} - f_{\bullet j} \right)^2 \quad (2.10)$$

```
# Distance à l'origine
rowdisto = rowprof.apply(lambda x : np.sum((x - rowmean.values)**2 / rowmean.values),
                         axis = 1)
print(rowdisto.round(3))
```

	2 conj. tr. également	trav. mari + absorbant	seul le mari trav.
Disto ²	0.367	0.089	0.097

TABLE 2.18 – Distances entre profils lignes et le profil moyen

L'inertie du point i par rapport à G_I traduit la quantité d'information portée par la modalité i . Elle est définie par le produit entre le poids de la modalité et sa distance à l'origine.

$$\begin{aligned} \text{Inertie}(i/G_I) &= P(i) \times d_{\chi^2}^2(i, G_I) = f_{i\bullet} \times d_{\chi^2}^2(i, G_I) \\ &= f_{i\bullet} \sum_{j \in J} \frac{1}{f_{\bullet j}} \left(\frac{f_{ij}}{f_{i\bullet}} - f_{\bullet j} \right)^2 = \sum_{j \in J} \frac{(f_{ij} - f_{i\bullet} f_{\bullet j})^2}{f_{i\bullet} f_{\bullet j}} \end{aligned}$$

Au coefficient n près, on reconnaît la contribution de la ligne i au χ^2 , d'où le nom de distance du χ^2 .

```
# Poids des lignes
rowweight = rowsum.values/rowsum.values.sum()
# Inertie des lignes
rowinertie = rowdisto*rowweight
# Affichage
rowinfos = pd.DataFrame(np.transpose([rowdisto, rowweight, rowinertie]),
                         columns = ["Disto2", "Weight", "Inertie"],
                         index = donnee.index)
print(rowinfos.round(3))
```

	Disto ²	Weight	Inertie(i/G _I)
2 conj. tr. également	0.3665	0.1514	0.0555
trav. mari + absorbant	0.0891	0.3219	0.0287
seul le mari trav.	0.0973	0.5267	0.0512

TABLE 2.19 – Décomposition des profils lignes

On voit que l'inertie du nuage N_I par rapport à G_I est égale (au coefficient n près) au critère χ^2 . Ainsi, examiner la dispersion de N_I autour de G_I revient bien à examiner l'écart entre les données et le modèle d'indépendance.

3) Inertie totale

L'inertie totale représente la quantité d'information disponible dans les données. C'est la somme des inerties de ses points.

$$I_{nI} = \sum_{i \in I} \text{Inertie}(i/G_I) \quad (2.11)$$

```
# Inertie total
inertietot = np.sum(rowinertie)
print("Inertie totale : %.4f" %(inertietot))

Inertie totale : 0.1354
```

2.2.2 Nuage des profils colonnes

De même qu'on a construit un nuage des profils lignes, on va construire un nuage des profils colonnes

1) Distance entre deux profils colonnes

Dans l'espace R^J des profils colonnes, la distance (au carré) entre le profil j et le profil k est mesurée par :

$$d_{\chi^2}^2(j,k) = \sum_{i \in I} \frac{1}{f_{i\bullet}} \left(\frac{f_{ij}}{f_{\bullet j}} - \frac{f_{ik}}{f_{\bullet k}} \right)^2 \quad (2.12)$$

Ici, chaque modalité i a un poids qui est $1/f_{i\bullet}$.

```
# Distance entre paires de modalités lignes
coldist = np.zeros(shape = (colprof.shape[0], colprof.shape[0]))
value = colprof.values.transpose()
for i in range(colprof.shape[0]-1):
    for j in range(i+1, colprof.shape[0]):
        coldist[i,j] = np.sum((value[i,:]-value[j,:])**2/colmean.values)
# Affichage
coldistdf = pd.DataFrame(coldist, index = donnee.columns,
                           columns = donnee.columns)
print(coldistdf.round(4))
```

	trav. à mi-temps	trav. plein temps
rester au foyer	0.4666	1.3445
trav. à mi-temps		0.3724

TABLE 2.20 – Distances entre profils colonnes

Comme pour la matrice des distances entre profils lignes, nous visualisons celle des profils colonnes.

```
# Graphique
plt.figure(figsize = (9,6))
sns.heatmap(coldistdf, vmin = 0, vmax = np.max(rowdist),
            linewidth = 0.1, cmap = "Blues", xticklabels = donnee.columns,
            yticklabels = donnee.columns)
plt.title('Distances entre profils colonnes', fontsize=12)
plt.xlabel('Modalités V2', fontsize=12)
plt.ylabel('Modalités V2', fontsize=12)
plt.show()
```

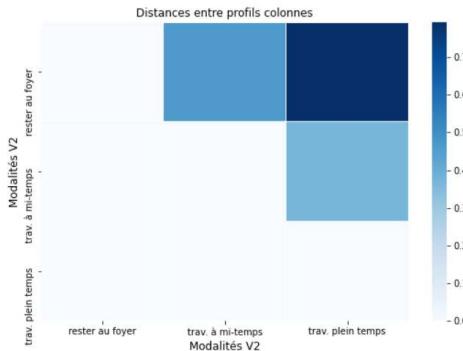


FIGURE 2.6 – Heatmap des distances entre profils colonnes

2) Distance entre un profil colonne et le profil moyen

Dans l'espace R^J des profils colonnes, le centre de gravité correspond au profil moyen. Ce dernier est noté G_J et a pour coordonnées $f_{i\bullet}$. Donc la distance (au carré) entre le profil j et le profil moyen G_J est :

$$d_{\chi^2}^2(j, G_J) = \sum_{i \in I} \frac{1}{f_{i\bullet}} \left(\frac{f_{ij}}{f_{\bullet j}} - f_{i\bullet} \right)^2 \quad (2.13)$$

```
# Distance à l'origine
mean = colmean.values
coldisto = colprof.transpose().apply(lambda x : np.sum((x-mean)**2/mean),
                                         axis = 1)
print(coldisto.round(4))
```

	rester au foyer	trav. à mi-temps	trav. plein temps
Disto ²	0.4158	0.0099	0.3287

TABLE 2.21 – Distance entre profils colonnes et profil colonne moyen

L'inertie du profil j par rapport au point G_J s'écrit :

$$\begin{aligned} \text{Inertie}(j/G_J) &= f_{\bullet j} \times d_{\chi^2}^2(j, G_J) \\ &= f_{\bullet j} \sum_{i \in I} \frac{1}{f_{i\bullet}} \left(\frac{f_{ij}}{f_{\bullet j}} - f_{i\bullet} \right)^2 = \sum_{j \in J} \frac{(f_{ij} - f_{i\bullet} f_{\bullet j})^2}{f_{i\bullet} f_{\bullet j}} \end{aligned}$$

```
# Poids des lignes
colweight = colsum.values/colsum.values.sum()
# Inertie des lignes
colinertie = coldisto*colweight
# Affichage
colinfos = pd.DataFrame(np.transpose([coldisto, colweight, colinertie]),
                           columns = ["Disto2", "poids", "Inertie"],
                           index = donnee.columns)
print(colinfos.round(4))
```

	Disto ²	poids	Inertie(j/G _J)
rester au foyer	0.4158	0.1647	0.0685
trav. à mi-temps	0.0099	0.6514	0.0065
trav. plein temps	0.3287	0.1839	0.0604

TABLE 2.22 – Décomposition des profils colonnes

On vérifie si l'inertie totale est la même.

```
# Inertie total
inertietot = np.sum(colinertie)
print("Inertie totale : %.4f" %(inertietot))

Inertie totale : 0.1354
```

L'inertie totale de N_J est donc la même que celle de $N_I (= \frac{1}{n} \chi^2)$: étudier la dispersion de N_J autour de G_J revient à étudier la liaison entre les deux variables V1 et V2.

3) Que se passe-t-il s'il y a indépendance ?

S'il y a indépendance, on a $\forall i, \frac{f_{ij}}{f_{i\bullet}} = f_{\bullet j}$. C'est-à-dire que dans chacun des nuages *tous les profils sont confondus au profil moyen*. Autrement dit, le nuage N_I dans ce cas est réduit à un seul point, l'origine des axes G_I . L'inertie du nuage est nulle.

Le résultat est identique pour les colonnes $\forall j, \frac{f_{ij}}{f_{\bullet j}} = f_{i\bullet}$. Et donc, plus les données s'écartent de l'indépendance et plus les profils s'écartent de l'origine. Le modèle d'indépendance stipule donc que les profils lignes d'une part et les profils colonnes d'autre part sont égaux au profil moyen correspondant.

2.2.3 Pourcentage d'inertie et inertie en AFC

On montre que le cœur de l'AFC est une diagonalisation de la matrice dont les valeurs propres sont les inerties projetées. Étant des inerties, ces valeurs propres sont positives et sont triées par ordre décroissant (le premier axe correspondant à l'inertie projetée maximum). Soit la matrice M définie par

$$M = \frac{1}{\sqrt{n}} R \quad (2.14)$$

où R est la matrice des résidus standardisés dans le contexte de l'AFC. R mesure l'écart entre les effectifs observés et les effectifs théoriques pondérés par la racine carré des effectifs théoriques. On a :

$$R = \frac{(\text{effectifs observés} - \text{effectifs théoriques})}{\sqrt{\text{effectifs théoriques}}} \quad (2.15)$$

```
# Matrice des résidus standardisés
resStd = (donnee.values - efftheorik)/np.sqrt(efftheorik)
print(resStd.round(4))
```

	rester au foyer	trav. à mi-temps	trav. plein temps
2 conj. tr. également	-4.5745	-2.1484	8.3736
trav. mari + absorbant	-6.4242	2.4444	1.4799
seul le mari trav.	7.4751	-0.7592	-5.6464

TABLE 2.23 – Matrice des résidus standardisés

Nous calculons ensuite la matrice M.

```
# Matrice M
M = resStd/np.sqrt(neff)
print(M.round(4))
```

	rester au foyer	trav. à mi-temps	trav. plein temps
2 conj. tr. également	-0.1102	-0.0517	0.2017
trav. mari + absorbant	-0.1547	0.0589	0.0356
seul le mari trav.	0.1800	-0.0183	-0.1360

TABLE 2.24 – Matrice M

La solution de l'AFC correspond à la décomposition en valeurs singulières de la matrice M :

$$M = U\Delta V^T \quad (2.16)$$

Où :

- La matrice U , de dimension (n, n) , contient les n-vecteurs propres singuliers à gauche. U est orthonormée. Ces vecteurs propres permettent d'obtenir les coordonnées factorielles des modalités lignes.
- $\Delta(n,p)$, une matrice diagonale dont les éléments correspondent aux valeurs singulières.
- $V(p,p)$ contient les vecteurs singuliers à droite. Tout comme U , V est également orthonormée.

Nous faisons appel à la fonction `np.linalg.svd()` de la librairie « Numpy » pour effectuer la décomposition en valeurs singulières.

```
# Décomposition en valeurs singulières (svd)
U,delta,V = np.linalg.svd(M)
```

Comme au chapitre sur l'ACP, rappelons que les valeurs singulières (δ_α) sont définies comme suit :

$$\begin{cases} Mv_\alpha = \delta_\alpha u_\alpha \\ M^T u_\alpha = \delta_\alpha v_\alpha \end{cases}$$

La variance restituée par les facteurs, les valeurs propres, correspondent au carré des valeurs singulières, avec ($\lambda_\alpha \leq 1$) :

$$\lambda_\alpha = \delta_\alpha^2 \quad (2.17)$$

Affichons les valeurs de δ .

```
# delta
print(delta)

[3.41818701e-01 1.36236740e-01 1.72095781e-17]
```

Nous constatons que seules les deux premières valeurs sont non-nulles. Ce qui est conforme à ce qui est attendu dans le cas d'une AFC (le nombre d'axe maximal est égal à $\min(I - 1, J - 1) = \min(3 - 1, 3 - 1) = 2$). Nous les passons ensuite au carré pour obtenir les valeurs propres.

```
# Valeurs singulières au carré
np.set_printoptions(precision=3, suppress = True)
eigenvalue = delta**2
print(eigenvalue)

[0.117 0.019 0.    ]
```

La dernière valeur est nulle. Par conséquent, on garde uniquement les 2 premières valeurs. Celles-ci correspondent à nos valeurs propres en AFC.

```
# Nombre d'axes maximales
fmax = min(n-1, p-1)
# Valeur propre
eigenvalue = eigen[:fmax]
percent = np.array([100*x/sum(eigenvalue) for x in eigenvalue])
cumpercent = np.cumsum(percent)
# dataframe
index = ['Dim.{}'.format(x+1) for x in range(fmax)]
columns = ['valeur propre','pourcentage d'inertie',
           'pourcentage d'inertie cumulée']
Eigen = pd.DataFrame(np.transpose([eigenvalue, percent, cumpercent]),
                      index=index, columns = columns)
Eigen.index.name = 'Dimension'
# Affichage
print(Eigen.round(3))
```

Dimension	valeur propre	pourcentage d'inertie	pourcentage d'inertie cumulée
Dim.1	0.117	86.292	86.292
Dim.2	0.019	13.708	100.000

TABLE 2.25 – Valeurs propres (=inerties projetées) de l'AFC du tableau (2.3)

On crée un fonction de visualisation.

```
# Visualisation des valeurs propres
def screeplot(data, choice=None, figsize=None):
    #
    p = data.shape[0]
    fig, axes = plt.subplots(figsize = figsize); axes.grid()
    axes.set_xlabel('Dimensions', fontsize=14)
    axes.set_title('Scree plot', fontsize=14)
```

```

axes.set_xticks([x for x in range(1,p+1)])
if choice is None or choice=='scree plot':
    eigen = data.iloc[:,0].round(3)
    ylim = np.max(eigen)+.05
    axes.set_ylim(0,ylim)
    axes.bar(np.arange(1,p+1),eigen.values,width=0.9)
    axes.plot(np.arange(1,p+1),eigen.values,c="black")
    axes.set_ylabel('Eigenvalue',fontsize=13)
    ## Add text
    for i in range(p):
        axes.scatter(i+1,eigen.values[i],color='black',alpha=1)
        axes.text(i+0.9,eigen.values[i],str(eigen.values[i]),
                  color = 'black',fontsize=11)
elif choice == "percentage":
    percent = data.iloc[:,1].round()
    axes.set_ylim(0,100)
    axes.bar(np.arange(1,p+1),percent.values,width=0.9)
    axes.plot(np.arange(1,p+1),percent.values,c="black")
    axes.set_ylabel('Percentage of variance',fontsize=13)
    ## Add text
    for i in range(p):
        axes.scatter(i+1,percent.values[i],color='black',alpha=1)
        axes.text(i+0.9,percent.values[i],f'{percent.values[i]}%',
                  color = 'black',fontweight='bold',fontsize=12)
elif choice == "cumulative":
    cumul = data.iloc[:,2].round()
    axes.set_ylim(0,105)
    axes.bar(np.arange(1,p+1),cumul.values,width=0.9)
    axes.set_ylabel('Cumulative percentage of variance',fontsize=13)
plt.show()
# Affichage
screeplot(data=Eigen,figsize=(8,6))

```

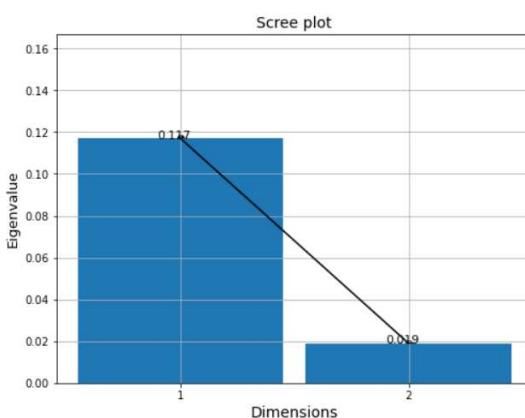


FIGURE 2.7 – Histogramme des valeurs propres

Khi-deux

On a vu qu'il existait une relation entre le khi-deux χ^2 et l'inertie totale ϕ^2 . En effet, on a :

$$\chi^2 = n \times \phi^2 \quad (2.18)$$

```
# Khi-deux
chi2 = neff*Eigen.iloc[:,0].sum()
print("Khi - deux vaut %.2f" %(chi2))

Khi - deux vaut 233.43
```

On retrouve la même valeur que celle calculée à partir des effectifs, soit $\chi^2 = 233.43$.

2.2.4 Ajustement des nuages N_I des profils lignes et N_J des profils colonnes

1) Coordonnées des modalités lignes

La matrice à exploiter est la matrice des vecteurs singuliers à gauche. Les coordonnées des modalités lignes sont obtenues à partir des vecteurs de U . Ainsi, pour un profil ligne i , sa coordonnée sur l'axe α est égale à :

$$F_\alpha(i) = \frac{\mathbf{u}_\alpha(i) \times \delta_\alpha}{\sqrt{\frac{n_{i\bullet}}{n}}} = \frac{\mathbf{u}_\alpha(i) \times \delta_\alpha}{\sqrt{f_{i\bullet}}} \quad (2.19)$$

```
# Coordonnées des profils lignes
coord = pd.DataFrame(U[:, :fmax] * delta[:, :fmax], index = donnee.index,
                      columns = index)
rowcoord = coord.apply(lambda x : x / np.sqrt(rowweight), axis=0)
rowcoord.index.name = 'V1'
# Affichage
print(rowcoord.T.round(3))
```

V1	2 conj. tr. également	trav. mari + absorbant	seul le mari trav.
Dim.1	-0.559	-0.244	0.310
Dim.2	0.233	-0.172	0.038

TABLE 2.26 – Coordonnées des profils lignes

Nous positionnons nos profils lignes sur un plan factoriel. Pour cela, nous définissons une fonction de visualisation.

```
# Fonction de visualisation en 2D
def ca_plot(data,eigen,axe1,axe2,main,figsize=None):
    try:
        if axe1==axe2:
            raise ValueError('Erreur: axe1 doit être différent de axe2.')
        elif axe1>axe2:
            raise ValueError('Erreur: axe1 doit être inférieur à axe2.')
        elif axe1<0 or axe2<0:
```

```

msg = 'Erreur: les valeurs des axes doivent être positives ou nulles.'
raise ValueError(msg)
else:
    # set limite
    n = data.shape[0]
    # Valeurs propres
    percent = np.array([100*x/sum(eigen) for x in eigen])
    dimi = round(percent[axe1],2); dimj = round(percent[axej],2)

    # Graphique
    fig, axes = plt.subplots(figsize = figsize); axes.grid()
    axes.axis([-0.8,0.8,-0.3,0.5])
    axes.set_title(f"Projection des profils {main}", fontsize=12)
    axes.set_xlabel(f"Dim.{1+axe1} ({dimi}%)", fontsize=12)
    axes.set_ylabel(f"Dim.{1+axej} ({dimj}%)", fontsize=12)
    for i in range(n):
        plt.scatter(data.iloc[i,axe1], data.iloc[i,axej],
                    c = "blue", alpha = 1)
        axes.text(data.iloc[i,axe1]-0.10,data.iloc[i,axej]+0.01,
                  data.index[i],color = "blue", fontsize = 11)
    plt.axhline(0, color='blue', linestyle="--", linewidth=0.5)
    plt.axvline(0, color='blue', linestyle="--", linewidth=0.5)
    plt.show()

# if false then raise the value error
except ValueError as e:
    print(e)

# Nuage des profils lignes sur les axes 1 et 2
ca_plot(data=rowcoord,eigen=eigenvalue,axe1=0,axej=1,
         main = 'lignes',figsize=(12,8))

```

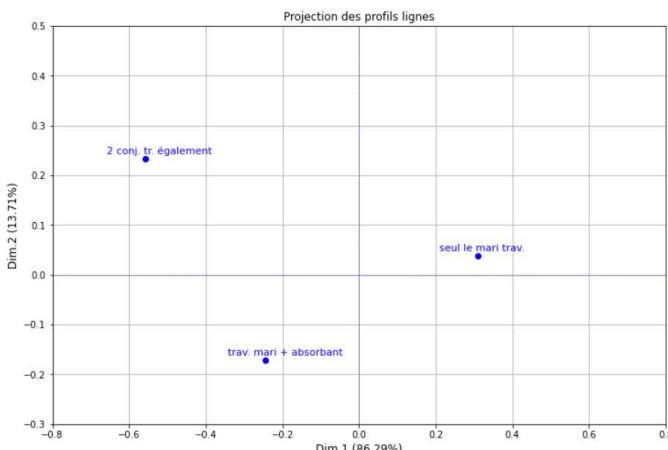


FIGURE 2.8 – Projection des modalités de V1 dans le plan factoriel

2) Coordonnées des modalités colonnes

Ici, nous utilisons la matrice des vecteurs singuliers à droite. Les coordonnées des modalités colonnes sont obtenues à partir des vecteurs de V . Pour un profil colonne j , sa coordonnée sur l'axe α est égale à :

$$G_\alpha(j) = \frac{\mathbf{v}_\alpha(j) \times \delta_\alpha}{\sqrt{\frac{n_{\bullet j}}{n}}} = \frac{\mathbf{v}_\alpha(j) \times \delta_\alpha}{\sqrt{f_{\bullet j}}} \quad (2.20)$$

Coordonnées des profils colonnes

```
coordcol = pd.DataFrame(np.transpose(V[:fmax, :])*delta[:2], columns=index,
                        index = donnee.columns)
colcoord = coordcol.apply(lambda x : x/np.sqrt(colweight), axis=0)
colcoord.index.name='V2'
print(colcoord.T.round(3))
```

V2	rester au foyer	trav. à mi-temps	trav. plein temps
Dim.1	0.618	-0.004	-0.541
Dim.2	0.183	-0.100	0.189

TABLE 2.27 – Coordonnées des profils colonnes

Nous faisons appel à la fonction `ca_plot`.

Nuage des profils colonnes sur les axes 1 et 2
`ca_plot(data=colcoord,eigen=eigenvalue,axe1=0,axe2=1,
main = 'colonnes',figsize=(12,8))`

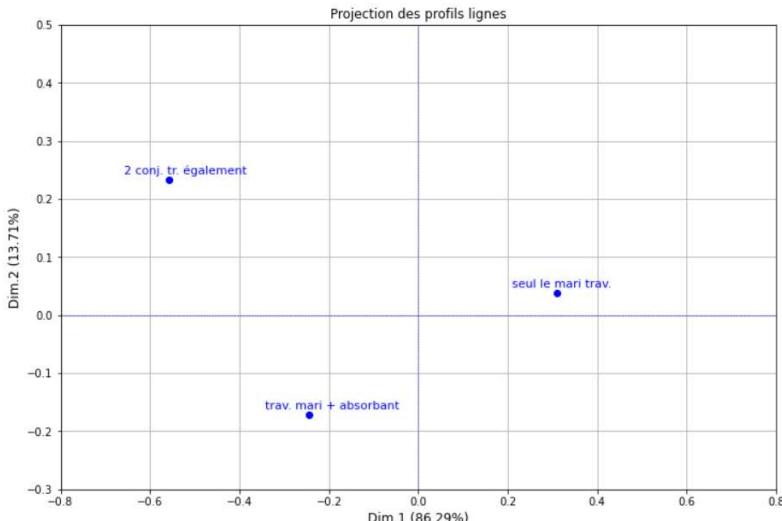


FIGURE 2.9 – Représentation dans profils colonnes

2.2.5 Représentation simultanée des profils - Relation de transition

Revenons maintenant sur l'interprétation du graphe de l'AFC afin d'introduire une caractéristique essentielle en AFC. Pour l'instant, nous avons commenté le graphe en considérant d'une part les lignes et d'autre part les colonnes. L'AFC permet une représentation simultanée des lignes et des colonnes. Dans son principe, cette représentation simultanée est possible car lignes et colonnes sont des éléments de même nature : il s'agit des modalités de variables qualitatives. La représentation simultanée des lignes et des colonnes fonctionne en AFC grâce à des relations qu'on appelle relation de transition ou propriétés barycentriques.

3 relations caractérisent l'Analyse Factorielle des Correspondances. La première relation a déjà été présentée : les deux nuages N_I et N_J ont la même inertie totale. En AFC, le caractère claire et cruciale de cette inertie totale ($\phi^2 = \text{écart à l'indépendance}$) montre bien que l'on étudie la même chose, via N_I d'une part ou via N_J d'autre part. La deuxième relation indique que, en projection sur l'axe α (u_α pour N_I dans \mathbb{R}^J , v_α pour N_J dans \mathbb{R}^I), l'inertie de N_I est égale à celle de N_J et est notée λ_α . Ainsi, non seulement les nuages N_I et N_J ont la même inertie totale mais ils ont la même inertie en projection sur les axes factorielles de même rang. Cette propriété caractérise les axes factoriels : aucune autre paire de directions (l'une dans \mathbb{R}^J , l'autre dans \mathbb{R}^I) ne l'a possède.

La troisième relation, clef de voûte de l'interprétation, relie les coordonnées des lignes et celles des colonnes sur les axes de même rang.

Sur un axe α quelconque, il est possible d'obtenir les coordonnées des modalités lignes $F_\alpha(i)$ à partir des modalités colonnes $G_\alpha(j)$ avec :

$$F_\alpha(i) = \frac{1}{\sqrt{\lambda_\alpha}} \sum_{j=1}^J \frac{f_{ij}}{f_{i\bullet}} \times G_\alpha(j) \quad (2.21)$$

avec $F_\alpha(i)$ la coordonnée du profil ligne i sur l'axe de rangs α (dans \mathbb{R}^J) ; $G_\alpha(j)$ la coordonnée du profil colonne j sur l'axe de rang α (dans \mathbb{R}^I) ; λ_α est l'inertie associée à l'axe α (en AFC $\lambda_\alpha \leq 1$).

Le long de l'axe α , on calcule le barycentre de toutes les colonnes ; chaque colonne j étant affectée du poids $\frac{f_{ij}}{f_{i\bullet}}$. Une fois qu'on a calculer ces centres (barycentres), on les dilate puisque le coefficient $\frac{1}{\sqrt{\lambda_\alpha}} \geq 1$. On peut formuler cette relation de transition de la façon suivante : la ligne i est au barycentre de toutes les colonnes ; chaque colonne étant affectée du poids $\frac{f_{ij}}{f_{i\bullet}}$ qui est le $j^{\text{ième}}$ terme du profil ligne i . Cette propriété barycentrique concrète peut se traduire ainsi de façon simplifier par : une ligne est du côté des colonnes auxquelles elle s'associe le plus. C'est-à-dire que si $\frac{f_{ij}}{f_{i\bullet}}$ est très grand, alors $G_\alpha(j)$ compte beaucoup dans le calcul de la coordonnée de i .

```
# Relation de transition - Coordonnées profils lignes
transition1 = rowprof.dot(colcoord)/np.sqrt(eigenvalue)
print(transition1.iloc[:,[0,1]].T.round(3))
```

En AFC, lignes et colonnes jouent des rôles symétriques. Si on permute les rôles joués par les lignes et les colonnes, on obtient la seconde propriété barycentrique. Réciproquement, pour un profil colonne j , on a la relation barycentrique suivante :

$$G_\alpha(j) = \frac{1}{\sqrt{\lambda_\alpha}} \sum_{i \in I} \frac{f_{jl}}{f_{\bullet i}} \times F_\alpha(i) \quad (2.22)$$

Concrètement, une colonne est du côté des lignes auxquelles elle s'associe le plus.

```
# Relation de transition - Coordonnées profils colonnes
transition2 = colprof.T.dot(rowcoord)/np.sqrt(eigenvalue)
print(transition2.iloc[:,[0,1]].T.round(3))
```

Cette double propriété barycentrique permet une utilisation de la représentation simultanée.

```
# Nuage simultané
def biplot(data1,data2,eigen,axe1,axe2,figsize=None):
    # Représentation simultanée des individus et des variables
    try:
        if axe1==axe2:
            raise ValueError('Erreur: axe1 doit être différent de axe2.')
        elif axe1>axe2:
            raise ValueError('Erreur: axe1 doit être inférieur à axe2.')
        elif axe1<0 or axe2<0:
            msg = 'Erreur: les valeurs des axes doivent être positives ou nulles.'
            raise ValueError(msg)
        else:
            n = data1.shape[0];p=data2.shape[0]
            # Valeurs propres
            percent = np.array([100*x/sum(eigen) for x in eigen])
            dimi = round(percent[axe1],2); dimj = round(percent[axe2],2)

            # Biplot
            fig = plt.figure(figsize=(12,8))
            axes1 = fig.add_subplot(111)
            axes2 = axes1.twiny()
            axes2 = axes2.twinx()
            axes1.grid()
            axes2.axis([-0.8,.8,-.6,0.6])
            axes1.axis([-0.8,.8,-.6,0.6])
            axes1.set_title("Biplot")
            axes1.set_xlabel(f"Dim.{1+axe1} ({dimi}%)")
            axes1.set_ylabel(f"Dim.{1+axe2} ({dimj}%)")
            # Affichage des individus
            for i in range(n):
                axes1.scatter(data1.iloc[i,0],data1.iloc[i,1],
                            c = "blue",alpha = 1,marker="^")
                axes1.text(data1.iloc[i,0]-0.10,data1.iloc[i,1]+0.01,
                           data1.index[i],color = "blue",fontsize=12)
            # Affichage des variables
            for k in range(p):
                axes2.scatter(data2.iloc[k,0],data2.iloc[k,1],
                            color='red',alpha=1,marker="s")
                axes2.text(data2.iloc[k,0]-0.10,data2.iloc[k,1]+0.01,
```

```

        data2.index[k], color = "red", fontsize=12)
plt.axhline(0, color='blue', linestyle="--", linewidth=0.5)
plt.axvline(0, color='blue', linestyle="--", linewidth=0.5)
plt.show()

except ValueError as e:
    print(e)

# Affichage
biplot(data1=rowcoord,data2=colcoord,eigen=eigenvalue,axeI=0,axeJ=1,
       figsize=(10,10))

```

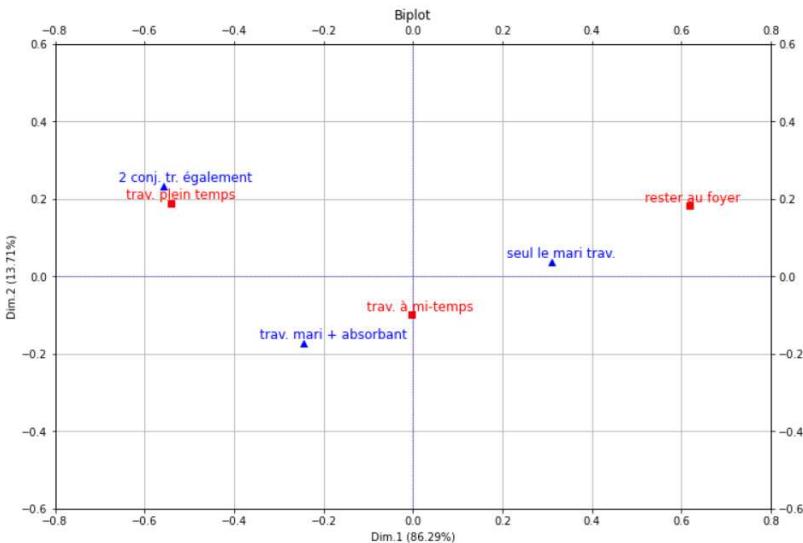


FIGURE 2.10 – Représentation simultanée des lignes et des colonnes (=superposition des figures 2.8 et 2.9)

2.3 Aides à l'interprétation

2.3.1 Analyse du point de vue des lignes

1) Qualité de la représentation - (\cos^2) des modalités lignes

Le Cosinus carré indique la qualité de représentation individuelle et cumulée des points modalités sur les $\min(I-1, J-1)$ premiers facteurs. Il représente l'information de la modalité rapportée par le ou les facteurs concernés. Il a le même numérateur que la contribution, mais normalisée cette fois-ci par l'inertie totale de la modalité. Il peut éventuellement être exprimé en pourcentage. La somme inter-facteur des COS^2 d'une modalité sur l'ensemble des axes est égale à 100%. Pour l'axe α , le COS^2 s'écrit d'un profil ligne i est :

$$\text{COS}_\alpha(i) = \frac{F_\alpha^2(i)}{d^2(i, G_I)} \quad (2.23)$$

```
# Cosinus des modalités lignes
rowcos2 = rowcoord.apply(lambda x: x**2/rowdisto.values, axis = 0)
print(rowcos2.T.round(3))
```

V1	2 conj. tr. également	trav. mari + absorbant	seul le mari trav.
Dim.1		0.667	0.985
Dim.2	0.149	0.333	0.015

TABLE 2.28 – Cosinus carrés des modalités de V1

```
# Affichage graphique
def plot_graph(data,axis,xlabel,title,figsize=None):
    p = data.shape[1]
    try:
        if axis<0 or axis>p:
            raise ValueError(f'axis doit être compris entre {0} et {p-1}.')
        else:
            sort = data.sort_values(by=f'Dim.{1+axis}', ascending=True)
            sort.iloc[:,axis].plot.barh(figsize=figsize)
            plt.xlabel(xlabel,fontsize=12)
            plt.title(f"{title} sur l'axe {1+axis}",fontsize=12)
            plt.show()
    except ValueError as f:
        print(f)
# Cosinus carré axe 1
plot_graph(data=rowcos2,axis=0,xlabel = 'Cosinus2',
            title = 'Cosinus carré des modalités de V1',figsize=(10,8))

# Cosinus carré axe 2
plot_graph(data=rowcos2,axis=1,xlabel = 'Cosinus',
            title = 'Cosinus carré des modalités de V1',figsize=(10,8))
```

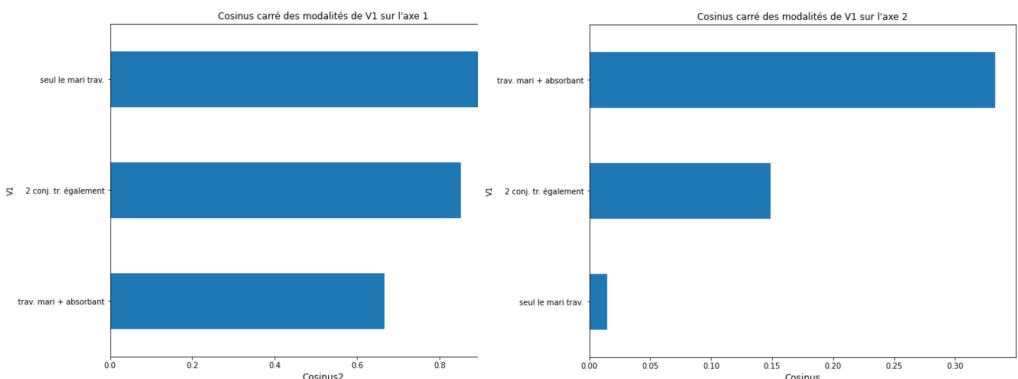


FIGURE 2.11 – Cosinus carrés des modalités de V1 sur les facteurs 1 et 2

2) Contribution des modalités lignes

Elles qualifient l'influence relative des modalités dans la définition des axes factoriels. La contribution d'une modalité correspond à la fraction d'information qu'elle porte dans la variance restituée par l'axe (λ_s). Concrètement, pour la CTR de la modalité ligne i sur l'axe α , on a :

$$CTR_{\alpha}(i) = \frac{f_{i\bullet} \times F_{\alpha}^2(i)}{\sum_{l=1}^I f_{l\bullet} F_{\alpha}^2(l)} = \frac{f_{i\bullet} \times F_{\alpha}^2(i)}{\lambda_{\alpha}} \quad (2.24)$$

Les contributions indiquent dans quelles mesure on peut considérer qu'un axe est dû à un élément ou à quelques éléments. Si par exemple, sur un axe, on retrouve qu'un élément contribue à 80% à l'inertie de cet axe, on pourra limiter l'interprétation de cet axe à cet élément.

```
# Contribution des modalités lignes
contrib = rowcoord.apply(lambda x: 100*x**2*rowprob, axis=0)
rowcontrib = contrib.apply(lambda x : x/eigenvalue, axis=1)
print(rowcontrib.round(3))
```

V1	2 conj. tr. également	trav. mari + absorbant	seul le mari trav.
Dim.1	40.432	16.371	43.197
Dim.2	44.429	51.436	4.135

TABLE 2.29 – Contributions des modalités de V1

```
# Contribution axe 1
plot_graph(data=rowcontrib, axis=0, xlabel = 'Contribution(%)',
            title = 'Contributions des modalités de V1', figsize=(10,8))

# Contribution axe 2
plot_graph(data=rowcontrib, axis=1, xlabel = 'Contribution(%)',
            title = 'Contributions des modalités de V1', figsize=(10,8))
```

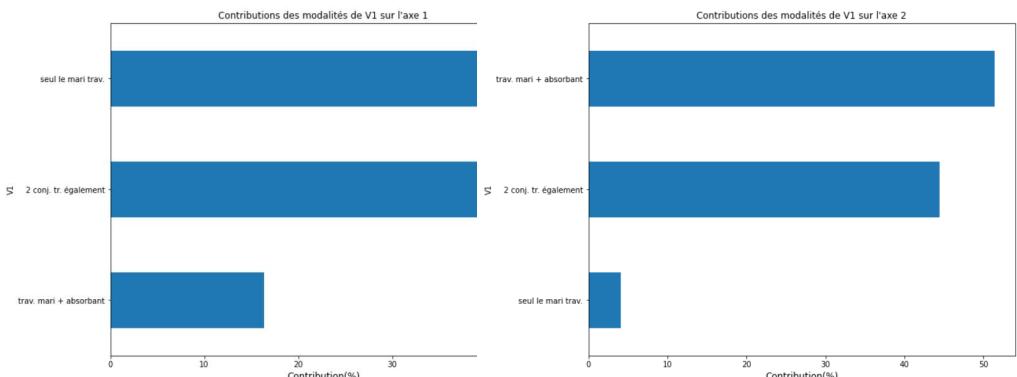


FIGURE 2.12 – Contributions des modalités de V1 sur les facteurs 1 et 2

2.3.2 Analyse du point de vue des colonnes

1) Qualité de la représentation - (\cos^2) des modalités colonnes

Les Cosinus carré indiquent la qualité de représentation des points colonnes sur les axes factoriels. Ils représentent l'information de la modalité rapporté par le ou les facteurs concerné(s).

$$\text{COS}_{\alpha}^2(j) = \frac{G_{\alpha}^2(j)}{\text{d}^2(j, G_J)} \quad (2.25)$$

```
# Cosinus carrés des points colonnes
colcos2 = colcoord.apply(lambda x:x**2/coldisto, axis = 0)
print(colcos2.T.round(3))
```

V2	rester au foyer	trav. à mi-temps	trav. plein temps
Dim.1	0.92	0.001	0.891
Dim.2	0.08	0.999	0.109

TABLE 2.30 – Cosinus carrés des modalités de V2

```
# Cosinus carré axe 1
plot_graph(data=colcos2, axis=0, xlabel = 'Cosinus2',
           title = 'Cosinus carré des modalités de V2', figsize=(10,8))

# Cosinus carré axe 2
plot_graph(data=colcos2, axis=1, xlabel = 'Cosinus',
           title = 'Cosinus carré des modalités de V2', figsize=(10,8))
```

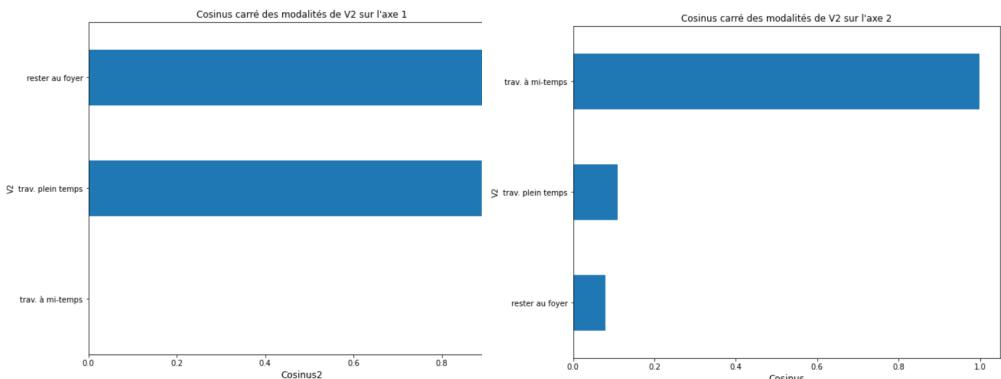


FIGURE 2.13 – Cosinus carrés des modalités de V2 sur les facteurs 1 et 2

2) Contribution des modalités colonnes

La contribution de la modalité colonne j sur l'axe α , on a :

$$CRT_{\alpha}(j) = \frac{f_{\bullet j} \times G_{\alpha}^2(j)}{\lambda_{\alpha}} \quad (2.26)$$

```

# Contribution des modalités colonnes
contrib2 = colcoord.apply(lambda x: 100*x**2*colprob, axis=0)
colcontrib = contrib2.apply(lambda x : x/eigenvalue, axis=1)
print(colcontrib.T.round(3))

```

V2	rester au foyer	trav. à mi-temps	trav. plein temps
Dim.1	53.913	0.007	46.079
Dim.2	29.613	34.853	35.533

TABLE 2.31 – Contribution des modalités de V2

```

# Contribution axe 1
plot_graph(data=colcontrib, axis=0, xlabel = 'Contribution(%)',
            title = 'Contributions des modalités de V2', figsize=(10,8))

# Contribution axe 2
plot_graph(data=colcontrib, axis=1, xlabel = 'Contribution(%)',
            title = 'Contributions des modalités de V2', figsize=(10,8))

```

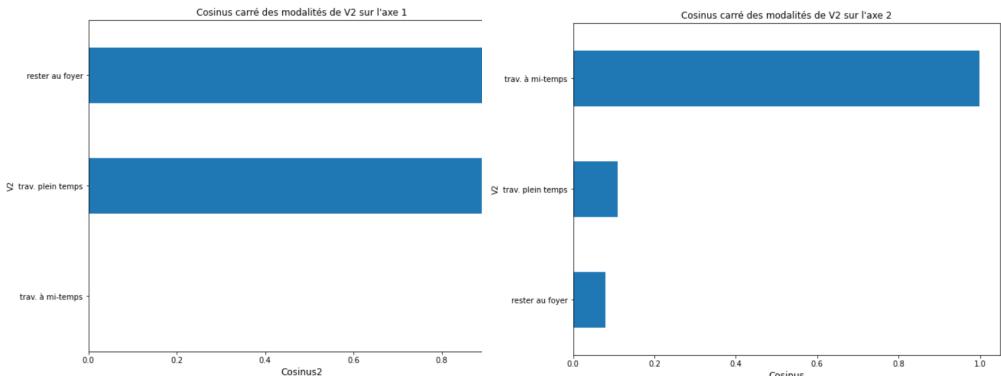


FIGURE 2.14 – Contributions des modalités de V2 sur les facteurs 1 et 2

2.3.3 Quelques mesures dérivées du χ^2

1) V de Cramer

On sait que le nombre maximum d'axe d'inertie non nul en AFC est inférieur au minimum des deux nombres ($I-1, J-1$). Ainsi, on a la conséquence suivante sur ϕ^2 :

$$\phi^2 = \sum_{k=1}^{\min(I-1, J-1)} \lambda_k \leq \min(I-1, J-1) \quad (2.27)$$

Sachant que toutes les valeurs propres sont inférieures ou égales à 1 au minimum de $I-1$ et $J-1$. D'où l'idée de rapporter ce ϕ^2 à sa valeur maximum qui est ainsi un indicateur appelé V de Cramer défini par :

$$V \text{ de Cramer} = \sqrt{\frac{\phi^2}{\min(I-1, J-1)}} \in [0,1] \quad (2.28)$$

C'est un indicateur borné de la liaison entre 2 variables. Le V de Cramer mesure l'intensité de la liaison entre deux variables qualitatives.

```
# V de cramer
Vcramer = np.sqrt(inertietot/fmax)
print("V de Cramer vaut %.4f" %(Vcramer))

V de Cramer vaut 0.2602
```

C'est une valeur faible. Elle nous indique qu'on est très très loin d'une association exclusive entre modalités.

2) Le T de Tschuprow

Le T de Tschuprow correspond également à une normalisation du ϕ^2 par les degrés de libertés. Sous certaines conditions, il varie entre 0 et 1. Sa formule est :

$$T = \sqrt{\frac{\phi^2}{\sqrt{(I-1) \times (J-1)}}}, \quad \text{avec } \phi^2 = \frac{\chi^2}{n} \quad (2.29)$$

```
# indicateurs de liaison
def VCramer(phi2,l,c):
    return (phi2/min(l-1,c-1))**(0.5)

def Tschuprow(phi2,l,c):
    return np.sqrt(phi2/np.sqrt((l-1)*(c-1)))

def mesure_liaison(data):
    l,c = data.shape
    N = data.sum().sum()
    chi2, pvalue= stats.chi2_contingency(data)[0:2]
    phi2 = chi2/N
    V = VCramer(phi2,l,c)
    T = Tschuprow(phi2,l,c)
    df=pd.DataFrame({"phi-2":phi2,"khi-2":chi2,'p.value':pvalue,
                     "VCramer":V,"Tschuprow":T},index=['valeur'])
    return df
indicateur = mesure_liaison(data=donnee)
indicateur.index.name = 'Mesure'
print(indicateur.round(4))
```

Mesure	phi-2	khi-2	p.value	VCramer	Tschuprow
valeur	0.1354	233.4304	0.0	0.2602	0.2602

TABLE 2.32 – Indicateurs de liaison entre V1 et V2

2.4 Éléments supplémentaires

Comme dans toute analyse factorielle, on peut introduire des éléments supplémentaires (lignes et/ou colonnes). Ils n'interviennent pas dans la construction des axes. En Analyse Factorielle des Correspondances, les éléments supplémentaires sont généralement des tableaux de contingence. Leur position sur le plan est calculée en utilisant les propriétés barycentriques. La représentation des modalités d'un tableau de contingence introduit en colonnes supplémentaires (croissant V1 et une troisième variable V3 par exemple) prend en compte l'intensité de liaison entre V1 et V2.

Illustrons cette approche par un exemple issu de l'ouvrage de N. Tabard, croisant V1 et une nouvelle question. Cette nouvelle variable est d'un format très classique dans les questionnaires d'opinion. Le libellé exact de la question est : Que pensez-vous de l'opinion suivante entendue quelquefois : les femmes qui ne travaillent pas se sentent coupées du monde ? avec 4 modalités de réponses :

- Tout à fait d'accord
- Plutôt d'accord
- Pas très d'accord
- Pas du tout d'accord.

La reproduction du tableau publiée est la suivante :

	F foyer coupées du monde tout à fait d accord	F foyer coupées du monde plutôt d accord	F foyer coupées du monde pas très d accord	F foyer coupées du monde pas du tout d accord
2 conj. tr. également	107	75	40	39
trav. mari + absorbant	192	175	100	88
seul le mari trav.	140	215	254	299

TABLE 2.33 – Tableau de contingence liant V1 et V3

```
# Colonnes supplémentaires
donneesup = pd.read_csv('femme_travail.csv', delimiter=";", encoding = "cp1252",
                        index_col = 0,usecols=[0,4,5,6,7])
donneesup
```

On calcule les indicateurs de liaison entre V1 et V3.

```
# Liaison V1 et V3
liaisonV1_V3 = mesure_liaison(data=donneesup)
print(liaisonV1_V3.round(4))
```

Mesure	phi-2	khi-2	p.value	VCramer	Tschuprow
valeur	0.0941	162.1882	0.0	0.2169	0.196

TABLE 2.34 – Indicateurs de liaison entre V1 et V3

La liaison entre V1 et V3 est hautement significative ($\chi^2 = 162.1882$, p.value = 0.000) mais peu intense ($\phi^2 = 0.0941$, VCramer = 0.2169), en particulier moins intense que celle entre V1 et V2 ($\chi^2 = 0.1354$, VCramer=0.2602).

La coordonnée d'un profil colonne supplémentaire j' sur l'axe α est donnée par :

$$G_\alpha(j') = \frac{1}{\sqrt{\lambda_\alpha}} \sum_{i \in I} \frac{f_{ij'}}{f_{\bullet i}} \times F_\alpha(i) \quad (2.30)$$

Afin d'appliquer cette formule, nous devons au préalable calculer la quantité $f_{ij'}/f_{\bullet i}$.

```
# profil colonne supplémentaire
colsupprofil = donneesup.apply(lambda x:x/np.sum(x), axis=0)
print(colsupprofil)
```

V3	F foyer coupées du monde tout à fait d accord	F foyer coupées du monde plutôt d accord	F foyer coupées du monde pas très d accord	F foyer coupées du monde pas du tout d accord
2 conj. tr. également	0.244	0.161	0.102	0.092
trav. mari + absorbant	0.437	0.376	0.254	0.207
seul le mari trav.	0.319	0.462	0.645	0.702

TABLE 2.35 – Profils colonnes supplémentaires

On peut à présent appliquer la formule ?? :

```
# Coordonnées des colonnes supplémentaires
colsupcoord = colsupprofil.T.dot(rowcoord)/np.sqrt(eigenvalue)
colsupcoord.index.name='V3'
print(colsupcoord.round(3))
```

V3	Dim.1	Dim.2
F foyer coupées du monde tout à fait d accord	-0.421	-0.046
F foyer coupées du monde plutôt d accord	-0.113	-0.070
F foyer coupées du monde pas très d accord	0.237	0.034
F foyer coupées du monde pas du tout d accord	0.339	0.092

TABLE 2.36 – Coordonnées des modalités de V3

```
# Nuage simultané
def biplot2(data1,data2,data3,eigen,axe1,axe2,figsize=None):
    # Représentation simultanée des individus et des variables
    try:
        if axe1==axe2:
            raise ValueError('Erreur: axe1 doit être différent de axe2.')
        elif axe1>axe2:
            raise ValueError('Erreur: axe1 doit être inférieur à axe2.')
    except:
        print("Error: axe1 ou axe2 must be integers between 1 and 3")
```

```

    elif axe1<0 or axej<0:
        msg = 'Erreur: les valeurs des axes doivent être positives ou nulles.'
        raise ValueError(msg)
    else:
        n = data1.shape[0]; p=data2.shape[0]; q = data3.shape[0]
        # Valeurs propres
        percent = np.array([100*x/sum(eigen) for x in eigen])
        dimi = round(percent[axe1],2); dimj = round(percent[axej],2)

        # Biplot
        fig = plt.figure(figsize=(12,8))
        axes1 = fig.add_subplot(111)
        axes2 = axes1.twiny()
        axes2 = axes2.twinx()
        axes1.grid()
        axes2.axis([-0.8,.8,-.6,0.6])
        axes1.axis([-0.8,.8,-.6,0.6])
        axes1.set_title("Biplot")
        axes1.set_xlabel(f"Dim.{1+axe1} ({dimi}%)")
        axes1.set_ylabel(f"Dim.{1+axej} ({dimj}%)")
        # Affichage des individus
        for i in range(n):
            axes1.scatter(data1.iloc[i,axe1],data1.iloc[i,axej],
                          c = "blue",alpha = 1,marker="^")
            axes1.text(data1.iloc[i,axe1]-0.10,data1.iloc[i,axej]+0.01,
                       data1.index[i],color = "blue",fontsize=12)
        # Affichage des variables
        for k in range(p):
            axes2.scatter(data2.iloc[k,axe1],data2.iloc[k,axej],
                          color='red',alpha=1,marker="s")
            axes2.text(data2.iloc[k,axe1]-0.10,data2.iloc[k,axej]+0.01,
                       data2.index[k],color = "red",fontsize=12)
        for l in range(q):
            axes2.scatter(data3.iloc[l,axe1],data3.iloc[l,axej],
                          color='black',alpha=1,marker="*")
            axes2.text(data3.iloc[l,axe1]-0.10,data3.iloc[l,axej]+0.01,
                       data3.index[l],color = "black",fontsize=12)
        plt.axhline(0, color='blue',linestyle="--", linewidth=0.5)
        plt.axvline(0, color='blue',linestyle="--", linewidth=0.5)
        plt.show()

except ValueError as e:
    print(e)

```

On peut avoir les quelques commentaires suivants :

- Les modalités exprimant l'accord avec l'opinion *les femmes au foyer se sentent coupées du monde* se trouvent du côté des attitudes défavorables à l'égard du travail féminin ; et interviennent pour les modalités exprimant un désaccord.

- Le nuage des modalités de V3 est plus concentré autour de l'origine que ceux des deux autres variables ; on retrouve le fait que la liaison entre V1 et V3 est moins intense que celle entre V1 et V2.
- La modalité *tout à fait d'accord* est plus éloignée de l'origine des axes que le l'est *pas du tout d'accord* ; elle semble être plus caractéristique d'une attitude favorable au travail féminin que *pas du tout d'accord* ne l'est d'une attitude défavorable.

Affichage

```
biplot2(data1=rowcoord,data2=colcoord,data3=colsupcoord,eigen=eigenvalue,
        axe1=0,axej=1,figsize=(10,10))
```

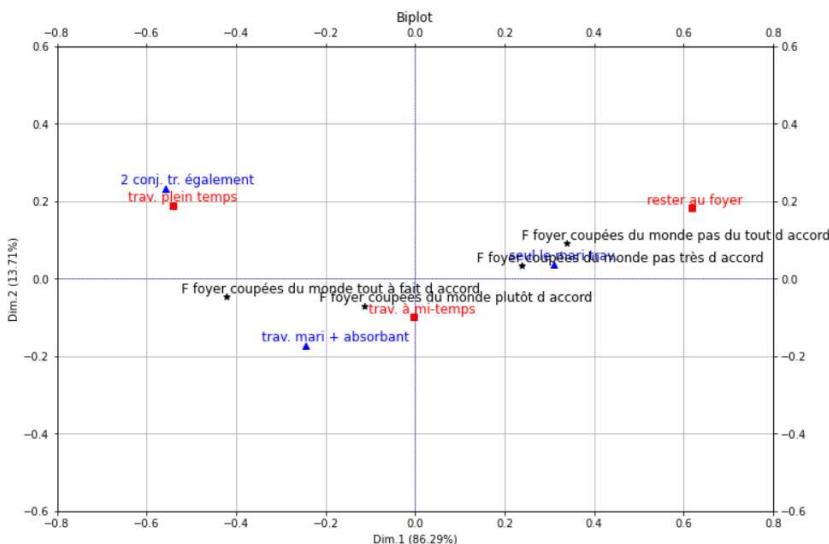


FIGURE 2.15 – Représentation de la figure 2.10 complétée par les modalités de la variable supplémentaire V3

V1	Distro2	poids	Inertie	coordonnées		Contributions		Cosinus carrés	
				Dim.1	Dim.2	Dim.1	Dim.2	Dim.1	Dim.2
2 conj. tr. également trav. mari + absorbant	0.367	0.151	0.055	-0.559	0.233	40.432	44.429	0.851	0.149
seul le mari trav.	0.089	0.322	0.029	-0.244	-0.172	16.371	51.436	0.667	0.333
	0.097	0.527	0.051	0.310	0.038	43.197	4.135	0.985	0.015

TABLE 2.37 – Analyse des modalités de V1

V2	Distro2	poids	Inertie	Coordonnées		Contributions		Cosinus carrés	
				Dim.1	Dim.2	Dim.1	Dim.2	Dim.1	Dim.2
rester au foyer	0.416	0.165	0.068	0.618	0.183	53.913	29.613	0.920	0.080
trav. à mi-temps	0.010	0.651	0.006	-0.004	-0.100	0.007	34.853	0.001	0.999
trav. plein temps	0.329	0.184	0.060	-0.541	0.189	46.079	35.533	0.891	0.109

TABLE 2.38 – Analyse des modalités de V2

CHAPITRE 3

ANALYSE FACTORIELLE DES CORRESPONDANCES MULTIPLES (AFCM)

Sommaire

3.1	Données - Objectifs et problématique	84
3.2	Mise en œuvre d'une ACM	101
3.3	Représentation des nuages N_I et N_K	110
3.4	Aide à l'interprétation des résultats	117
3.5	Approche par machine learning	131

L'Analyse Factorielle des Correspondances (AFC) peut se généraliser de plusieurs manières dans le cas où non plus deux variables sont mises en correspondance, mais des ensembles de variables. La généralisation la plus simple et la plus employée est l'analyse factorielle des correspondances multiples (AFCM) ou Analyse des Correspondances Multiples (ACM). Les principales caractéristiques de la méthode seront décrites à partir d'un exemple d'application. Nous commencerons par décrire les données sur lesquelles on travaille en ACM. A partir de ces données, nous dégagerons des objectifs et une problématique liés à l'ACM. Ceci nous conduira à une transformation du tableau des données. Comme dans toute analyse factorielle, en ACM, on construira des nuages de points : nuages des lignes et nuages des colonnes. Il s'agit ici d'un nuage des individus et d'un nuage des modalités. Nous verrons comment visualiser le nuage des individus et comment l'interpréter grâce aux modalités. Nous verrons également comment visualiser le nuage des modalités. Ensuite, nous montrerons qu'en ACM le nuage des individus et celui des modalités peuvent être représentés simultanément sur un même graphe : on parle de représentation simultanée des deux nuages. Nous verrons également les différentes aides à l'interprétation classique en analyse factorielle. Une approche d'analyse supervisée sera utilisée en fin de chapitre.

3.1 Données - Objectifs et problématique

Les données sur lesquelles nous travaillons sont constituées par tableau rectangulaire. On trouve en lignes, les individus avec la lettre i réservée aux individus et I le nombre d'individus ou encore l'ensemble des individus. En colonnes, on retrouve les variables qualitatives. La lettre j est réservée aux variables qualitatives, j est la variable courante et J est l'ensemble des variables ou

encore le nombre de variables qualitatives. Cependant, distinguons bien une variable qualitative, par exemple couleur, des modalités qu'elle peut prendre, par exemple bleu, rouge, vert, etc..

	Variables qualitatives						
	1	...	j	...	J		
Individus	1						
	i	v_{ij}	
	I						

TABLE 3.1 – Tableau de données en ACM

A l'intersection de la ligne i et de la colonne j , on trouve v_{ij} , la modalité de la variable j possédée par l'individu i . $v_{ij} \in \mathcal{M}_j$ avec \mathcal{M}_j qui est l'ensemble des modalités de la j ème variable. i varie de 1 à I et j de 1 à J . On considère que la variable qualitative j possède K_j modalités et $K = \sum_{j=1}^J K_j$ le nombre total de modalités. Les données peuvent être étudiées à partir des individus, des variables et des modalités. Et cela amène à se poser plusieurs types de questions relatives à ces objets de nature différente.

Chien	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. Francais	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

TABLE 3.2 = Données Canidés

Nous illustrons l'analyse des correspondances multiples à l'aide d'un exemple sur les races canines.

nines ([Bre82]). Ces données sont extraites du livre Statistique : Méthodes pour décrire, expliquer et prévoir écrit par [Ten07]. Ces données contiennent le descriptif de 27 races de chien : Taille, Poids, Vélocité, Intelligence, codées sur trois modalités, Affection, Agressivité et fonction. Les modalités des différentes variables sont les suivantes :

- Taille, poids, vélocité, intelligence : faible (-), moyen (+), 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. Nous allons procéder à une analyse des correspondances multiples des premières caractéristiques, sans tenir compte des variables fonction et cote.

Les principales questions auxquelles nous nous posons sont les suivantes : Quels sont les chiens qui se ressemblent ? Quels sont les chiens qui sont dissemblances ? Sur quels caractères sont fondées ces ressemblances/dissemblances ? Quelles sont les associations entre modalités ? Quelles sont les relations entre les modalités ?

A partir du tableau (3.2), on remarque que les paires d'individus (Bull-Dog,Teckel), (Chiuhua,Pékinois) et (Dalmatien,Labrador) ont des valeurs identiques pour les 6 variables. Par conséquent, il y aura des observations confondues.

```
# Chargement des données
import pandas as pd

donnee = pd.read_excel('races_canines_acm.xls',sheet_name=0,header=0,index_col=0)
```

On sauvegarde dans deux variables I et J la dimension de notre tableau : le nombre d'individus actifs et le nombre de variables actives de notre jeu de données.

```
# Dimension du tableau
I = donnee.shape[0] # nombre de lignes
J = donnee.shape[1] # nombre de colonnes
print(f'Total individus : {I}. Total variables : {J}.!')
```

Total individus : 27. Total variables : 6.

3.1.1 Statistiques et graphiques

Nous faisons ressortir quelques statistiques descriptives sur nos variables. Dans le cas des variables qualitatives, l'attribut `.describe()` renvoie 4 éléments : count qui représente le nombre d'élément (sans NAN) que la variable, unique le nombre de modalités que contient la variable, top et freq respectivement pour la modalité la plus fréquente (le mode) de la variable et sa fréquence d'apparition.

```
# Statistiques descriptives
stats = donnee.describe(include=[object])
stats.index.name = 'Statistiques'
print(stats)
```

Statistiques	Taille	Poids	Velocite	Intelligence	Affection	Agressivite
count	27	27	27	27	27	27
unique	3	3	3	3	2	2
top	Taille++	Poids+	Veloc-	Intell+	Affec+	Agress-
freq	15	14	10	13	14	14

TABLE 3.3 – Statistiques descriptives sur les variables

A l'aide d'un diagramme à barres, nous visualisons nos différentes variables.

```
# Diagrammes en barres
import matplotlib.pyplot as plt

for i,name in enumerate(donnee.columns):
    plt.subplot(2, 3, i+1)
    donnee[name].value_counts().plot.bar(figsize=(12,8))
    plt.title(name)
    plt.xticks(rotation= 0)
    plt.tight_layout()
    plt.suptitle("Diagramme en barres", fontsize=12)
plt.show()
```

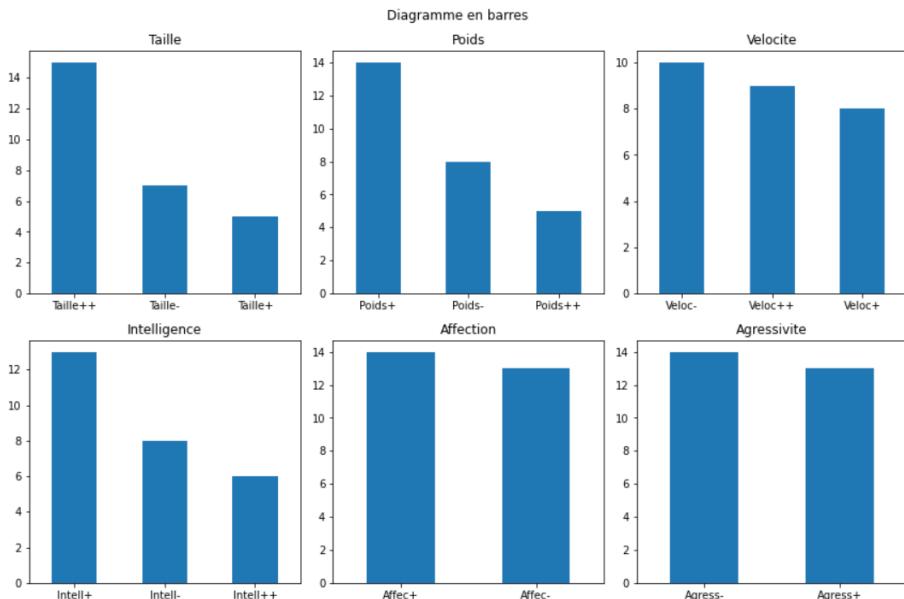


FIGURE 3.1 – Diagramme en barres des variables

Il est utile d'étudier le croisement des variables du problème prises deux à deux. Pour cela, nous calculons la statistique du test du Khi-deux d'indépendance avec son niveau de signification.

```

# khi-deux de contingence et pvalue
import scipy.stats as stat
import numpy as np
chi2 = pd.DataFrame(np.zeros(shape=(J,J), dtype=float), index=donnee.columns,
                     columns = donnee.columns)
pvalue = pd.DataFrame(np.zeros(shape=(J,J), dtype=float), index=donnee.columns,
                      columns = donnee.columns)
for i in range(J):
    for j in range(i+1,J):
        tab = pd.crosstab(donnee.iloc[:,i], donnee.iloc[:,j])
        chi = stat.chi2_contingency(tab)
        chi2.iloc[i,j] = chi[0]; pvalue.iloc[i,j]=chi[1]
# Affichage
print(chi2.round(3),pvalue.round(3))

```

	Poids	Velocite	Intelligence	Affection	Agressivite
Taille	25.329 (0.000)	15.891 (0.003)	3.608 (0.462)	13.954 (0.001)	2.051 (0.359)
Poids		18.470 (0.001)	1.356 (0.852)	9.476 (0.009)	2.552 (0.279)
Velocite			3.156 (0.532)	2.967 (0.227)	0.575 (0.750)
Intelligence				3.891 (0.143)	1.157 (0.561)
Affection					0.915 (0.339)

TABLE 3.4 – χ^2 d'indépendance (niveau de significativité)

Les caractères physiques taille, poids et vélocité sont très liés entre eux. L'affection apparaît comme reliée à la taille et au poids. Les variables intelligence et agressivité sont indépendantes entre elles et des autres variables. L'analyse factorielle des correspondances multiples va permettre, entre autres, de retrouver ces configurations types en visualisant les racines canines et les modalités.

3.1.2 Du tableau de codage condensé au tableau disjonctif complet

Du point de vue du codage, on peut remplacer v_{ij} par : codage littéral et écrit « bleu » ou un codage numérique et écrire « 2 » si par convention on avait dit que « bleu » était représenté par le chiffre 2. Ce genre de tableau est appelée tableau de codage condensé (cf. Table (3.1)).

A côté de ce tableau de contingence, on construit un tableau tableau disjonctif complet (cf. Table (3.5)) en abrégé TDC. Dans le TDC, les lignes sont les individus, mais les colonnes sont cette fois ci les modalités des variables qualitatives. Ainsi, à la colonne j du tableau de codage condensé correspondent K_j colonnes (si l'on suppose que la variable j à K_j modalités). K_j colonnes correspondant dont chacune à une modalité de la variable j .

A l'intersection de la ligne i et de la colonne k , on trouve y_{ik} qui vaut 1 si l'individu i possède la modalité k de la variable j et 0 sinon. Le tableau complet joue un rôle clé en Analyse des Correspondances Multiples car c'est ce tableau qui est analysé par les logiciels statistiques. Cependant, l'utilisateur ne le construit pas explicitement.

	Modalités				
	1	...	k	...	K
Individus	1			⋮	
i				⋮	
	y_{ik}
				⋮	
I				⋮	

TABLE 3.5 – Tableau disjonctif complet

```
# Tableau disjonctif complet
dummies = pd.get_dummies(donnee, prefix="",prefix_sep="")
print(dummies)
```

	Taille		Poids		Vélocité		Intelligence		Affection		Agressivité	
	Taille+	Taille++	Poids+	Poids++	Veloc+	Veloc++	Intell+	Intell++	Affec+	Affec-	Agress+	Agress-
Chien	Taille+	Taille++	Poids+	Poids++	Veloc+	Veloc++	Intell+	Intell++	Affec+	Affec-	Agress+	Agress-
Beauceron	0	1	0	1	0	0	0	1	0	0	1	0
Basset	0	0	1	0	0	1	0	0	1	0	1	0
Berger All	0	1	0	1	0	0	0	1	0	1	0	1
Boxer	1	0	0	1	0	0	1	0	0	1	0	1
Bull-Dog	0	0	1	0	0	1	1	0	0	1	0	0
Bull-Mastif	0	1	0	0	1	0	0	1	0	1	1	0
Caniche	0	0	1	0	0	1	1	0	0	1	0	1
Chihuahua	0	0	1	0	0	1	0	0	1	1	0	0
Cocker	1	0	0	0	0	1	0	0	1	0	1	0
Colley	0	1	0	1	0	0	1	0	0	1	0	0
Dalmatien	1	0	0	1	0	0	1	0	0	1	0	0
Doberman	0	1	0	1	0	0	1	0	0	1	1	0
Dogue All	0	1	0	0	1	0	1	0	0	1	1	0
Epag. Breton	1	0	0	1	0	0	1	0	1	0	0	1
Epag. Français	0	1	0	1	0	0	1	0	0	1	0	1
Fox-Hound	0	1	0	1	0	0	1	0	0	1	1	0
Fox-Terrier	0	0	1	0	0	1	0	0	1	0	1	0
Gd Bleu Gasc	0	1	0	1	0	0	1	0	0	1	0	1
Labrador	1	0	0	1	0	0	1	0	0	1	0	1
Levrier	0	1	0	1	0	0	1	0	0	1	0	1
Mastiff	0	1	0	0	1	0	0	1	0	1	1	0
Pekinois	0	0	1	0	0	1	0	0	1	1	0	0
Pointer	0	1	0	1	0	0	1	0	0	1	0	1
St-Bernard	0	1	0	0	1	0	0	1	0	0	1	0
Setter	0	1	0	1	0	0	1	0	0	0	1	0
Teckel	0	0	1	0	0	1	0	0	1	0	0	1
Terre-Neuve	0	1	0	0	1	0	0	0	1	0	0	1

TABLE 3.6 – Tableau disjonctif complet

On utilise les terminologies de « disjonctif » parce que dans le petit bloc, on ne peut pas rencontrer deux fois un nombre 1 et « complet » parce que l'on aura forcement un 1. On sauvegarde également dans une variable K le nombre total de modalités.

```
# total de modalités
K = dummies.shape[1]
print(f'Le nombre total de modalités est {K}.')
```

Le nombre total de modalités est 16.

Pour bien comprendre la structure d'un tableau disjonctif complet, nous allons calculer ses marges.

a) Marge Colonne

On parle de marge colonne car elle a la forme d'une colonne. En, effet, il s'agit de la somme des éléments d'une ligne sur l'ensemble de ses colonnes. On rappelle que y_i^k prend la valeur 1 si l'individu possède la modalité k et 0 sinon. Par conséquent, pour chaque bloc de colonnes correspondant à une même variable, on trouve une et une seule fois la valeur 1. Ainsi, en sommant sur l'ensemble des colonnes, on tombe sur le nombre de variables J .

$$\forall i, \quad MC(i) = \sum_{k=1}^K y_{ik} \quad (3.1)$$

On remarque que lorsqu'on effectue la somme des termes d'une ligne i sur l'ensemble de ses colonnes, on tombe sur le nombre de variables qualitatives J . La marge colonne est donc constante pour tous les individus et la somme des termes du tableau vaut $I \times J$, soit le nombre d'individus multiplié par le nombre de variables.

	1	...	k	...	K	marge colonne
Individus	1		:			
			:			
	i	y_{ik}	...	$\sum_{k=1}^K y_{ik} = J$
	I		:			

TABLE 3.7 – Marge colonne d'un TDC

b) Marge ligne

Comme y_{ik} vaut 1 si l'individu possède la modalité k , alors la somme des termes de la colonne k est égale au nombre d'individus possédant la modalité k .

$$\forall k, \quad ML(k) = I_k = \sum_{i=1}^I y_{ik} \quad (3.2)$$

	1	\cdots	k	\cdots	K
Individus	1		\vdots		
	i	\cdots	\cdots	y_{ik}	\cdots
	I			\vdots	
marge ligne			$\sum_{i=1}^I y_{ik} = I \times p_k$		

TABLE 3.8 – Marge ligne du TDC

```
# Marge colonne
colMarge= dummies.sum(axis=1)
print(colMarge)
```

Chien	marge colonne
Beauceron	6
Basset	6
Berger All	6
Boxer	6
Bull-Dog	6
Bull-Mastif	6
Caniche	6
Chihuahua	6
Cocker	6
Colley	6
Dalmatien	6
Doberman	6
Dogue All	6
Epag. Breton	6
Epag. Français	6
Fox-Hound	6
Fox-Terrier	6
Gd Bleu Gasc	6
Labrador	6
Levrier	6
Mastiff	6
Pekinois	6
Pointer	6
St-Bernard	6
Setter	6
Teckel	6
Terre-Neuve	6

TABLE 3.9 – Marge colonne

```
# Marge ligne
```

```

rowMarge = dummies.sum(axis=0)
print(rowMarge)

```

modalité	Taille+	Taille++	Taille-	Poids+	Poids++	Poids-	Veloc+	Veloc++	Veloc-	Intell+	Intell++	Intell-	Affec+	Affec-	Agress+	Agress-
marge ligne	5	15	7	14	5	8	8	9	10	13	6	8	14	13	13	14

TABLE 3.10 – Marge ligne

On rappelle que y_{ik} prend la valeur 1 si l'individu i possède la modalité k . On va trouver comme somme des termes de la colonne k , le nombre d'individus possédant la modalité k . On note $I \times p^k$ où I est le nombre total d'individus et p^k la proportion d'individus possédant la modalité k ($p^k = I^k/I$ où I^k le nombre d'individus possédant la modalité k). Si on fait la somme des K^j termes de cette marge ligne correspondant à une même variable, on trouve $\sum_{k=1}^{K^j} I \times p^k = I$ qui est donc une constante.

```

# Vérification
somme = []
for name in donnee.columns:
    modalite = rowMarge.loc[list(np.unique(donnee[[name]]))].sum()
    somme.append(modalite)
print(somme)

[27, 27, 27, 27, 27]

```

On retrouve évidemment que la somme des termes de l'ensemble du tableau vaut $I \times J$.

3.1.3 Transformation du tableau disjonctif complet

L'analyse des correspondances multiples travaille à partir du tableau disjonctif complet, mais ce dernier doit être au préalable transformé. Comme en ACP, en ACM, tous les individus ont le même poids, par conséquent, il n'y a aucune raison d'accorder plus d'importance aux réponses d'un enquêté que plutôt qu'à celle d'un autre. Vu que la somme des poids doit être égal à 1, le poids de chaque individu est égal à $1/I$.

On rappelle également que la valeur y_{ik} du tableau disjonctif complet vaut 1 si l'individu i possède la modalité k de la variable j . Cette valeur de 1 ne dépend pas de la modalité k , en particulier de son effectif. Or si un individu possède une modalité rare, cela le caractérise beaucoup plus qu'une modalité fréquente. A la limite, si une modalité est possédée par tous les individus, elle ne caractérise absolument pas l'un d'entre eux. D'où l'idée de diviser y_{ik} par p_k . On obtient ainsi une valeur x_{ik} qui sera d'autant plus grande que la modalité possédée est rare. En codant ainsi, le tableau disjonctif complet, la moyenne des x_{ik} vaut 1 :

$$\frac{1}{I} \sum_{i=1}^I x_{ik} = \frac{1}{I} \sum_{i=1}^I \frac{y_{ik}}{p_k} = \frac{1}{I} \frac{I \times p_k}{p_k} = 1 \quad (3.3)$$

En analyse des correspondances multiples, les données sont centrées car on va mettre

$$x_{ik} = \frac{y_{ik}}{p_k} - 1 \quad (3.4)$$

Le tableau des x_{ik} est le tableau sur lequel l'analyse factorielle est effectuée. x_{ik} est la donnée centrée réduite.

3.1.4 Objectifs - Problématique

1) Étude des individus

Ce qui caractérise l'étude des individus en ACM, c'est d'abord une approche multidimensionnelle. Un individu, qui est une ligne du tableau, est considéré du point de vue de l'ensemble de ses modalités. A quel moment ceci intervient ? Ceci intervient lorsque l'on compare les individus. On dira que les individus se ressemblent si dans l'ensemble, ils ont choisi les mêmes modalités et deux individus sont différents s'il y a peu de modalités en commun dans leur réponse. Alors l'ensemble de ces ressemblances et de ces différences entre individus constitue ce qu'on appelle la variabilité des individus. Et donc, en ACM, c'est bien la variabilité des individus qu'on étudie. Si tous les individus avaient répondu de la même façon aux questionnaires, il n'y aurait pas d'analyse statistique ; il suffirait de présenter le questionnaire type. Cependant, tous les individus n'ont pas répondu la même chose.

En ACM, on va regarder cette variabilité d'un point de vue multidimensionnel. Comme nous sommes en analyse factorielle, la façon de décrire cette variabilité est d'en extraire les principales dimensions. On va ainsi mettre en évidence les dimensions qui séparent par exemple les individus extrêmes et des individus moyens. Ces dimensions, elles vont être décrites en relation avec les modalités. On ne dira pas qu'une dimension sépare les individus 1, 3 et 4 de 10, 11, 12 par exemple, mais plutôt une dimension qui sépare les hommes et les femmes, les habitants du Nord et les habitants du Sud, les ouvriers et les cadres, etc.

Dans le TDC, un individu est représenté par une ligne du tableau. C'est un ensemble de K valeurs numériques. C'est donc un point dans un espace à K dimensions. Chaque dimension correspond à une modalité k . Chaque modalité en ACM est associé à un poids proportionnel à son effectif. Comme la somme des poids doit être égale à 1, le poids d'une modalité k est donc p_k/J .

Le point M_i a comme coordonnée x_{ik} , on lui affecte le poids $1/J$. Lorsqu'on considère l'ensemble des points, on considère le nuage N_I . Ce nuage N_I a pour centre de gravité G_I qui est confondu avec l'origine puisque les variables sont centrées. Tout d'abord, on calcule le profil moyen qui fait office de pondération dans l'expression 3.7.

$$\forall k, \quad \frac{I_k}{I \times J} = \frac{p_k}{J} \quad (3.5)$$

```
# Profil individu moyen
indMoyen = rowMarge/(I*J)
print(indMoyen.round(3))
```

modalité	profil moyen
Taille+	0.031
Taille++	0.093
Taille-	0.043
Poids+	0.086
Poids++	0.031
Poids-	0.049
Veloc+	0.049
Veloc++	0.056
Veloc-	0.062
Intell+	0.080
Intell++	0.037
Intell-	0.049
Affec+	0.086
Affec-	0.080
Agress+	0.080
Agress-	0.086

TABLE 3.11 – Profil moyen des modalités

a) Distance entre individus

Notons par M_i et M_j les points correspondant aux individus i et j respectivement. La distance entre les individus i et l est donnée par la formule suivante :

$$d^2(i,j) = \sum_k \frac{p_k}{J} (x_{ik} - x_{jk})^2 \quad (3.6)$$

Chaque carré des différences de coordonnées est pondérée par le poids p_k/J de la modalité correspondante. En fonction du tableau disjonctif complet, on obtient :

$$\begin{aligned} d^2(i,j) &= \sum_k \frac{p_k}{J} (x_i^k - x_j^k)^2 = \sum_k \frac{p_k}{J} \left(\frac{y_i^k}{p^k} - \frac{y_j^k}{p^k} \right)^2 \\ &= \sum_k \frac{1}{J p^k} (y_i^k - y_j^k)^2 = \frac{1}{J} \sum_k \frac{1}{p^k} (y_i^k - y_l^k)^2 \end{aligned}$$

Si on remplace p_k par I_k/I , on a :

$$d^2(i,j) = \sum_{k=1}^K \frac{1}{I_k} \left(\frac{y_{ik}}{J} - \frac{y_{jk}}{J} \right)^2 \quad (3.7)$$

Ainsi, la distance entre deux individus correspond à la somme des différences au carré pondérée par l'inverse du profil moyen.

```
# Distance du Khi-deux entre les individus
import numpy as np
rowdist = pd.DataFrame(np.zeros(shape=(I,I), dtype=float),
                       index = donnee.index,
                       columns = donnee.index)
for i in range(I):
    for j in range(i+1,I):
```

```

rowdist.iloc[i,j] = np.sum(1/indMoyen*(dummies.values[i,:]/J-
                                         dummies.values[j,:]/J)**2)

# Graphique
import seaborn as sns
plt.figure(figsize = (18,9))
plt.title('Heatmap des distances entre individus', fontsize=12)
sns.heatmap(rowdist.values, linewidth = 0.1, cmap = "Blues",
            annot=True, fmt = '.2f', xticklabels = donnee.index,
            yticklabels = donnee.index)
plt.xlabel('Individus', fontsize=12)
plt.ylabel('Individus', fontsize=12)
plt.show()

```

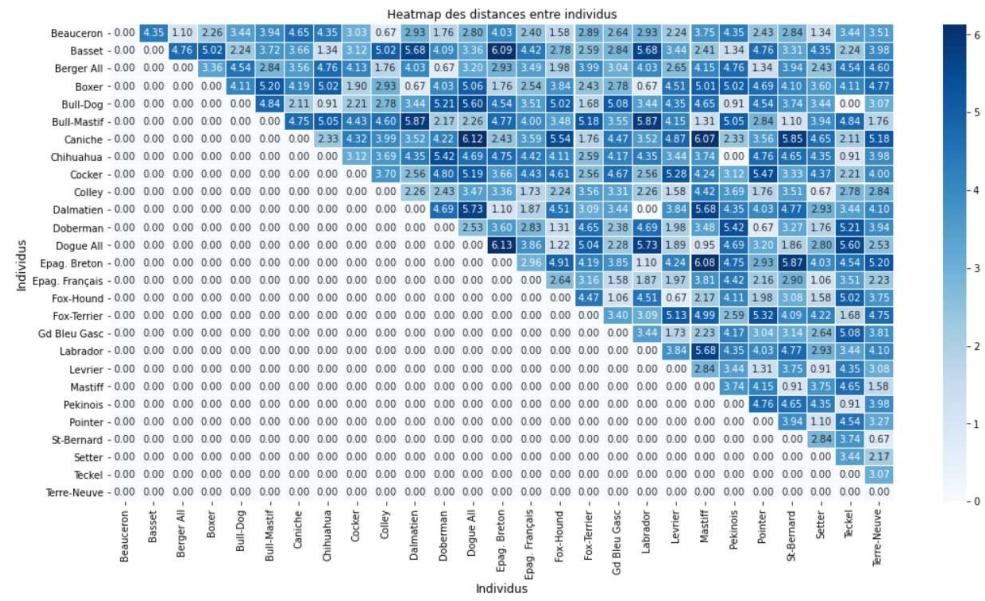


FIGURE 3.2 – Distance (au carré) entre individus

b) Distance à l'origine

Notons par M_i et G_I les points correspondant à l'individu i et à l'individu moyen respectivement. La distance (au carré) entre i et G_I est donnée par la formule ci-après :

$$d^2(i, G_I) = \sum_k \frac{p_k}{J} (x_{ik})^2 = \sum_k \frac{p_k}{J} \left(\frac{y_{ik}}{p_k} - 1 \right)^2 = \frac{1}{J} \sum_k \frac{y_{ik}}{p_k} - 1 \quad (3.8)$$

C'est une somme qui va être d'autant plus grande que les modalités possédées par l'individu i sont rares; c'est-à-dire associées à des p_k qui sont petits. Cette relation met bien en évidence qu'un individu est d'autant plus loin de l'origine qu'il possède des modalités rares, c'est-à-dire p_k petits.

$$d^2(i, G_I) = \sum_{k=1}^K \frac{1}{I_k} \left(\frac{y_{ik}}{J} - \frac{I_k}{I \times J} \right)^2 \quad (3.9)$$

```
# Distance à l'origine des observations
rowdisto = dummies.apply(lambda x : np.sum(1/indMoyen*(x/J - indMoyen)**2),
                         axis = 1)
print(rowdisto.round(3))
```

c) Inertie totale du nuage N_I

Pour calculer l'inertie totale d'un nuage, on calcule d'abord l'inertie de chaque point :

$$\text{Inertie}(i/G_I) = \frac{1}{I} \times d^2(i, G_I), \quad \forall i \quad (3.10)$$

où $1/I$ est le poids de l'individu i .

```
# Poids des observations
rowweight = np.ones(I)/I
# Inertie des lignes
rowinertie = rowdisto.values*rowweight
# Affichage
rowinfos = pd.DataFrame(np.transpose([rowdisto, rowweight, rowinertie]),
                         columns = ["Disto2", "Poids", "Inertie"],
                         index = donnee.index)
print(rowinfos.round(3))
```

L'inertie totale du nuage correspond donc à la somme des inerties de chaque point :

$$\begin{aligned} \text{Inertie}(N_I) &= \sum_{i=1}^I \text{Inertie}(i) = \sum_{i=1}^I \frac{1}{I} d^2(i, G_I) \\ &= \sum_{i=1}^I \frac{1}{I} \left(\frac{1}{J} \sum_{k=1}^K \frac{y_{ik}}{p_k} - 1 \right) = \sum_{i=1}^I \sum_{k=1}^K \frac{y_{ik}}{IJp_k} - \sum_{i=1}^I \frac{1}{I} \\ &= \frac{K}{J} - 1 \end{aligned}$$

On voit que cet inertie totale ne dépend pas du contenu du tableau lui-même, mais simplement d'un aspect de son format, c'est-à-dire le nombre de modalités (K) et le nombre de variables (J).

On remarque que ce résultat est très différent de celui de l'AFC. En effet, en AFC, l'inertie totale d'un nuage est égale à ϕ^2 . Ce ϕ^2 est très important en AFC puisque c'est lui qui mesure l'écart à l'indépendance. Par contre, ce résultat est assez analogue à celui de l'ACP normée parce qu'en ACP normée, l'inertie totale est égale au nombre de variables. Elle ne dépend pas du contenu du tableau, mais de son format.

```
# Inertie totale
inertietot = np.sum(rowinertie)
print("Inertie totale égale à %.3f."%(inertietot))
```

Inertie totale égale à 1.667.

2. Étude des variables et des modalités

En ACM, les variables sont qualitatives. Comme pour l'analyse en composantes principales, on cherche à dresser un bilan de liaisons entre variables. Ces liaisons s'étudient soit deux à deux, soit globalement. Ici, on va s'intéresser aux associations entre modalités puisque deux variables qualitatives sont liées si les modalités de l'une s'associent de façon particulière aux modalités de l'autre. Ce que l'on souhaite, c'est une visualisation d'ensemble des associations entre modalités. Ce qui nous permettra d'obtenir une visualisation d'ensemble des liaisons entre les variables. On va chercher un indicateur quantitatif (variable synthétique) fondé sur les variables qualitatives qui résume le mieux possible les variables, c'est-à-dire qui résume l'information contenue dans ces variables.

Toutes ces problématiques ressemblent beaucoup à la problématique de l'ACP. Cela tient à la structure du tableau qui est individus \times variables dans les deux cas. D'un point de vue technique, tout va se passer de façon différente puisque dans un cas, en ACP, on a des variables quantitatives et dans l'autre cas, en ACM, on a des variables qualitatives. L'ACM travaille sur le tableau disjonctif complet.

Une colonne du tableau disjonctif complet est une fonction indicatrice. Cette fonction indicatrice prend une valeur des I individus. C'est donc un ensemble de I valeurs numériques. En ce sens, c'est un point d'un espace à I dimensions. Chacune de ces dimensions correspond à un individu. Rappelons que :

- Un individu i est affecté d'un poids $1/I$
- La modalité elle-même est représentée par le point M_k à partir de ses coordonnées x_{ik}
- Le poids d'une modalité est proportionnel à son effectif p_k/J

Lorsque l'on considère l'ensemble de ces K modalités, on considère un nuage N_K des modalités. Ce nuage possède une propriété remarquable qui est la suivante : si l'on effectue pour un individu i la somme des coordonnées x_{ik} des modalités d'une même variable, on obtient 0.

$$\sum_{k \in K_j} \frac{p_k}{J} x_{ik} = 0 \quad (3.11)$$

Ce qui veut dire que l'origine est confondu avec le centre de gravité des modalités d'une même variable. Cette propriété valant pour toutes les variables, le centre de gravité du nuage N_K est confondu avec l'origine. On a là une propriété qui est tout à fait analogue à celle de l'AFC puisque, en AFC, chacun des deux nuages, celui des lignes comme celui des colonnes, a bien l'origine comme centre de gravité.

2.1 Analyse du point de vue des modalités

a) Distance entre modalités

Pour le calcul des distances, on aura besoin de la marge ligne (cf. Table 3.10). Notons par M_k et M_l les points correspondants aux modalités k et l respectivement. La distance entre les modalités k et l se calcule comme suit :

$$d^2(k, l) = \sum_{i=1}^I \frac{1}{I} \left(\frac{y_{ik}}{p_k} - \frac{y_{il}}{p_l} \right)^2 = I \sum_{i=1}^I \left(\frac{y_{ik}}{I_k} - \frac{y_{il}}{I_l} \right)^2 \quad (3.12)$$

Si nos modalités sont possédées exactement par les mêmes individus ($y_{ik} = y_{il}$), $p_k = p_l$ bien sûr et la distance est nulle. On voit que plus on a des individus qui ont choisi l'une seule de ces modalités, plus cette distance est grande.

```
# Distance entre les modalités
moddist = pd.DataFrame(np.zeros((K, K), dtype=float), index = dummies.columns,
                      columns = dummies.columns)
for i in range(K):
    for j in range(i+1,K):
        moddist.iloc[i,j] = np.sum(I*(dummies.values[:,i]/rowMarge[i]-
                                      dummies.values[:,j]/rowMarge[j])**2)

# Heatmap
plt.figure(figsize = (18,9))
sns.heatmap(moddist, linewidth = 0.1, cmap = "Blues",
            annot=True, fmt = '.2f', xticklabels = dummies.columns,
            yticklabels = dummies.columns)
plt.title('Heatmap des distances entre modalités', fontsize=12)
plt.xlabel('Modalités', fontsize=12)
plt.ylabel('Modalités', fontsize=12)
plt.show()
```



FIGURE 3.3 – Distances (au carré) entre modalités

b) Distance des modalités avec l'origine

La modalité k peut être vue comme une indicatrice. On peut donc calculer sa variance. Cette variance est égale au carré de la distance entre cette modalité et l'origine dans l'espace R^I :

$$\text{Var}(k) = d^2(k, G_K) = I \sum_{i=1}^K \left(\frac{x_{ik}}{I_k} - \frac{1}{I} \right)^2 \quad (3.13)$$

On voit que cette distance n'est rien d'autre que la somme des carrés des coordonnées pondérée par les poids $1/I$ mais qui sont constants :

$$\text{Var}(k) = d^2(k, 0) = \sum_i \frac{1}{I} (x_i^k)^2 \quad (3.14)$$

Si on exprime cette variance en fonction du TDC, on aura :

$$\begin{aligned} \text{Var}(k) &= d^2(k, 0) = \sum_i \frac{1}{I} \left(\frac{y_i^k}{p^k} - 1 \right)^2 \\ &= \frac{1}{p^k} - 1 = \frac{I}{I_k} - 1 \end{aligned}$$

C'est-à-dire que la distance entre un point M_k représentant la modalité k et l'origine 0 est d'autant plus grande que cette modalité est rare. En ACM, les modalités rares sont très éloignées de l'origine.

```
# moyenne
modmean = np.ones(n)/n
# Distance à l'origine
dummiesweight = dummies/modSum
moddisto = dummiesweight.apply(lambda x : np.sum(n*(x-modmean)**2),axis = 0)
print(moddisto.round(3))
```

c) Inertie totale du nuage N_K

Rappelons simplement qu'en Analyse factorielle, ce qui compte dans la construction des axes, c'est l'inertie. Pour avoir l'inertie totale du nuage, il faut au préalable calculé l'inertie d'une modalité. L'inertie d'une modalité k correspond au produit du carré de sa distance à l'origine avec son poids, soit :

$$\text{Inertie}(k) = \frac{p_k}{J} d^2(k, G_K) \quad (3.15)$$

On voit bien que lorsqu'une modalité est rare, la distance augmente, mais le poids diminue. Il y a donc antagonisme.

$$\text{Inertie}(k) = \frac{p_k}{J} d^2(k, G_K) = \frac{I_k}{I \times J} \left(\frac{I}{I_k} - 1 \right) = \frac{1}{J} \left(1 - \frac{I_k}{I} \right) \quad (3.16)$$

Cette formule montre bien que lorsqu'une modalité est rare, elle a une forte inertie. Elle va donc influencer les résultats de l'ACM. Le problème en ACM est celui des modalités rares qui ont une forte influence plus que les modalités très rares.

```
#Poids des modalités
modweight = modSum/(n*p)
# Inertie des lignes
modinertie = moddisto.values*modweight
# Affichage
modinfos = pd.DataFrame(np.transpose([moddisto, modweight, modinertie]),
```

```

        columns = ["Disto2", "Poids", "Inertie"],
        index = dummies.columns)
print(modinfos.round(3))

```

Comme dans le cas du Nuage N_I , l'inertie totale du nuage N_K est égale à la somme des inerties de l'ensemble de ces modalités.

$$\text{Inertie}(N_K) = \sum_{k=1}^K \text{Inertie}(k) \quad (3.17)$$

```

# Inertie totale
inertietot = np.sum(modinertie)
print('Inertie totale : %.3f.' %(inertietot))

Inertie totale : 1.667.

```

L'inertie totale est de 1.667 et identique à celle calculée avec les individus.

Remarque

L'inertie de l'ensemble des K_j modalités d'une variable j , dite inertie de la variable j , vaut :

$$\text{Inertie}(j) = \sum_{k=1}^{K_j} \frac{1}{J} \left(1 - \frac{I_k}{I} \right) \quad (3.18)$$

```

# Inertie des variables
varinertie = pd.DataFrame(np.zeros(shape=(1,J), dtype=float), index = ['inertie'],
                           columns= donnee.columns,)

for name in donnee.columns:
    varinertie.loc[:,name] = modinfos.loc[np.unique(donnee[name]),'Inertie'].sum()

# Affichage
print(varinertie.round(3))

```

Variable	Taille	Poids	Velocite	Intelligence	Affection	Agressivite
Inertie	0.333	0.333	0.333	0.333	0.167	0.167

TABLE 3.12 – Inertie des variables (équation 3.18)

Comme $\sum_{k=1}^{K_j} I_k = I$, on a :

$$\text{Inertie}(j) = \frac{K_j - 1}{J} \quad (3.19)$$

Ainsi, l'inertie d'une variable ne dépend que du nombre de modalités qui la constituent : elle est d'autant plus grande que ce nombre est grand.

```

# Inertie des variables - Approche 2
varinertie2 = pd.DataFrame(np.zeros(shape=(1,J), dtype=float), index = ['inertie'],
                           columns= donnee.columns,)

for name in donnee.columns:
    varinertie2.loc[:,name] = (len(np.unique(donnee[name]))-1)/J

# Affichage
print(varinertie2.round(3))

```

Variable	Taille	Poids	Velocite	Intelligence	Affection	Agressivite
Inertie	0.333	0.333	0.333	0.333	0.167	0.167

TABLE 3.13 – Inertie des variables (équation 3.19)

On peut calculer l'inertie associé à l'ensemble des modalités ce qui correspond à l'inertie du nuage des modalités (N_K) :

$$\text{Inertie}(N_K) = \sum_{j=1}^J \frac{K_j - 1}{J} = \frac{K}{J} - 1 \quad (3.20)$$

```
# Inertie totale
print('Inertie totale : %.3f.'%(np.sum(varinertie.values)))

Inertie totale : 1.667.
```

Cette inertie ne dépend que la structure du questionnaire, soit précisément, que du nombre moyen de modalités par variables.

3.2 Mise en œuvre d'une ACM

L'Analyse des Correspondances Multiples (ACM) n'est pas une nouvelle méthode au sens mathématique du terme, mais tout simple une application particulière de l'Analyse Factorielle des Correspondances (AFC) en ce sens où elle s'applique à des tableaux croisant des individus et leurs réponses à plusieurs variables qualitatives. Cependant, on la considère comme une méthode à part entière du fait de ses propriétés spécifiques et des résultats intéressants qu'elle fournit. L'ACM s'applique à des tableaux croisant des individus en ligne et des variables qualitatives en colonnes.

Nous mettons en œuvre une analyse des correspondances multiples à l'aide du package `mca` en fixant `benzecri=False` en lui donnant le tableau disjonctif complet pour son algorithme.

```
# Chargement du package
from mca import MCA
mcaModel = MCA(dummies,ncols=K,benzecri=False)
```

Le modèle déjà pré-entraîné, nous affichons les valeurs propres obtenues.

```
# eigenvalue dataframe
percent = np.array([100*x/sum(mcaModel.L) for x in mcaModel.L])
cumpercent = np.cumsum(percent)
columns = ['valeur propre','pourcentage d\'inertie',
           'pourcentage d\'inertie cumulée']
index = ['Dim.{}'.format(x+1) for x in range(hmax)]
Eigen = pd.DataFrame(np.transpose([mcaModel.L,percent,cumpercent]),
                      index=index,columns = columns)
Eigen.index.name = 'Dimension'
# Affichage
print(Eigen.round(3))
```

Dimension	valeur propre	pourcentage d'inertie	pourcentage d'inertie cumulée
Dim.1	0.482	28.896	28.896
Dim.2	0.385	23.084	51.981
Dim.3	0.211	12.657	64.638
Dim.4	0.158	9.453	74.091
Dim.5	0.150	9.008	83.099
Dim.6	0.123	7.398	90.497
Dim.7	0.081	4.888	95.385
Dim.8	0.046	2.740	98.125
Dim.9	0.024	1.413	99.537
Dim.10	0.008	0.463	100.000

TABLE 3.14 – Valeurs propres de l'ACM du tableau (3.2)

On peut réaliser une Analyse Factorielle des Correspondances sur le tableau disjonctif complet ou sur le tableau de Burt : on obtient les mêmes axes factoriels, même si les valeurs propres sont différentes : si λ est valeur propre du tableau disjonctif complet, alors λ^2 est valeur propre du tableau de Burt.

3.2.1 ACM via une AFC sur le tableau disjonctif complet

La représentation la plus naturelle est de passer par le tableau disjonctif complet. Constitué des valeurs positives ou nulles, ses marges lignes et colonnes ont un sens. Ses profils également en ont. En se basant des formules du tableau disjonctif complet, on peut réécrire l'équation 3.6 de la façon suivante :

$$\begin{aligned} d^2(i,j) &= \frac{I}{J} \sum_{k=1}^K \frac{1}{I^k} (y_i^k - y_j^k)^2 \\ &= \sum_{k=1}^K \frac{IJ}{I^k} (y_i^k - y_j^k)^2 \\ &= \sum_{k=1}^K \frac{1}{I^k(IJ)} \left(\frac{y_i^k/(IJ)}{1/I} - \frac{y_j^k/(IJ)}{1/I} \right)^2 \end{aligned}$$

Avec les notations du tableau de contingence introduites en analyse factorielle des correspondances appliquées au tableau disjonctif complet, on a :

$$1. f_{ik} = \frac{y_{ik}}{I \times J};$$

$$2. f_{\bullet k} = \sum_{i=1}^K \frac{y_{ik}}{I \times J} = \frac{I_k}{I \times J};$$

$$3. f_{i\bullet} = \sum_{k=1}^K \frac{y_{ik}}{I \times J}$$

A partir de là, on reconnaît la formule de la distance du χ^2 entre les profils lignes i et j calculés sur le tableau disjonctif complet :

$$d_{\chi^2}^2(i,j) = \sum_{k=1}^K \frac{1}{f_{\bullet k}} \left(\frac{f_{ik}}{f_{\bullet k}} - \frac{f_{jk}}{f_{\bullet k}} \right)^2 \quad (3.21)$$

Réiproquement, en réécrivant l'équation 3.12, la distance (au carré) entre deux modalités k et l devient :

$$\begin{aligned} d^2(k,l) &= I \sum_{i=1}^I \left(\frac{y_i^k}{I^k} - \frac{y_i^l}{I^l} \right)^2 \\ &= \sum_{i=1}^I \frac{1}{1/I} \left(\frac{y_i^k/(I \times J)}{I^k/(I \times J)} - \frac{y_i^l/(I \times J)}{I^l/(I \times J)} \right)^2 \end{aligned}$$

On reconnaît la distance du χ^2 entre les profils colonnes k et l calculés sur le tableau disjonctif complet :

$$d_{\chi^2}^2(k,l) = \sum_{i=1}^I \frac{1}{f_{i \bullet}} \left(\frac{f_{ik}}{f_{\bullet k}} - \frac{f_{il}}{f_{\bullet l}} \right)^2 \quad (3.22)$$

On voit donc que les conditions sont réunies pour pouvoir appliquer l'algorithme de l'analyse factorielle des correspondances. Nous analysons donc par ce biais : Les relations entre les modalités, les proximités entre les individus et les associations individus-modalités.

Nous mettons en œuvre une AFC sur le tableau disjonctif complet en utilisant la méthode **CA** du module « fanalysis ».

```
# ACM via une AFC sur le TDC
from fanalysis.ca import CA

mcaModelCA1 = CA(row_labels=dummies.index.values, col_labels=dummies.columns.values)
mcaModelCA1.fit(dummies.values)
# Valeurs propres
EigenCA1 = pd.DataFrame(mcaModelCA1.eig_.T[:,0], columns = ['lambda'])
EigenCA1.index +=1
EigenCA1.T.round(3)
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lambda	0.482	0.385	0.211	0.158	0.15	0.123	0.081	0.046	0.024	0.008	0.0	0.0	0.0	0.0	0.0

TABLE 3.15 – Valeurs propres issues de l'AFC sur le TDC

Remarque

L'AFC ne sait pas que nous lui avons passé une variante très particulière d'un tableau de contingence, comprenant de nombreuses redondances. C'est la raison pour laquelle il produit des facteurs en trop qui ont une variance nulle. En dehors de cela, les valeurs propres fournies sont exactement celles de l'ACM obtenues à l'aide de l'objet MCA de « mca ». Cette approche est certainement la plus simple pour obtenir les résultats de l'ACM si nous ne disposons pas d'une implémentation dédiée dans l'outil logiciel que nous utilisons.

Dans une AFC sur le tableau disjonctif complet, on construit un nuage des individus, qui est la nuage des profils lignes de l'AFC, muni de la distance du χ^2 . Par définition de cette distance, un individu possédant des modalités rares sera éloigné des autres. Deux individus sont proches s'ils ont le plus souvent les mêmes modalités pour l'ensemble des variables.

3.2.2 ACM via une AFC sur le tableau de Burt

Une seconde voie consiste à travailler à partir du tableau de Burt. Il s'agit d'un tableau de contingence constitué du croisement de l'ensemble des variables. Il est symétrique. Sur les blocs diagonaux, nous avons le croisement des variables avec elles-mêmes.

```
# Tableau de Burt
```

```
burt = dummys.T.dot(dummys)
print(burt)
```

modalité	Taille+	Taille++	Taille-	Poids+	Poids++	Poids-	Veloc+	Veloc++	Veloc-	Intell+	Intell++	Intell-	Affec+	Affec-	Agress+	Agress-
Taille+	5	0	0	4	0	1	4	0	1	4	1	0	5	0	2	3
Taille++	0	15	0	10	5	0	2	9	4	6	4	5	3	12	9	6
Taille-	0	0	7	0	0	7	2	0	5	3	1	3	6	1	2	5
Poids+	4	10	0	14	0	0	6	8	0	7	4	3	7	7	6	8
Poids++	0	5	0	0	5	0	0	1	4	2	1	2	0	5	4	1
Poids-	1	0	7	0	0	8	2	0	6	4	1	3	7	1	3	5
Veloc+	4	2	2	6	0	2	8	0	0	5	2	1	6	2	3	5
Veloc++	0	9	0	8	1	0	0	9	0	3	3	3	3	6	5	4
Veloc-	1	4	5	0	4	6	0	0	10	5	1	4	5	5	5	5
Intell+	4	6	3	7	2	4	5	3	5	13	0	0	9	4	5	8
Intell++	1	4	1	4	1	1	2	3	1	0	6	0	3	3	3	3
Intell-	0	5	3	3	2	3	1	3	4	0	0	8	2	6	5	3
Affec+	5	3	6	7	0	7	6	3	5	9	3	2	14	0	5	9
Affec-	0	12	1	7	5	1	2	6	5	4	3	6	0	13	8	5
Agress+	2	9	2	6	4	3	3	5	5	5	3	5	5	8	13	0
Agress-	3	6	5	8	1	5	5	4	5	8	3	3	9	5	0	14

TABLE 3.16 – Tableau de Burt du

On remarque également, qu'ici, la matrice est constituée des valeurs positives ou nulles, et donc les marges et les profils sont interprétables. Les conditions sont réunies pour pouvoir appliquer une AFC. A priori, nous analysons principalement les relations entre les modalités qui sont à la fois en ligne et en colonne du tableau avec l'AFC. Il est possible de revenir sur les individus grâce aux relations de transition.

Attention, les duplications étant multiples dans le tableau de Burt, les individus sont comptabilisés plusieurs fois, il faudra corriger les résultats de l'AFC.

```
# ACM via une AFC sur le tableau de Burt
```

```
mcaModelCA2 = CA(row_labels=burt.index.values,col_labels=burt.columns.values)
mcaModelCA2.fit(burt.values)
```

Pour obtenir les vraies valeurs propres de l'analyse des correspondances multiples, une correction doit être apportée sur celles fournies par l'analyse factorielle des correspondances sur le tableau de Burt. En effet, si δ_α est la valeur propres de l'AFC sur le tableau de Burt, alors celle de l'ACM correspond à :

$$\lambda_\alpha = \sqrt{\delta_\alpha} \quad \forall \alpha \quad (3.23)$$

```

# Valeurs propres
EigenCA2 = pd.DataFrame(np.transpose([mcaModelCA2.eig_.T[:,0],
                                         np.sqrt(mcaModelCA2.eig_.T[:,0])]),
                           columns = ['Delta', 'Lambda'])
EigenCA2.index +=1
print(EigenCA2.round(3))

```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Delta	0.232	0.148	0.045	0.025	0.023	0.015	0.007	0.002	0.001	0.000	0.0	0.0	0.0	0.0	0.0
Lambda	0.482	0.385	0.211	0.158	0.150	0.123	0.081	0.046	0.024	0.008	0.0	0.0	0.0	0.0	0.0

TABLE 3.17 – Valeurs propres issues de l’AFC sur le tableau de Burt

Remarque :

- Si $B_\alpha(k)$ est la coordonnées de la modalité k issue de l’AFC sur le tableau de Burt, alors celle de l’ACM est obtenue par le ratio :

$$G_\alpha(k) = \frac{1}{\sqrt{\lambda_\alpha}} B_\alpha(k), \quad \forall \alpha \quad (3.24)$$

- Les coordonnées des individus sont obtenues grâce aux relations de transition.

3.2.3 ACM via une ACP sur le tableau des profils

La troisième approche consiste à appliquer une ACP sur le tableau des profils. Nous faisons le choix des profils lignes ici, mais l’opération pour les profils colonnes est également possible.

On a vu qu’en ACM, la distance du χ^2 entre 2 individus i et j s’écrit :

$$d_{ACM}^2(i,j) = \sum_k \frac{p^k}{J} (x_i^k - x_j^k)^2 = \frac{1}{J} \sum_k \frac{1}{p^k} (y_i^k - y_j^k)^2$$

avec $p_k = I_k/I$ où I_k est le nombre d’individu possédant la modalité k . En réécrivant la formule de distance, on a :

$$d_{ACM}^2(i,l) = \frac{1}{J} \sum_k \frac{1}{\frac{I^k}{I}} (y_i^k - y_l^k)^2 = \sum_k^K \frac{1}{\frac{I^k}{I \times J}} \left(\frac{y_i^k}{J} - \frac{y_l^k}{J} \right)^2$$

On a vu qu’en ACP normée, la distance euclidienne pondérée entre deux individus i et j pouvait s’écrire :

$$d_{ACP}^2(i,j) = \sum_k \frac{1}{(\sigma^k)^2} (y_i^k - y_j^k)^2$$

où σ_k^2 est la variance de la k-ième colonne des indicatrices. Son expression peut être simplifiée¹ :

$$(\sigma^k)^2 = p^k(1-p^k) = \frac{I^k}{I} \left(1 - \frac{I^k}{I} \right) = \frac{I_k(I-I_k)}{I^2}$$

1. y_i^k suit une loi de Bernoulli de paramètre p^k

Pour qu'il y ait équivalence avec l'Analyse des Correspondances Multiples, il faudrait pondérer la $k^{\text{ème}}$ indicatrice avec u_k où :

$$u_k = \frac{I - I_k}{I \times J} \quad (3.25)$$

On peut dès lors obtenir les résultats de l'ACM via un programme d'ACP en appliquant cette procédure sur les indicatrices :

$$d_{ACP \Rightarrow ACM}^2(i, l) = \sum_k \frac{u_k}{(\sigma^k)^2} (y_{ik} - y_{lk})^2 \quad (3.26)$$

Nous calculons les profils lignes à partir du tableau disjonctif complet :

$$\forall i, \quad PL(i) = \begin{cases} 1/J & \text{si } y_{ik} = 1 \\ 0 & \text{sinon} \end{cases}$$

Tableau de des profils lignes

```
rowprofil = dummies.apply(lambda x:x/np.sum(x), axis = 1)
#Affichage
rowprofil.round(2)
```

chien	Taille+	Taille++	Taille-	Poids+	Poids++	Poids-	Veloc+	Veloc++	Veloc-	Intell+	Intell++	Intell-	Affec+	Affec-	Agress+	Agress-								
Beauceron	0.00	0.17	0.00	0.17	0.00	0.00	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00								
Basset	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00							
Berger All	0.00	0.17	0.00	0.17	0.00	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00						
Boxer	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00						
Bull-Dog	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00					
Bull-Mastif	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00					
Caniche	0.00	0.00	0.17	0.00	0.00	0.17	0.17	0.00	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00					
Chihuahua	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.00	0.00	0.17	0.00					
Cocker	0.17	0.00	0.00	0.00	0.00	0.17	0.00	0.00	0.17	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00				
Colley	0.00	0.17	0.00	0.17	0.00	0.00	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00			
Dalmatiens	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00		
Doberman	0.00	0.17	0.00	0.17	0.00	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.00		
Dogue All	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.00		
Epag. Breton	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00		
Epag. Français	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00		
Fox-Hound	0.00	0.17	0.00	0.17	0.00	0.00	0.00	0.17	0.00	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.00		
Fox-Terrier	0.00	0.00	0.17	0.00	0.00	0.17	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.00		
Gd Bleu Gasc	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.00		
Labrador	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00		
Levrier	0.00	0.17	0.00	0.17	0.00	0.00	0.00	0.17	0.00	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	
Mastiff	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	
Pekinois	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	
Pointer	0.00	0.17	0.00	0.17	0.00	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	
St-Bernard	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.00	0.17	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.17	0.00	0.17	0.00	0.17	0.00	
Setter	0.00	0.17	0.00	0.17	0.00	0.00	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00
Teckel	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00
Terre-Neuve	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.00	0.17	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.17	0.00

TABLE 3.18 – Profils lignes calculés sur le TDC

Nous réduisons les variables par l'écart type. Pour cela nous créons une fonction de réduction qu'on appliquera à la matrice des profils lignes.

```
# Réduire les variables par les écarts types
def StandardScaler(x):
    return (x/x.std(ddof=0))
```

```
rowprofil_sc = rowprofil.transform(StandardScaler)
rowprofil_sc.round(2)
```

chien	Taille+	Taille++	Taille-	Poids+	Poids++	Poids-	Véloc+	Véloc++	Véloc-	Intell+	Intell++	Intell-	Affec+	Affec-	Agress+	Agress-
Beauceron	0.00	2.01	0.00	2.0	0.00	0.00	0.00	2.12	0.00	2.0	0.00	0.00	2.0	0.0	2.0	0.0
Basset	0.00	0.00	2.28	0.0	0.00	2.19	0.00	0.00	2.07	0.0	0.00	2.19	0.0	2.0	0.0	0.0
Berger All	0.00	2.01	0.00	2.0	0.00	0.00	0.00	2.12	0.00	0.0	2.41	0.00	2.0	0.0	2.0	0.0
Boxer	2.57	0.00	0.00	2.0	0.00	0.00	2.19	0.00	0.00	2.0	0.00	0.00	2.0	0.0	2.0	0.0
Bull-Dog	0.00	0.00	2.28	0.0	0.00	2.19	0.00	0.00	2.07	2.0	0.00	0.00	2.0	0.0	0.0	2.0
Bull-Mastif	0.00	2.01	0.00	0.0	2.57	0.00	0.00	0.00	2.07	0.0	2.41	0.00	0.0	2.0	2.0	0.0
Caniche	0.00	0.00	2.28	0.0	0.00	2.19	2.19	0.00	0.00	0.0	2.41	0.00	2.0	0.0	0.0	2.0
Chihuahua	0.00	0.00	2.28	0.0	0.00	2.19	0.00	0.00	2.07	0.0	0.00	2.19	2.0	0.0	0.0	2.0
Cocker	2.57	0.00	0.00	0.0	0.00	2.19	0.00	0.00	2.07	2.0	0.00	0.00	2.0	0.0	2.0	0.0
Colley	0.00	2.01	0.00	2.0	0.00	0.00	0.00	2.12	0.00	2.0	0.00	0.00	2.0	0.0	0.0	2.0
Dalmatien	2.57	0.00	0.00	2.0	0.00	0.00	2.19	0.00	0.00	2.0	0.00	0.00	2.0	0.0	0.0	2.0
Doberman	0.00	2.01	0.00	2.0	0.00	0.00	0.00	2.12	0.00	0.0	2.41	0.00	0.0	2.0	2.0	0.0
Dogue All	0.00	2.01	0.00	0.0	2.57	0.00	0.00	2.12	0.00	0.0	0.00	2.19	0.0	2.0	2.0	0.0
Epag. Breton	2.57	0.00	0.00	2.0	0.00	0.00	2.19	0.00	0.00	0.0	2.41	0.00	2.0	0.0	0.0	2.0
Epag. Français	0.00	2.01	0.00	2.0	0.00	0.00	2.19	0.00	0.00	2.0	0.00	0.00	0.0	2.0	0.0	2.0
Fox-Hound	0.00	2.01	0.00	2.0	0.00	0.00	0.00	2.12	0.00	0.0	0.00	2.19	0.0	2.0	2.0	0.0
Fox-Terrier	0.00	0.00	2.28	0.0	0.00	2.19	2.19	0.00	0.00	2.0	0.00	0.00	2.0	0.0	2.0	0.0
Gd Bleu Gasc	0.00	2.01	0.00	2.0	0.00	0.00	2.19	0.00	0.00	0.0	0.00	2.19	0.0	2.0	2.0	0.0
Labrador	2.57	0.00	0.00	2.0	0.00	0.00	2.19	0.00	0.00	2.0	0.00	0.00	2.0	0.0	0.0	2.0
Lевrier	0.00	2.01	0.00	2.0	0.00	0.00	0.00	2.12	0.00	0.0	0.00	2.19	0.0	2.0	0.0	2.0
Mastiff	0.00	2.01	0.00	0.0	2.57	0.00	0.00	0.00	2.07	0.0	0.00	2.19	0.0	2.0	2.0	0.0
Pekinois	0.00	0.00	2.28	0.0	0.00	2.19	0.00	0.00	2.07	0.0	0.00	2.19	2.0	0.0	0.0	2.0
Pointer	0.00	2.01	0.00	2.0	0.00	0.00	0.00	2.12	0.00	0.0	2.41	0.00	0.0	2.0	0.0	2.0
St-Bernard	0.00	2.01	0.00	0.0	2.57	0.00	0.00	0.00	2.07	2.0	0.00	0.00	0.0	2.0	2.0	0.0
Setter	0.00	2.01	0.00	2.0	0.00	0.00	0.00	2.12	0.00	2.0	0.00	0.00	0.0	2.0	0.0	2.0
Teckel	0.00	0.00	2.28	0.0	0.00	2.19	0.00	0.00	2.07	2.0	0.00	0.00	2.0	0.0	0.0	2.0
Terre-Neuve	0.00	2.01	0.00	0.0	2.57	0.00	0.00	0.00	2.07	2.0	0.00	0.00	0.0	2.0	0.0	2.0

TABLE 3.19 – Profils lignes standardisés

Nous calculons la pondération des modalités (u_k).

```
# Pondération par les modalités
modpond = (I - rowMarge)/(I*J)
print(modpond.round(2))
```

	Taille+	Taille++	Taille-	Poids+	Poids++	Poids-	Véloc+	Véloc++	Véloc-	Intell+	Intell++	Intell-	Affec+	Affec-	Agress+	Agress-
pond	0.14	0.07	0.12	0.08	0.14	0.12	0.12	0.11	0.10	0.09	0.13	0.12	0.08	0.09	0.09	0.08

TABLE 3.20 – Pondération des modalités

Nous l'appliquons aux données en multipliant par la racine carré ².

```
# Applique la pondération au profil
rowprofil_pond = rowprofil_sc*np.sqrt(modpond.values)
```

Il ne nous reste plus qu'à faire appel à l'ACP avec la classe PCA de « fanalysis ». Nous utilisons une ACP non normée (`std_unit = False`) puisque les données ont été explicitement préparées en amont.

2. Cette manipulation n'est pas nécessaire si l'ACP implémentée sait prendre en compte les pondérations des variables.

```

# ACM via une ACP sur le tableau des profils
from fanalysis.pca import PCA
mcaModelPCA = PCA(std_unit=False, row_labels=dummies.index,
                   col_labels=dummies.columns) #n_components = hmax,
mcaModelPCA.fit(rowprofil_pond.values)
# Valeurs propres
EigenPCA= pd.DataFrame(mcaModelPCA.eig_.T[:,0],columns = ['lambda'])
EigenPCA.index +=1
EigenPCA.T.round(3)

```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
lambda	0.482	0.385	0.211	0.158	0.15	0.123	0.081	0.046	0.024	0.008	0.0	0.0	0.0	0.0	0.0	0.0

TABLE 3.21 – Valeurs propres issues de l'ACP sur le tableau des profils

Nous avons les bonnes valeurs propres. L'Analyse en Composantes Principales ne sait pas qu'il y a des redondances artificielles dans les données, d'où les facteurs aux variances nulles en excéderent.

3.2.4 Inertie et pourcentage d'inertie en ACM

L'inertie d'un axe en Analyse des Correspondances Multiples est particulière car elle est égale à la moyenne des carrés des rapports de corrélations entre l'axe et les variables. Cette propriété valide l'interprétation des facteurs de l'ACM en tant que variable synthétique. Les facteur de l'ACM sont des variables quantitatives qui synthétisent les variables qualitatives. Les rapports de corrélation sont compris entre 0 et 1, la moyenne des rapports de corrélation et donc l'inertie sera toujours comprise entre 0 et 1 :

$$\lambda_\alpha = \frac{1}{J} \sum_{j=1}^J \eta^2(F_\alpha, V_j)$$

λ_α est la moyenne des carrés des rapports de corrélation. Nous visualisons les valeurs propres.

```

# Visualisation des valeurs propres
def screeplot(data,choice=None,figsize=None):
    #
    p = data.shape[0]
    fig,axes = plt.subplots(figsize = figsize); axes.grid()
    axes.set_xlabel('Dimensions',fontsize=14)
    axes.set_title('Scree plot',fontsize=14)
    axes.set_xticks([x for x in range(1,p+1)])
    if choice is None or choice=='scree plot':
        eigen = data.iloc[:,0].round(2)
        ylim = np.max(eigen)+0.05
        axes.set_ylim(0,ylim)
        axes.bar(np.arange(1,p+1),eigen.values,width=0.9)
        axes.plot(np.arange(1,p+1),eigen.values,c="black")
        axes.set_ylabel('Eigenvalue',fontsize=13)
        ## Add text
        for i in range(p):

```

```

        axes.scatter(i+1,eigen.values[i],color='black',alpha=1)
        axes.text(i+.75,0.01+eigen.values[i],str(eigen.values[i]),
                  color = 'black')
    elif choice == "percentage":
        percent = data.iloc[:,1].round()
        axes.set_ylim(0,100)
        axes.bar(np.arange(1,p+1),percent.values,width=0.9)
        axes.plot(np.arange(1,p+1),percent.values,c="black")
        axes.set_ylabel('Percentage of variance',fontsize=13)
        ## Add text
        for i in range(p):
            axes.scatter(i+1,percent.values[i],color='black',alpha=1)
            axes.text(i+.6,0.01+percent.values[i],f'{percent.values[i}%',color = 'black',fontweight='bold',fontsize=12)
    elif choice == "cumulative":
        cumul = data.iloc[:,2].round()
        axes.set_ylim(0,105)
        axes.bar(np.arange(1,p+1),cumul.values,width=0.9)
        axes.set_ylabel('Cumulative percentage of variance',fontsize=13)
    plt.show()
# Affichage
screeplot(data=Eigen,figsize=(9,7))

```

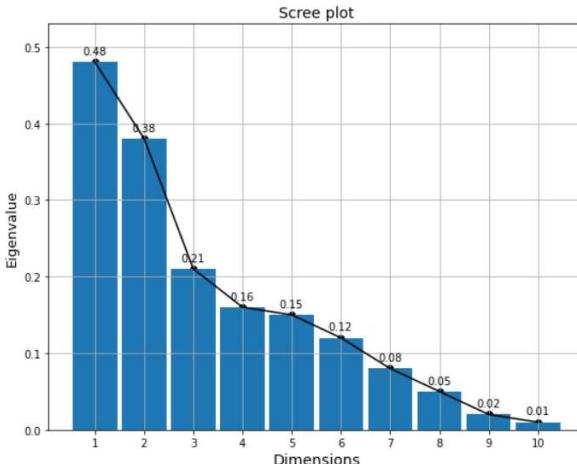


FIGURE 3.4 – Histogramme des valeurs propres

Si on s'intéresse au **pourcentage d'inertie**, ils sont généralement faibles en ACM ou en AFC, car les individus évoluent dans un espace de dimension élevé \mathbb{R}^{K-J} , d'autant plus élevé que le nombre de modalité par variable est plus grand. En effet, pour un axes λ , le pourcentage d'inertie est :

$$\frac{\lambda_\alpha}{\sum_{t=1}^{K-J} \lambda_t} \times 100 \leq \frac{1}{J} \times 100 \leq \frac{J}{K-J} \times 100$$

On remarque que l'inertie moyenne est égale à 1 sur le nombre de variable. En effet :

$$\frac{1}{K-J} \sum_{t=1}^{K-J} \lambda_t = \frac{1}{K-J} \left(\frac{K}{J} - 1 \right)$$

Cette valeur peut aider à décider combien de dimensions interpréter en ACM (on évitera d'interpréter les dimensions qui ont une inertie inférieure à $1/J$). Ce critère conduit à ne retenir que trois axes, cependant, le diagramme des valeurs montre une chute après λ_2 . On interprétera donc uniquement les deux premiers axes.

3.3 Représentation des nuages N_I et N_K

En ACM, comme dans toute analyse factorielle, on construit deux nuages de points : le nuage des lignes (individus) et le nuage des colonnes (variables).

3.3.1 Nuage N_I des individus

Les coordonnées factorielles des individus sont directement accessibles grâce à l'attribut `.fs_r`.

```
# Coordonnées des individus
rowcoord = pd.DataFrame(mcaModel.fs_r(N=hmax), index=donnee.index,
                         columns = index)
print(rowcoord.iloc[:, [0,1]].round(3))
```

On positionne nos individus dans sur le premier plan factoriel de l'ACM.

```
# Fonction de visualisation en 2D
def mca_plot(data,eigen,axe1,axe2,main,figsize=None):
    try:
        if axe1==axe2:
            raise ValueError('Erreur: axe1 doit être différent de axe2.')
        elif axe1>axe2:
            raise ValueError('Erreur: axe1 doit être inférieur à axe2.')
        elif axe1<0 or axe2<0:
            msg = 'Erreur: les valeurs des axes doivent être positives ou nulles.'
            raise ValueError(msg)
        else:
            # set limite
            n = data.shape[0]
            # Valeurs propres
            percent = np.array([100*x/sum(eigen) for x in eigen])
            dim1 = round(percent[axe1],2); dim2 = round(percent[axe2],2)

    # Graphique
```

```

fig, axes = plt.subplots(figsize = figsize); axes.grid()
axes.axis([-1.5,1.5,-1.5,1.5])
axes.set_title(main)
axes.set_xlabel(f"Dim.{1+axe1} ({dimi}%)")
axes.set_ylabel(f"Dim.{1+axe2} ({dimj}%)")
for i in range(n):
    plt.scatter(data.iloc[i,axe1], data.iloc[i,axe2],
                c = "black", alpha = 1)
    axes.text(data.iloc[i,axe1],data.iloc[i,axe2],data.index[i],
              color = "black", fontsize = 11)
plt.axhline(0, color='blue',linestyle="--", linewidth=0.5)
plt.axvline(0, color='blue',linestyle="--", linewidth=0.5)
plt.show()

# if false then raise the value error
except ValueError as e:
    print(e)

# Nuage des individus sur les axes 1 et 2
mca_plot(data=rowcoord,eigen=eigenvalue,axe1=0,axe2=1,
          main= 'Projection des individus',figsize=(12,8))

```

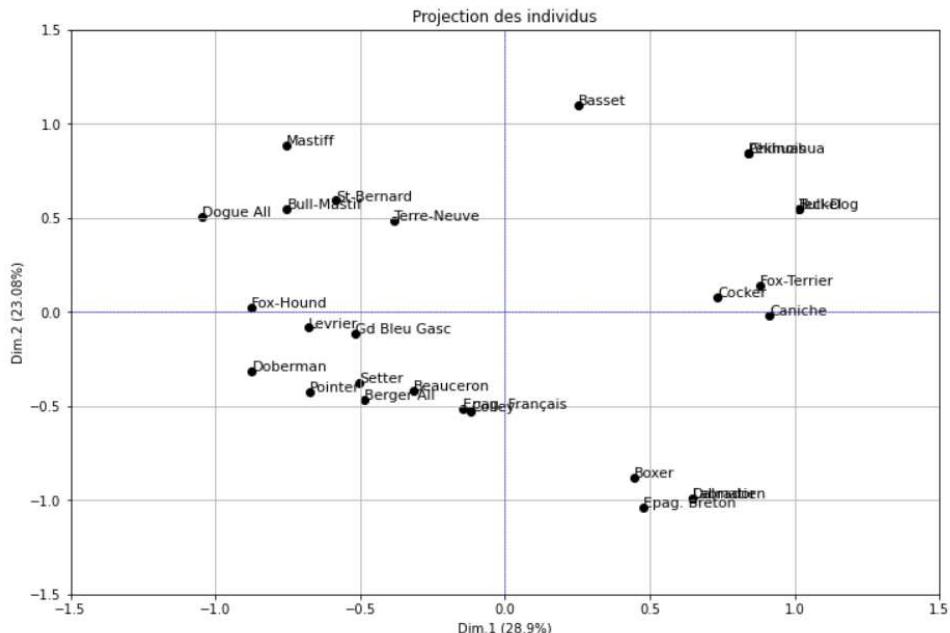
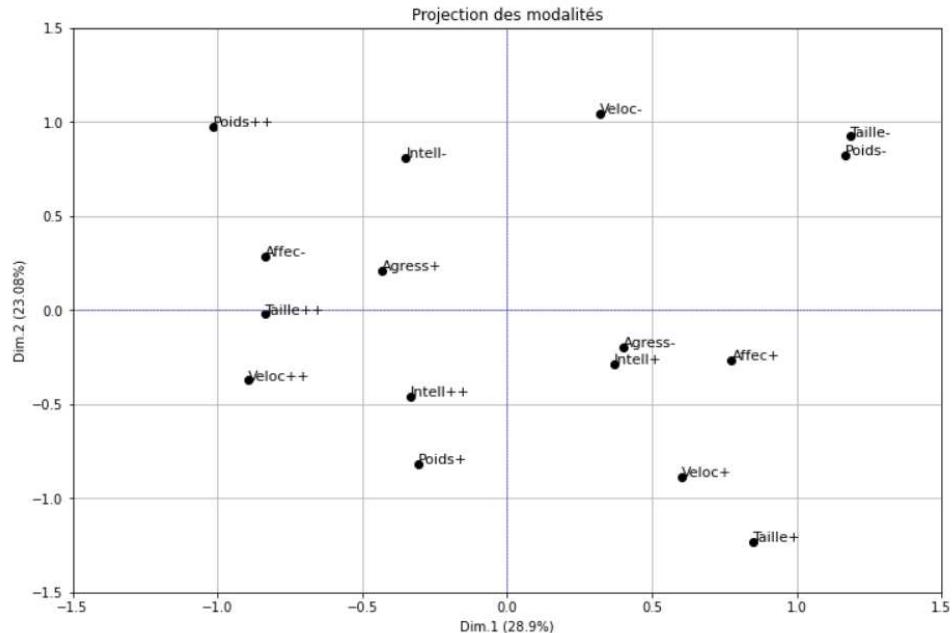


FIGURE 3.5 – Carte de races canines

3.3.2 Nuage N_K des modalités

```
# Coordonnées des modalités
modcoord = pd.DataFrame(mcaModel.fs_c(N=hmax), index=dummies.columns,
                         columns = index)
print(modcoord.iloc[:, [0,1]].round(3))

# Nuage des modalités sur les axes 1 et 2
mca_plot(data=modcoord,eigen=eigenvalue,axe1=0,axe2=1,
          main= 'Projection des modalités',figsize=(12,8))
```



3.3.3 Représentation simultanée

En ACM comme en AFC, on peut représenter simultanément les lignes et les colonnes sur un même graphique, c'est-à-dire les individus et les modalités. En ACM, on a deux nuages : le nuage des individus N_I donc le centre de gravité G_I est confondu avec l'origine des axes et le nuage des modalités N_K qui lui aussi a son centre de gravité G_K confondu avec l'origine des axes. A ces deux nuages on applique une analyse factorielle, c'est-à-dire une projection sur une suite d'axes orthogonaux d'inertie maximum. Lorsqu'on applique l'analyse factorielle sur le nuage des individus N_I , on obtient un plan constitué par les deux premiers axes notés F_1 et F_2 . Sur ces axes, on retrouve les individus i et j . Le nuage ayant l'origine comme centre de gravité G_I . On applique aussi l'analyse factorielle au nuage des modalités N_K et on obtient un plan constitué aussi par les deux premiers axes appelés G_1 et G_2 . On reconnaît les modalités k et l . Le nuage a pour centre de gravité G_K l'origine des axes. Ces deux sont superposés et l'on

obtient une représentation simultanée des ligne i et j et des modalités k et l . Chacun des deux nuages, le nuage ligne et le nuage colonne, a son centre de gravité confondu avec l'origine des axes.

Les règles d'interprétation de la relation simultanée sont constituées par deux propriétés barycentriques.

La première propriété barycentrique s'énonce ainsi : « A un coefficient près $(1/\sqrt{\lambda_\alpha})$ (le même pour tous les individus), un individu est au barycentre des modalités qu'il possède ». En d'autres termes, un individu est à côté des modalités qu'il possède et à l'opposé des modalités qu'il ne possède pas.

La deuxième propriété barycentrique s'énonce comme suit : « A un coefficient près $(1/\sqrt{\lambda_\alpha})$ (le même pour toutes les modalités), une modalité est au barycentre des individus qui la possède ». On a une propriété qui s'exprime de la même manière que la précédente. Cette symétrie entre les deux propriétés fait que l'on parle de double propriété barycentrique : « une modalité est au barycentre des individus qui la possèdent ; un individu est au barycentre des modalités qu'il possède ».

```
# Nuage simultané
def biplot(data1,data2,eigen,axe1,axej,figsize=None):
    # Représentation simultanée des individus et des variables
    try:
        if axe1==axej:
            raise ValueError('Erreur: axe1 doit être différent de axej.')
        elif axe1>axej:
            raise ValueError('Erreur: axe1 doit être inférieur à axej.')
        elif axe1<0 or axej<0:
            msg = 'Erreur: les valeurs des axes doivent être positives ou nulles.'
            raise ValueError(msg)
        else:
            n = data1.shape[0];p=data2.shape[0]
            # Valeurs propres
            percent = np.array([100*x/sum(eigen) for x in eigen])
            dimi = round(percent[axe1],2); dimj = round(percent[axej],2)

            # Biplot
            fig = plt.figure(figsize=figsize)
            axes1 = fig.add_subplot(111)
            axes2 = axes1.twiny()
            axes2 = axes2.twinx()
            axes1.grid()
            axes2.axis([-1.5,1.5,-1.5,1.5])
            axes1.axis([-1.5,1.5,-1.5,1.5])
            axes1.set_title("Biplot")
            axes1.set_xlabel(f"Dim.{1+axe1} ({dimi}%)")
            axes1.set_ylabel(f"Dim.{1+axej} ({dimj}%)")
            # Affichage des individus
            for i in range(n):
                axes1.scatter(data1.iloc[i,0],data1.iloc[i,1],
                            c = "blue",alpha = 1,marker="^")
```

```

        axes1.text(data1.iloc[i,0]-0.10,data1.iloc[i,1]+0.01,
                    data1.index[i],color = "blue",fontsize=12)
    # Affichage des variables
    for k in range(p):
        axes2.scatter(data2.iloc[k,0],data2.iloc[k,1],
                      color='red',alpha=1,marker="s")
        axes2.text(data2.iloc[k,0]-0.10,data2.iloc[k,1]+0.01,
                    data2.index[k],color = "red",fontsize=12)
    plt.axhline(0, color='blue',linestyle="--", linewidth=0.5)
    plt.axvline(0, color='blue',linestyle="--", linewidth=0.5)
    plt.show()

except ValueError as e:
    print(e)

# Affichage
biplot(data1=rowcoord,data2=modcoord,eigen=eigenvalue,axe1=0,axej=1,
       figsize=(16,9))

```

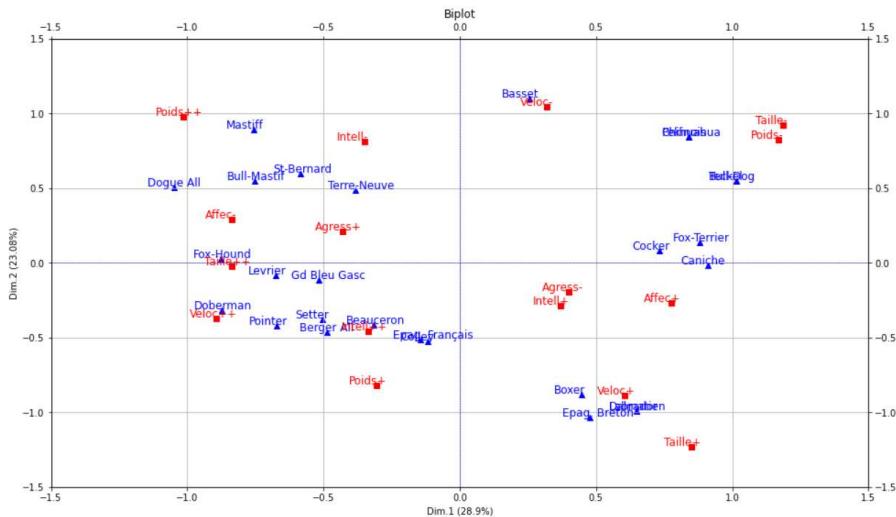


FIGURE 3.7 – Carte de races canines avec leurs caractéristiques

L'axe 1 oppose les chiens de petite taille et affectueux aux chiens de grande taille, très rapides et agressifs. L'axe 2 oppose les chiens de taille moyenne, très intelligents à des chiens très lents et peu intelligents.

3.3.4 Relation de transition

A l'instar de l'analyse factorielle des correspondances, il est possible d'obtenir les coordonnées des modalités (colonnes) à partir de celles des individus (lignes) et inversement.

Pour les coordonnées des individus ($F_\alpha(i)$) à partir des coordonnées des modalités ($G_\alpha(k)$), la relation s'écrit :

$$F_\alpha(i) = \frac{1}{\sqrt{\lambda_\alpha}} \sum_{j=1}^J \sum_{k=1}^{K_j} \frac{y_{ik}}{J} G_\alpha(k) \quad (3.27)$$

On note que y_{ik}/J correspond au profil ligne de l'individu i . Cette relation de transition traduit la première relation barycentrique.

```
# Coordonnées de l'individu i
transition1 = rowprofil.dot(modcoord)/np.sqrt(eigenvalue)
print(transition1.iloc[:,[0,1]].round(3))
```

Inversement, il est possible d'obtenir les coordonnées des modalités (G_α) à partir de celles des individus (F_α).

$$G_\alpha(k) = \frac{1}{\sqrt{\lambda_\alpha}} \sum_{i=1}^I \frac{y_{ik}}{I_k} F_\alpha(i) \quad (3.28)$$

Cette seconde relation traduit le deuxième relation propriété barycentrique.

```
# Coordonnée de la modalité k
transition2 = dummiessweight.T.dot(rowcoord)/np.sqrt(eigenvalue)
print(transition2.iloc[:,[0,1]].round(3))
```

3.3.5 Représentation simultanée barycentrique

Il existe une alternative pour faire figurer les individus et les modalités dans un même repère. On place les individus normalement, mais les modalités, on les positionne de manière à ce que leurs coordonnées correspondent réellement à la moyenne des coordonnées des individus qui leur sont rattachés ([Sap06], page 225; [HLP16], page 138). Chaque modalité est située au barycentre des individus qui leur correspondent, d'où l'appellation relation barycentrique.

1) Moyennes conditionnelles

On calcule les moyennes conditionnelles pour chaque modalité des variables qualitatives.

```
# Concaténation avec les coordonnées
concat = pd.concat([rowcoord, donnee], axis = 1)
# Moyenne conditionnelle de taille
condMeanTaille =pd.pivot_table(concat,values = index,index = ["Taille"],
                                aggfunc = "mean")
# Moyenne conditionnelle de Poids
condMeanPoids =pd.pivot_table(concat, values = index,index = ["Poids"],
                                aggfunc = "mean")
# Moyenne conditionnelle de velocite
condMeanVeloc =pd.pivot_table(concat, values = index,index = ["Velocite"],
                                aggfunc = "mean")
# Moyenne conditionnelle de Intelligence
condMeanIntell =pd.pivot_table(concat, values = index,index = ["Intelligence"],
                                aggfunc = "mean")
```

```

# Moyenne conditionnelle de Affection
condMeanAffec =pd.pivot_table(concat, values = index,index = ["Affection"],
                                aggfunc = "mean")
# Moyenne conditionnelle de Agressivite
condMeanAgress =pd.pivot_table(concat, values = index,index = ["Agressivite"],
                                aggfunc = "mean")
# Concaténation des moyennes conditionnelles
CondMean = pd.concat([condMeanTaille, condMeanPoids, condMeanVeloc,
                      condMeanIntell, condMeanAffec, condMeanAgress],
                      axis = 0)
print(CondMean.loc[:,['Dim.1','Dim.2']].round(3))

```

2) Correction des coordonnées avec des valeurs propres

Grâce aux relations de transition, il suffit en pratique de multiplier les coordonnées factorielles des points - modalités par la racine carrée de la valeur propre associée. Il s'agit donc d'une contraction des positions des modalités dans le repère puisque la valeur propre est nécessairement inférieure à 1 en ACM ($\lambda_\alpha \leq 1, \forall \alpha$).

```

# Correction des coordonnées par valeurs propres
correctcoordeig = modcoord.apply(lambda x : x*np.sqrt(eigenvalue),
                                   axis = 1)
print(correctcoordeig.iloc[:,[0,1]].round(3))

```

Approche 1			Approche 2		
modalité	Dim.1	Dim.2	modalité	Dim.1	Dim.2
Taille+	0.591	-0.764	Taille+	0.591	-0.764
Taille++	-0.581	-0.013	Taille++	-0.581	-0.013
Taille-	0.822	0.573	Taille-	0.822	0.573
Poids+	-0.212	-0.508	Poids+	-0.212	-0.508
Poids++	-0.704	0.604	Poids++	-0.704	0.604
Poids-	0.811	0.511	Poids-	0.811	0.511
Veloc+	0.419	-0.551	Veloc+	0.419	-0.551
Veloc++	-0.619	-0.231	Veloc++	-0.619	-0.231
Veloc-	0.222	0.648	Veloc-	0.222	0.648
Intell+	0.256	-0.177	Intell+	0.256	-0.177
Intell++	-0.233	-0.285	Intell++	-0.233	-0.285
Intell-	-0.242	0.502	Intell-	-0.242	0.502
Affec+	0.538	-0.166	Affec+	0.538	-0.166
Affec-	-0.580	0.178	Affec-	-0.580	0.178
Agress+	-0.299	0.130	Agress+	-0.299	0.130
Agress-	0.278	-0.120	Agress-	0.278	-0.120

TABLE 3.22 – Coordonnées des modalités corrigées

On voit que les deux approches fournissent les mêmes résultats. Réalisons une nouvelle représentation graphique.

```

# Affichage
biplot(data1=rowcoord,data2=correctcoordeig,eigen=eigenvalue,
        axei=0,axej=1,figsize=(16,9))

```

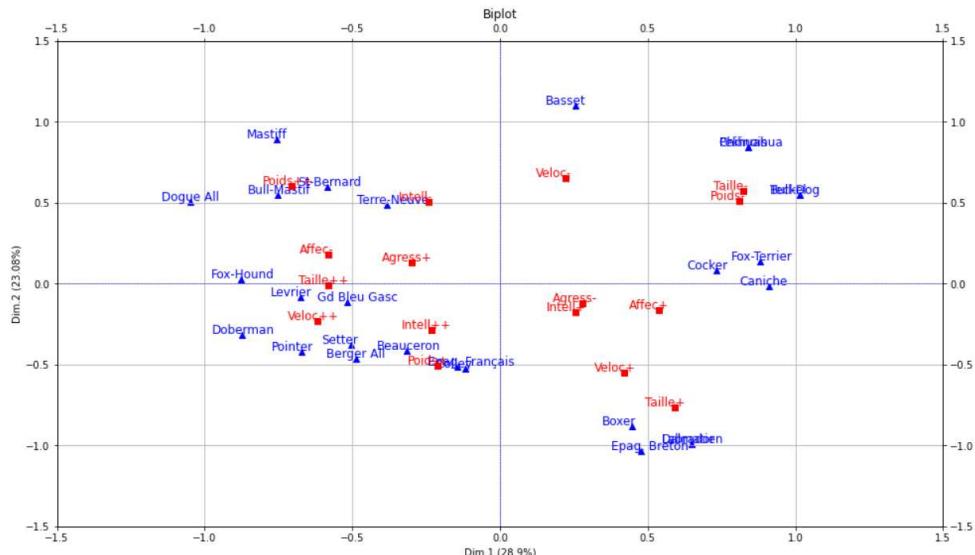


FIGURE 3.8 – Représentation barycentrique

3.4 Aide à l’interprétation des résultats

3.4.1 Analyse du point de vue des individus

1) Cosinus carré des individus

Le cosinus carré COS2 traduit la qualité de représentation des individus sur les axes factoriels. Pour un individu i , son cosinus carré sur l’axe α est donnée par :

$$\text{COS}^2_\alpha(i) = \frac{F_\alpha^2(i)}{d^2(i, G_I)} \quad (3.29)$$

```
# Cosinus carré des individus
rowcos2 = rowcoord.apply(lambda x:x**2/rowdisto.T.values, axis = 0)
print(rowcos2.iloc[:, [0, 1]].round(3))
```

On affiche graphiquement le cosinus des individus sur les axe 1 et 2.

```
# Affichage graphique des contributions
def plot_graph(data,axis,xlabel,title,figsize=None):
    p = data.shape[1]
    try:
        if axis<0 or axis>p:
            raise ValueError(f'axis doit être compris entre {0} et {p-1}.')
        else:
            sort = data.sort_values(by=f'Dim.{1+axis}', ascending=True)
            sort.iloc[:,axis].plot.barh(figsize=figsize)
            plt.xlabel(xlabel,fontsize=12)
    except:
        print("Error: Invalid axis value")
```

```

plt.title(f"[title] sur l'axe {1+axis}", fontsize=12)
plt.show()
except ValueError as f:
    print(f)

# Cosinus carré axe 1
plot_graph(data=rowcos2, axis=0, xlabel = 'Cosinus2', title = 'Rows cosinus2',
            figsize=(10,8))

# Cosinus carré axe 2
plot_graph(data=rowcos2, axis=1, xlabel = 'Cosinus', title = 'Rows cosinus2',
            figsize=(10,8))

```

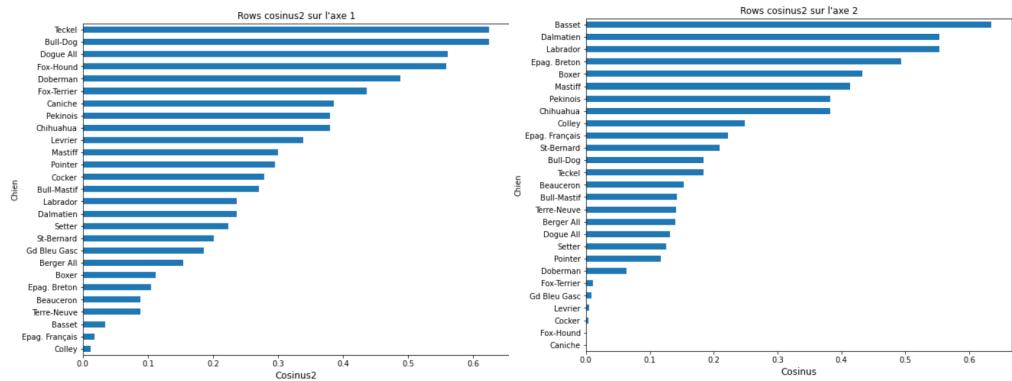


FIGURE 3.9 – Cosinus carrés des individus sur les facteurs 1 et 2

2) Contributions des individus

La contribution d'un individu à la formation d'un axe factoriel est la part d'information apportée par cet individu pour former l'axe. Elles permettent de déterminer les individus qui pèsent le plus dans la définition de chaque axe.

$$\text{CTR}_\alpha(i) = \frac{1}{I} \frac{F_\alpha^2(i)}{\lambda_\alpha} \quad (3.30)$$

```

# Contribution des individus
rowcontrib = rowcoord.apply(lambda x : 100*x**2/(I*eigenvalue), axis=1)
print(rowcontrib.iloc[:,[0,1]].round(3))

```

On affiche graphiquement les contributions des individus sur les axes 1 et 2.

```

# Contribution axe 1
plot_graph(data=rowcontrib, axis=0, xlabel = 'Contribution (%)',
            title = 'Rows contributions', figsize=(10,8))

```

```

# Contribution axe 2
plot_graph(data=rowcontrib, axis=1, xlabel = 'Contribution (%)',
            title = 'Rows contributions', figsize=(10,8))

```

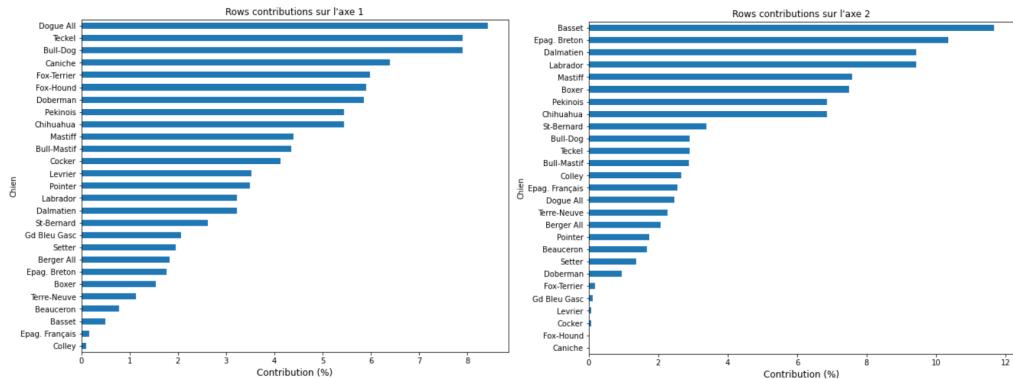


FIGURE 3.10 – Contributions des individus sur les facteurs 1 et 2

3.4.2 Analyse du point de vue des modalités

1) Cosinus carré des modalités

Les cosinus carrés (COS2) représentent la qualité de représentation des modalités sur les axes, pris individuellement ou cumulés. Ils nous donnent une indication sur l'information des modalités captée par les facteurs. Le cosinus carré d'une modalité k sur l'axe α est basé sur le carré des coordonnées, mais normalisé avec sa distance à l'origine :

$$\text{COS}_\alpha^2(k) = \frac{G_\alpha^2(k)}{d^2(k, G_K)} \quad (3.31)$$

```
# Cosinus des modalités
modcos2 = modcoord.apply(lambda x:x**2/moddisto.values, axis = 0)
print(modcos2.iloc[:,[0,1]].round(3))
```

On observe graphiquement sur les axes 1 et 2.

```
# Cosinus carré axe 1
plot_graph(data=modcos2, axis=0, xlabel = 'Cosinus2',
           title = "Cosinus carré des modalités", figsize=(10,8))

# Cosinus carré axe 2
plot_graph(data=modcos2, axis=1, xlabel = 'Cosinus',
           title = "Cosinus carré des modalités", figsize=(10,8))
```

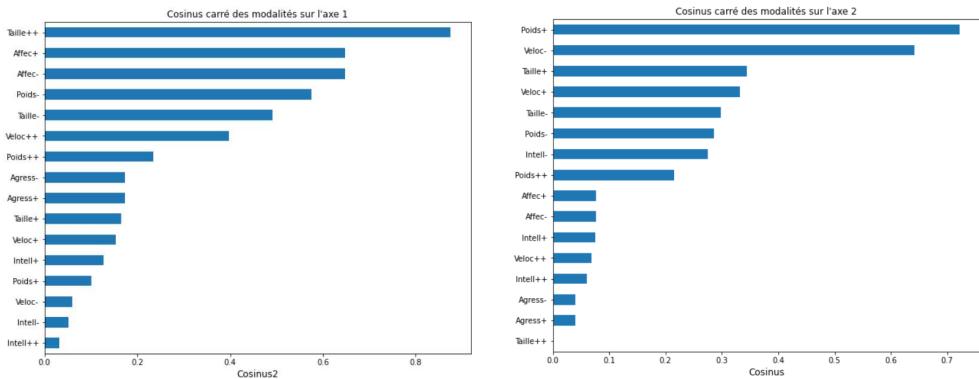


FIGURE 3.11 – Cosinus carrés des modalités sur les facteurs 1 et 2

2) Contribution des modalités

La contribution d'une modalité à la formation d'un axe est la part d'information apportée par cette modalité pour former l'axe. Ces contributions quantifient leur influence dans la construction de l'axe. En ACM, la contribution d'une modalité k à l'axe α est définie par le ratio entre le carré de la coordonnée et l'importance du facteur traduite par sa variance restituée.

$$\text{CTR}_\alpha(k) = \frac{p_k \times G_\alpha^2(k)}{\lambda_\alpha} \quad (3.32)$$

```
# Contribution des modalités
contrib = modcoord.apply(lambda x : 100*x**2/eigenvalue, axis=1)
modcontrib = contrib.apply(lambda x : x*modweight, axis=0)
print(modcontrib.iloc[:, [0, 1]].round(3))
```

Remarque

Comme un poids est associé à chaque modalité, les contributions importantes ne correspondent pas nécessairement aux points les plus éloignés sur le graphe. On fait une visualisation graphique des contributions sur les axes 1 et 2

```
# Contribution axe 1
plot_graph(data=modcontrib, axis=0, xlabel = 'Contribution (%)',
           title = 'Contributions des modalités', figsize=(10,8))

# Contribution axe 2
plot_graph(data=modcontrib, axis=1, xlabel = 'Contribution (%)',
           title = 'Contributions des modalités', figsize=(10,8))
```

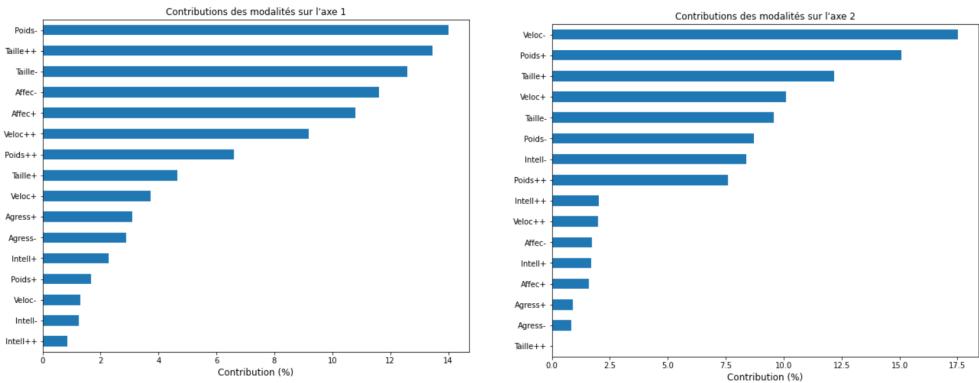


FIGURE 3.12 – Contributions des modalités sur les facteurs 1 et 2

3) Valeur test

la valeur test permet de caractériser l'intensité de l'écartement des modalités par rapport à l'origine. Son expression dérive de celle de la coordonnée, modulée par l'effectif rattachée à la modalité :

$$VT_\alpha(k) = G_\alpha(k) \sqrt{\frac{(I-1)I_k}{I - I_K}} \quad (3.33)$$

Cette valeur test est distribuée - très approximativement - selon une loi normale.

```
# Valeur test des modalités
modVtest = modcoord.apply(lambda x : x*np.sqrt(((I-1)*rowMarge)/(I-rowMarge)))
print(modVtest.iloc[:, [0,1]].round(3))
```

En vérité, par rapport aux indicateurs usuels (coordonnées, contributions, cosinus carrés), la valeur test n'apporte pas grandes choses dans le contexte spécifique de l'analyse des correspondances multiples. Plus intéressant est le rapport de corrélation. Il permet de caractériser les facteurs à l'aide des variables, avec leurs modalités prises dans un ensemble, et non plus des modalités individuellement.

3.4.3 Analyse du point de vue des variables

1) Cosinus carré des variables

Pour une variable j , son cosinus carré correspond à la somme des cosinus carrés de ses modalités :

$$\text{COS}_\alpha^2(j) = \sum_{k \in K_j} \text{COS}_\alpha^2(k) \quad (3.34)$$

```
# Cosinus carré des variables
varcos2 = pd.DataFrame(np.zeros((J,hmax), dtype=float), columns=index,
                       index=donnee.columns)
varcos2.index.name = 'Variable'
for name in donnee.columns:
    varcos2.loc[name, :] = modcos2.loc[np.unique(donnee[name]), :].sum()
print(varcos2.iloc[:, [0,1]].round(3))
```

2) Contribution des variables

Pour une variable j , la contribution à la construction d'un axe se calcule en sommant les contributions de toutes les modalités. Cette contribution est égale au rapport de corrélation au carré entre l'axe et la variable divisée par le nombre de variable.

$$CTR_\alpha(j) = \sum_{k \in K_j} CTR_\alpha(k) \quad (3.35)$$

```
# Contribution des variables
varcontrib = pd.DataFrame(np.zeros((J,hmax), dtype=float), columns=index,
                           index=donnee.columns)
varcontrib.index.name = 'Variable'
for name in donnee.columns:
    varcontrib.loc[name,:] = modcontrib.loc[np.unique(donnee[name]), :].sum()
print(varcontrib.iloc[:, [0,1]].round(3))
```

3) Rapport de corrélation

Le carré du rapport de corrélation caractérise, pour chaque variable la dispersion relative de ses modalités. Il se définit par le ratio entre la variance inter-modalités et la variance totale. Sachant que les facteurs sont centrés, et que leur variance totale équivaut à la valeur propre qui leur est associée, il s'écrit pour la variables j sur l'axe α :

$$\eta_\alpha^2(j) = \sum_{k \in K_j} \frac{I_k}{I} G_\alpha^2(k) = \sum_{k \in K_j} p_k G_\alpha^2(k) = J \times \lambda_\alpha \times CTR_\alpha(j) \quad (3.36)$$

```
# Rapport de corrélation
eta2= varcontrib.apply(lambda x : J*x*eigenvalue/100, axis=1)
print(eta2.iloc[:, [0,1]].round(3))
```

On les positionne dans une graphique.

```
# Fonction de visualisation en 2D
def corr_plot(data,eigen,axe1,axe2,main,figsize=None):
    try:
        if axe1==axe2:
            raise ValueError('Erreur: axe1 doit être différent de axe2.')
        elif axe1>axe2:
            raise ValueError('Erreur: axe1 doit être inférieur à axe2.')
        elif axe1<0 or axe2<0:
            msg = 'Erreur: les valeurs des axes doivent être positives ou nulles.'
            raise ValueError(msg)
        else:
            # set limite
            n = data.shape[0]
            # Valeurs propres
            percent = np.array([100*x/sum(eigen) for x in eigen])
            dim1 = round(percent[axe1],2); dim2 = round(percent[axe2],2)
```

```

# Graphique
fig, axes = plt.subplots(figsize = figsize); axes.grid()
axes.axis([0,+1,0,+1])
axes.set_title(main)
axes.set_xlabel(f"Dim.{1+axe1} ({dimi}%)")
axes.set_ylabel(f"Dim.{1+axe2} ({dimj}%)")
for i in range(n):
    plt.scatter(data.iloc[i,axe1], data.iloc[i,axe2],
                c = "darkcyan", alpha = 1)
    axes.text(data.iloc[i,axe1],data.iloc[i,axe2],data.index[i],
              color = "darkcyan", fontsize = 11)
plt.show()

# if false then raise the value error
except ValueError as e:
    print(e)

corr_plot(data=eta2,eigen=eigenvalue,axe1=0,axe2=1,
          main= 'Rapport de corrélation',figsize=(16,9))

```

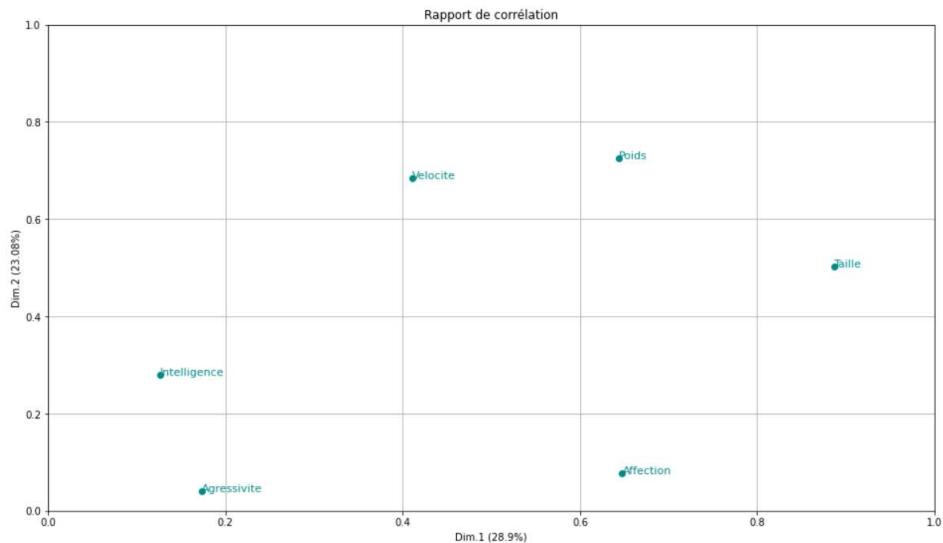


FIGURE 3.13 – Rapport de corrélation avec les axes 1 et 2

3.4.4 Éléments supplémentaires

Les propriétés barycentriques sont aussi utilisées dans le calcul des coordonnées des éléments supplémentaires (individus ou variables). Dans toute l'analyse factorielle, un élément est dit supplémentaire s'il n'est pas utilisé dans la construction des axes. Par opposition, un élément qui est utilisé dans la construction des axes est dit actif.

Un élément supplémentaire peut être soit un individu, soit une modalité. Précisons que toutes les modalités d'une même variable doivent avoir le même statut, soient être toutes actives, soient être toutes supplémentaires ; c'est pourquoi on parle de variables actives et de variables supplémentaires.

Les propriétés barycentriques servent également à calculer la position d'un individu ou d'une modalité supplémentaire.

1) Individus supplémentaires

```
# Individus supplémentaires
ind_sup = pd.read_excel('races_canines_acm.xls', sheet_name=1, header=0, index_col=0)
print(ind_sup)
```

Chien	Taille	Poids	Velocite	Intelligence	Affection	Agressivite
Medor	Taille+	Poids-	Veloc-	Intell++	Affec-	Agress+
Djeck	Taille++	Poids++	Veloc+	Intell+	Affec+	Agress-
Taico	Taille-	Poids+	Veloc++	Intell++	Affec+	Agress+
Rocky	Taille+	Poids+	Veloc+	Intell-	Affec+	Agress-
Boudog	Taille-	Poids-	Veloc++	Intell+	Affec-	Agress+
Wisky	Taille+	Poids++	Veloc-	Intell-	Affec+	Agress+

TABLE 3.23 – Individus supplémentaires

On sauvegarde dans une variable `n` le nombre de lignes de notre nouvelle base correspondant ainsi au nombre d'individus supplémentaires.

```
# Dimension
n = ind_sup.shape[0]
print(f'Nous avons {n} chiens supplémentaires.')
```

Nous avons 6 chiens supplémentaires.

Les propriétés barycentriques vont nous permettre d'obtenir les coordonnées factorielles de nos individus supplémentaires à partir de leur description, plus précisément de leur profil. Tout d'abord, transformons notre nouvelle en codage 0/1 (TDC).

```
# Codage en 0/1
dummiesSup = pd.get_dummies(ind_sup, prefix = "", prefix_sep = "")
```

chien	Taille+	Taille++	Taille-	Poids+	Poids++	Poids-	Veloc+	Veloc++	Veloc-	Intell+	Intell++	Intell-	Affec+	Affec-	Agress+	Agress-
Medor	1	0	0	0	0	1	0	0	1	0	1	0	0	1	1	0
Djeck	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1
Taico	0	0	1	1	0	0	0	1	0	0	1	0	1	0	1	0
Rocky	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0	1
Boudog	0	0	1	0	0	1	0	1	0	1	0	0	0	1	1	0
Wisky	1	0	0	0	1	0	0	0	1	0	0	1	1	0	1	0

TABLE 3.24 – TDC sur les individus supplémentaires

Ensuite calculons le profil de chacun de nos individus.

```
# Profil des individus supplémentaires
rowSupProf = dummiesSup.apply(lambda x: x/np.sum(x),axis=1)
print(rowSupProf.round(2))
```

chien	Taille+	Taille++	Taille-	Poids+	Poids++	Poids-	Veloc+	Veloc++	Veloc-	Intell+	Intell++	Intell-	Affec+	Affec-	Agress+	Agress-
Medor	0.17	0.00	0.00	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.17	0.00
Djeck	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17
Taico	0.00	0.00	0.17	0.17	0.00	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00
Rocky	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.00	0.00	0.17	0.17	0.00	0.00	0.17
Boudog	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.17	0.00	0.17	0.00	0.00	0.00	0.17	0.17	0.00
Wisky	0.17	0.00	0.00	0.00	0.17	0.00	0.00	0.00	0.17	0.00	0.00	0.17	0.17	0.00	0.17	0.00

TABLE 3.25 – Profils lignes supplémentaires

Enfin, on applique la formule pour obtenir les coordonnées des individus à partir des coordonnées des variables. Pour un individu supplémentaire i' , sa coordonnée factorielle sur l'axe α est :

$$F_\alpha(i') = \frac{1}{\sqrt{\lambda_\alpha}} \sum_{j=1}^J \sum_{k=1}^{K_j} \frac{y_{i'k}}{J} G_\alpha(k) \quad (3.37)$$

```
# Coordonnées factorielles des individus supplémentaires
rowsupcoord = rowsupprofil.dot(modcoord)/np.sqrt(eigenvalue)
print(rowsupcoord.iloc[:,[0,1]].round(3))
```

Chien	Dim.1	Dim.2
Medor	0.177	0.181
Djeck	0.071	-0.183
Taico	-0.001	-0.211
Rocky	0.475	-0.696
Boudog	0.136	0.427
Wisky	0.036	0.413

TABLE 3.26 – Coordonnées des individus supplémentaires

On peut à présent positionner ces individus dans le repère.

```
# Positionnement des individus supplémentaires
biplot(data1=rowcoord,data2=rowsupcoord,eigen=eigenvalue,
        axe1=0,axe2=1,figsize=(16,9))
```

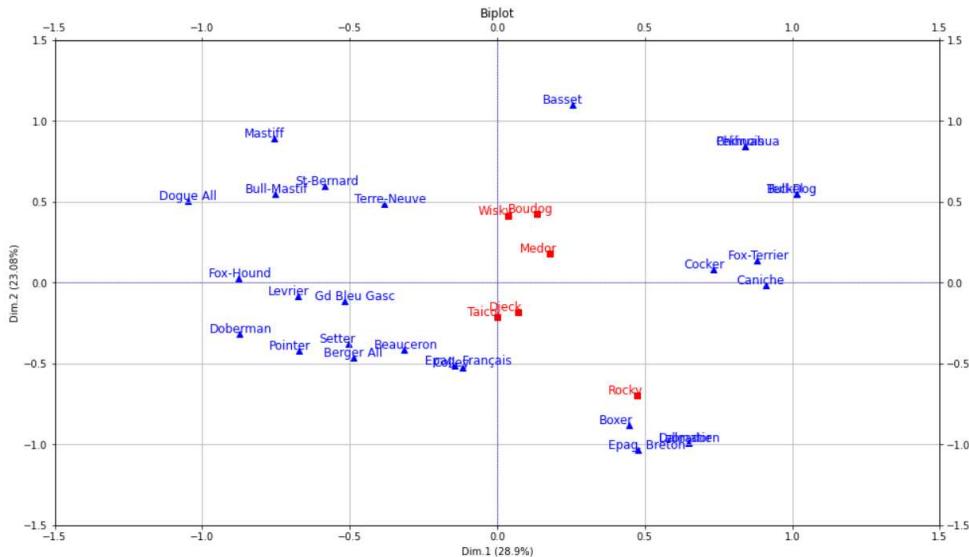


FIGURE 3.14 – Nuage des individus actifs et supplémentaires (en rouge)

2) Variables supplémentaires

Tout comme en ACP, le principe de variable illustrative est aussi valable pour l'ACM. Elles peuvent être qualitatives ou quantitatives, non exploitées pour la formation des axes, mais que l'on utilise pour mieux comprendre et commenter les résultats. Les variables supplémentaires ont pour vocation de renforcer l'interprétation des composantes.

2.1) Variables qualitatives supplémentaires

Au début de ce chapitre, nous avons mis en supplémentaire la variable « Fonction », elle sera exploitée dans cette partie afin de renforcer l'interprétation.

```
# Variables qualitatives supplémentaires
vsqual = pd.read_excel('races_canines_acm.xls', sheet_name=2, header=0, index_col=0)
```

L'objectif est de positionner les modalités de cette variable dans le plan factoriel. Deux approches permettent d'aboutir aux mêmes résultats :

a) Approche par moyenne conditionnelle

Tout d'abord, on effectue une concaténation de notre variable avec les coordonnées des individus actifs. On calcule ensuite les moyennes conditionnelles sur tous nos axes.

```
# Structures temporaires réunissant les coordonnées et la variable Fonction
df = pd.concat([rowcoord, vsqual], axis = 1)
# Calcul des moyennes conditionnelles
condmeanfonction = pd.pivot_table(df, values = index, index = "Fonction",
                                    aggfunc = "mean")
print(condmeanfonction[['Dim.1','Dim.2']].round(3))
```

```

# Structures temporaires réunissant les coordonnées et la variable Fonction
df = pd.concat([rowcoord, vsqual], axis = 1)
# Calcul des moyennes conditionnelles
condmeanfonction = pd.pivot_table(df, values = ['Dim.1','Dim.2','Dim.3'],
                                    index = "Fonction",aggfunc = "mean")
print(condmeanfonction.iloc[:,[0,1]].round(3))

```

Fonction	Dim.1	Dim.2
chasse	-0.224	-0.268
compagnie	0.721	0.059
utilite	-0.650	0.228

TABLE 3.27 – Moyennes conditionnelles

Remarque

Si $\bar{G}_\alpha(k')$ est la moyenne conditionnelle des observations qui portent la modalité supplémentaire k' sur l'axe α , il faut la corriger par la valeur propre de l'axe pour que la position soit raccordée avec celle des modalités actives. Soit :

$$G_\alpha(k') = \frac{1}{\sqrt{\lambda_\alpha}} \bar{G}_\alpha(k') \quad (3.38)$$

```

# Coordonnées factorielles des modalités supplémentaires
modsupcoord = condmeanfonction.apply(lambda x : x/np.sqrt(eigenvalue[:3]),
                                         axis = 1)
print(modsupcoord.iloc[:,[0,1]].round(3))

```

b) Approche par relation de transition

Elle est plus classique et consiste à utiliser les relations de transition. Ainsi, on produit les coordonnées factorielles de la modalité k' à partir de son profil et de coordonnées des individus ($F_\alpha(i)$). Soit :

$$G_\alpha(k') = \frac{1}{\sqrt{\lambda_\alpha}} \sum_{i=1}^I \frac{y_{ik'}}{I_{k'}} F_\alpha(i) \quad (3.39)$$

Appliquons la formule de relation de transition définie ci dessus.

```

# Codage en 0/1
dummiesfunc = pd.get_dummies(vsqual, prefix = "",prefix_sep = "")

# Calcul des profils
modsupprofil = dummiesfunc.apply(lambda x : x/np.sum(x), axis = 0)

#Coordonnées des modalités supplémentaires
modsupcoord = modsupprofil.T.dot(rowcoord)/np.sqrt(eigenvalue)
print(modsupcoord.iloc[:,[0,1]].round(3))

```

Approche 1		
Fonction	Dim.1	Dim.2
chasse	-0.322	-0.432
compagnie	1.039	0.095
utilite	-0.936	0.367

Approche 2		
Fonction	Dim.1	Dim.2
chasse	-0.322	-0.432
compagnie	1.039	0.095
utilite	-0.936	0.367

TABLE 3.28 – Coordonnées des modalités supplémentaires

On constate les coordonnées sont identiques suivant les deux approches. On l'insère dans le plan factoriel.

```
# Positionnement des modalités supplémentaires
biplot(data1=modcoord,data2=modsupcoord,eigen=eigenvalue,
       axei=0,axej=1,figsize=(16,9))
```

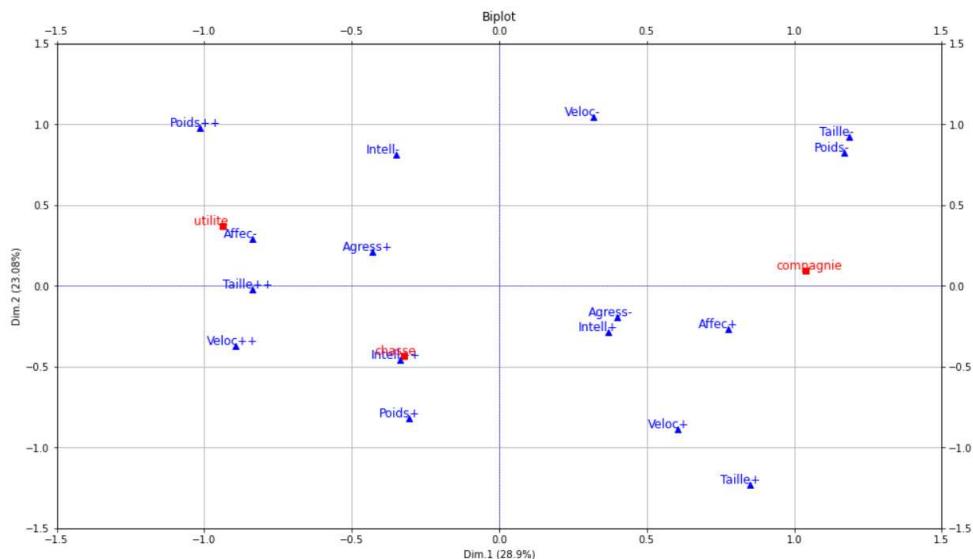


FIGURE 3.15 – Nuage des modalités actives et supplémentaires (en rouge)

On calcule la distance à l'origine de chacun des modalités supplémentaires.

```
# Marge ligne
marge = dummiesfunc.sum(axis=0)
# pk des modalités
pk = marge/I
# Distance à l'origine
modsupdisto = (1/pk) - 1
print(modsupdisto.round(3))
```

Cette distance à l'origine nous permet de calculer leur valeur test.

```
# Valeur test des modalités supplémentaires
modsupVtest = modsupcoord.apply(lambda x:x*np.sqrt(((I-1)*marge)/(I-marge)))
print(modsupVtest.iloc[:,[0,1]].round(3))
```

modalité	Effectifs	Disto ²	Coordonnées		Valeur test	
			Dim.1	Dim.2	Dim.1	Dim.2
chasse	9	2.000	-0.322	-0.432	-1.162	-1.559
compagnie	10	1.700	1.039	0.095	4.065	0.373
utilite	8	2.375	-0.936	0.367	-3.099	1.215

TABLE 3.29 – Coordonnées et valeurs tests des modalités supplémentaires

2.2) Variable quantitative supplémentaire

Au début de ce chapitre, nous avons mis en supplémentaire la variable « Cote », elle sera exploitée dans cette partie afin de renforcer l’interprétation.

```
# Variables quantitatives supplémentaires
vsquant = pd.read_excel('races_canines_acm.xls',sheet_name=3,header=0,index_col=0)
```

Pour une variable supplémentaire quantitative, le coefficient de corrélation est l’indicateur qui s’impose le plus. On le calcule entre chaque facteur et la variable supplémentaire. Nous allons calculer le coefficient de corrélation entre notre variable âge et les deux premiers axes factoriels.

```
# Corrélation avec les deux premiers axes
cotecorr = np.corrcoef(vsquant.values,rowcoord.iloc[:,[0,1]],
                      rowvar = False)[0, 1:]
corr = pd.DataFrame(cotecorr.reshape(1,2),columns=['Dim.1','Dim.2'],
                     index=['Cote'])
print(corr)
```

	Dim.1	Dim.2
Cote	-0.042141	0.601291

TABLE 3.30 – Corrélation de Cote avec les axes 1 et 2

On positionne notre variable dans un cercle de corrélation.

```
# Visualisation du nuage des variables (=Cercle de corrélation)
def corrplot(data,eigen,axe1,axe2,figsize=None):
    try:
        if axe1==axe2:
            raise ValueError('Erreur: axe1 doit être différent de axe2.')
        elif axe1>axe2:
            raise ValueError('Erreur: axe1 doit être inférieur à axe2.')
        elif axe1<0 or axe2<0:
            msg = 'Erreur: les valeurs des axes doivent être positives ou nulles.'
            raise ValueError(msg)
        else:
```

```

p = data.shape[0]
# Valeurs propres
percent = np.array([100*x/sum(eigen) for x in eigen])
dimi = round(percent[axe1],2); dimj = round(percent[axej],2)

# Graphique
fig, axes = plt.subplots(figsize = figsize)
axes.grid()
axes.axis([-1.5,1.5,-1.5,1.5])
axes.set_title("Cercle de corrélation")
axes.set_xlabel(f"Dim.{1+axe1} ({dimi}%)")
axes.set_ylabel(f"Dim.{1+axej} ({dimj}%)")
for j in range(p):
    axes.arrow(0,0,data.iloc[j,axe1],data.iloc[j,axej],
               head_width = 0.02,head_length = 0.02,color='black')
    axes.text(data.iloc[j,axe1],data.iloc[j,axej],data.index[j],
              color = "black")
# cercle
from matplotlib.patches import Ellipse
ellipse = Ellipse((0,0),width = 2, height = 2,facecolor = "none",
                   edgecolor = "blue")
axes.add_patch(ellipse)
plt.axhline(0, color='blue',linestyle="--", linewidth=0.5)
plt.axvline(0, color='blue',linestyle="--", linewidth=0.5)
plt.show()

# if false then raise the value error
except ValueError as e:
    print(e)

# Nuage des variables
corrplot(data=corr,eigen=eigenvalue,axe1=0,axej=1,figsize=(10,10))

```

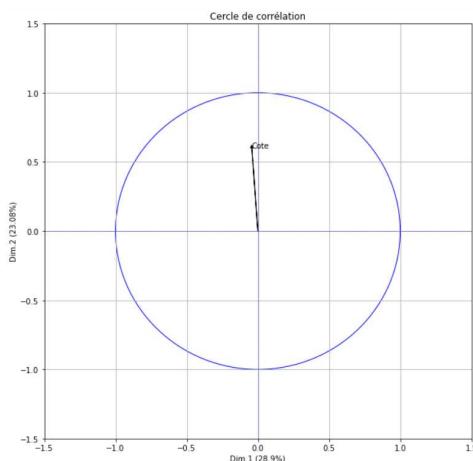


FIGURE 3.16 – Cercle de corrélation

3.5 Approche par machine learning

Cette partie entre dans ce qu'on appelle communément l'apprentissage supervisé. L'objectif ici est d'utiliser l'Analyse des Correspondances Multiples en tant que méthode de pré-traitement. Il s'agit ici de prédire grâce à l'ACM et aux méthodes de machine learning (Analyse discriminante, régression logistique) les fonctions des individus supplémentaires que nous avons créée. Pour ce faire, nous allons d'abord découper notre jeu de données composés de variables actives en deux : le train set ou échantillon d'apprentissage et le test - set ou échantillon test.

Ici, nous avons deux types de variables : les **target** ou variable à expliquer ou à prédire et les **features** ou variables explicatives.

```
# dataset
features = rowcoord.values
target = np.ravel(vsqual.values)
featuresSup = rowsupcoord.values
```

Nous importons la classe `LogisticRegression` afin de réaliser une régression logistique. Nous allons entraîner notre modèle sur l'échantillon d'apprentissage (individus actifs). Ensuite on fera la prévision sur l'échantillon test (individus supplémentaires).

```
# Classe pour la régression logistique
from sklearn.linear_model import LogisticRegression

# instanciation
model1 = LogisticRegression()
# Prédiction pour les individus supplémentaires
model1.fit(features,target)
targetSup = model1.predict(featuresSup)
prediction = pd.DataFrame(targetSup, index = ind_sup.index,
                           columns = ["Fonction Pred."])
print(prediction)
```

Chien	Fonction Pred.
Medor	chasse
Djeck	compagnie
Taico	utilite
Rocky	chasse
Boudog	compagnie
Whisky	utilite

TABLE 3.31 – Prévision sur les individus supplémentaires

Races canines	Disto ²	Poids	Inertie	Coordonnées			Contributions			Cosinus carres		
				Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3
Beauceron	1.135	0.037	0.042	-0.317	-0.418	-0.101	0.774	1.680	0.181	0.089	0.154	0.009
Basset	1.910	0.037	0.071	0.254	1.101	-0.191	0.497	11.674	0.638	0.034	0.635	0.019
Berger All	1.539	0.037	0.057	-0.486	-0.464	-0.498	1.819	2.077	4.357	0.154	0.140	0.161
Boxer	1.798	0.037	0.067	0.447	-0.882	0.692	1.539	7.485	8.408	0.111	0.433	0.266
Bull-Dog	1.644	0.037	0.061	1.013	0.550	-0.163	7.897	2.911	0.469	0.624	0.184	0.016
Bull-Mastif	2.092	0.037	0.077	-0.753	0.547	0.498	4.356	2.879	4.347	0.271	0.143	0.118
Caniche	2.161	0.037	0.080	0.912	-0.016	-0.577	6.401	0.003	5.836	0.385	0.000	0.154
Chihuahua	1.861	0.037	0.069	0.841	0.844	-0.470	5.437	6.855	3.877	0.380	0.383	0.119
Cocker	1.926	0.037	0.071	0.733	0.079	0.662	4.135	0.060	7.700	0.279	0.003	0.228
Colley	1.110	0.037	0.041	-0.117	-0.526	-0.335	0.106	2.665	1.969	0.012	0.249	0.101
Dalmatien	1.773	0.037	0.066	0.647	-0.990	0.459	3.222	9.439	3.692	0.236	0.553	0.119
Doberman	1.564	0.037	0.058	-0.873	-0.315	-0.452	5.864	0.958	3.592	0.488	0.064	0.131
Dogue All	1.955	0.037	0.072	-1.047	0.507	0.165	8.430	2.474	0.478	0.561	0.131	0.014
Epag. Breton	2.177	0.037	0.081	0.478	-1.037	0.062	1.757	10.351	0.067	0.105	0.494	0.002
Epag. Français	1.198	0.037	0.044	-0.145	-0.516	0.117	0.161	2.561	0.241	0.018	0.222	0.011
Fox-Hound	1.376	0.037	0.051	-0.877	0.025	-0.362	5.909	0.006	2.303	0.558	0.000	0.095
Fox-Terrier	1.782	0.037	0.066	0.882	0.139	0.054	5.977	0.186	0.050	0.436	0.011	0.002
Gd Bleu Gasc	1.439	0.037	0.053	-0.517	-0.113	0.044	2.058	0.124	0.034	0.186	0.009	0.001
Labrador	1.773	0.037	0.066	0.647	-0.990	0.459	3.222	9.439	3.692	0.236	0.553	0.119
Levrier	1.352	0.037	0.050	-0.677	-0.083	-0.596	3.521	0.067	6.228	0.339	0.005	0.262
Mastiff	1.905	0.037	0.071	-0.756	0.888	0.588	4.394	7.585	6.064	0.300	0.414	0.181
Pekinois	1.861	0.037	0.069	0.841	0.844	-0.470	5.437	6.855	3.877	0.380	0.383	0.119
Pointer	1.539	0.037	0.057	-0.673	-0.424	-0.686	3.487	1.730	8.256	0.295	0.117	0.306
St-Bernard	1.688	0.037	0.063	-0.583	0.594	0.894	2.617	3.393	14.040	0.202	0.209	0.474
Setter	1.135	0.037	0.042	-0.504	-0.377	-0.289	1.955	1.369	1.467	0.224	0.125	0.074
Teckel	1.644	0.037	0.061	1.013	0.550	-0.163	7.897	2.911	0.469	0.624	0.184	0.016
Terre-Neuve	1.664	0.037	0.062	-0.384	0.485	0.661	1.131	2.267	7.667	0.088	0.142	0.262

TABLE 3.32 – Analyse des individus

Modalités	Effectifs	Disto ²	Poids	Inertie	Coordonnées			Contributions			Cosinus carres			Valeurs testis		
					Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3
Taille+	5	4.400	0.031	0.136	0.851	-1.232	1.016	4.642	12.171	15.104	0.165	0.345	0.235	2.069	-2.994	2.470
Taille++	15	0.800	0.093	0.074	-0.837	-0.021	-0.051	13.459	0.010	0.115	0.875	0.001	0.003	-4.770	-0.117	-0.292
Taille-	7	2.857	0.043	0.123	1.185	0.924	-0.616	12.598	9.587	7.772	0.491	0.299	0.133	3.575	2.787	-1.858
Poids+	14	0.929	0.086	0.080	-0.305	-0.819	-0.231	1.674	15.062	2.191	0.100	0.722	0.058	-1.616	-4.333	-1.224
Poids++	5	4.400	0.031	0.136	-1.015	0.974	1.222	6.604	7.609	21.833	0.234	0.216	0.339	-2.468	2.367	2.970
Poids-	8	2.375	0.049	0.117	1.169	0.824	-0.359	14.010	8.722	3.013	0.575	0.286	0.054	3.808	2.727	-1.187
Veloc+	8	2.375	0.049	0.117	0.604	-0.888	0.356	3.737	10.117	2.972	0.153	0.332	0.053	1.997	-2.937	1.179
Veloc++	9	2.000	0.056	0.111	-0.892	-0.372	-0.763	9.180	1.996	15.335	0.398	0.069	0.291	-3.217	-1.341	-2.751
Veloc-	10	1.700	0.062	0.105	0.320	1.045	0.402	3.312	17.517	4.722	0.060	0.642	0.095	1.251	4.086	1.571
Intell+	13	1.077	0.080	0.086	0.369	-0.286	0.493	2.274	1.700	9.253	0.127	0.076	0.226	1.815	-1.403	2.423
Intell++	6	3.500	0.037	0.130	-0.335	-0.459	-0.600	0.863	2.032	6.319	0.032	0.060	0.103	0.913	-1.252	-1.635
Intell-	8	2.375	0.049	0.117	-0.349	0.809	-0.352	1.249	8.391	2.892	0.051	0.275	0.052	-1.155	2.675	-1.163
Affec+	14	0.929	0.086	0.080	0.775	-0.267	-0.061	10.791	1.601	0.151	0.648	0.077	0.004	4.104	-1.412	-0.322
Affec-	13	1.077	0.080	0.086	-0.835	0.287	0.065	11.622	1.724	0.163	0.648	0.077	0.004	-4.104	1.412	0.322
Agress+	13	1.077	0.080	0.086	-0.432	0.209	0.334	3.103	0.913	4.232	0.173	0.041	0.103	-2.120	1.028	1.639
Aggress-	14	0.929	0.086	0.080	0.401	-0.194	-0.310	2.881	0.848	3.930	0.173	0.041	0.103	2.120	-1.028	-1.639

TABLE 3.33 – Analyse des modalités

Variables	Inertie	Cosinus carres			Contributions			Rapport de corrélation		
		Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3
Taille	0.333	1.531	0.644	0.371	30.698	21.767	22.992	0.887	0.502	0.291
Poids	0.333	0.910	1.224	0.451	22.288	31.393	27.038	0.644	0.725	0.342
Velocite	0.333	0.612	1.043	0.440	14.229	29.631	23.030	0.411	0.684	0.291
Intelligence	0.333	0.210	0.411	0.381	4.387	12.124	18.465	0.127	0.280	0.234
Affection	0.167	1.295	0.153	0.008	22.413	3.324	0.314	0.648	0.077	0.004
Aggressivite	0.167	0.346	0.081	0.207	5.984	1.760	8.162	0.173	0.041	0.103

TABLE 3.34 – Analyse des variables

CHAPITRE 4

CLASSIFICATION ET KMEANS

Sommaire

4.1 La classification ascendante hiérarchique (CAH)	134
4.2 La méthode de partitionnement : KMeans	147

La classification, c'est l'action de constituer ou de construire des classes, des groupes ou des catégories. Des classes sont des ensembles d'individus (ou d'objets) qui possèdent des traits de caractères communs c'est-à-dire des individus qui se ressemblent du point de vue de l'ensemble des caractères qui les décrit. On peut citer des classes sociales (CSP, etc.) ou des classes politiques etc. On regroupe des individus qui ont des caractéristiques communes. Cela mène à deux types de classification : des classifications appelées hiérarchiques pour lesquelles on cherche à construire un arbre hiérarchique pour voir comment s'organise les objets ou les individus. Dans cette catégorie, on retrouve la classification ascendante hiérarchique (CAH). En outre, on a des classifications de type méthode de partitionnement où on essaye de constituer des groupes d'individus qui se ressemblent et de constituer une partition.

4.1 La classification ascendante hiérarchique (CAH)

L'objectif de la classification automatique est de former des groupes d'individus ou de variables afin de structurer un ensemble de données. On cherche souvent des groupes homogènes c'est-à-dire que les objets sont ressemblant à l'intérieur d'un même groupe et dissemblables entre groupes. Les méthodes de classification se distinguent entre autres par la structure de classification obtenue (partition, recouvrement, hiérarchie, pyramide).

Les méthodes de classifications ascendantes hiérarchiques sont toujours précédées d'une analyse factorielle (ACP, AFC, ACM) car elles utilisent les coordonnées factorielles des individus et non les données brutes : on dit que ce sont des méthodes de classification sur facteur. L'algorithme de ces méthodes est un algorithme itératif : on part de la partition trivial où chaque individu constitue une classe. A l'étape initiale, on agrège les individus les plus proches au sens de la distance. On remplace les individus par leurs centres de gravité avec des poids et on met à jour le tableau des distances. On reprend l'agrégation jusqu'à ce que l'on tombe sur la classe formée de tous les individus. En pratique, on représente par un arbre appelé arbre de classification ou

dendrogramme.

Les données utilisées dans le cadre de ce chapitre sont les mêmes que celles utilisées lors de l'Analyse en Composantes Principales. Nous allons mettre en œuvre une classification sur la base de données de température. En effectuant cette classification, nous allons chercher d'une part à regrouper des villes qui ont des profils météo similaires et d'autre part à caractériser les différents groupes de villes.

4.1.1 Mise en oeuvre de la classification

La classification nécessite de choisir un indice d'agrégation (on choisit l'indice d'agrégation de Ward) ainsi qu'une distance entre individus (on choisit la distance euclidienne). Au chapitre 1, nous avons réalisé une analyse en composantes principales sur les mêmes données. Nous importons les coordonnées factorielles des villes ainsi que les données brutes (actives et supplémentaires).

1. CAH à l'aide du critère de Ward

L'algorithme de classification ascendante hiérarchique est itératif. A l'étape initiale, chaque individu forme une classe : la somme des carrés totale est égale à la somme des carrés interclasses et la somme des carrés intraclasses est nulle. A l'étape courante, on part d'une partition de l'ensemble des individus en k classes G_1, \dots, G_k . Le critère de Ward consiste à regrouper les deux classes G_a et G_b les plus proches au sens de la distance de Ward. Pendant cette itération la somme des carrés intraclasses augment d'une quantité $D(G_a, G_b)$. A l'étape finale, il n'y a plus qu'une seule classe : la somme des carrés totale est maintenant égale à la somme des carrés intraclasses et la somme des carrés interclasses est nulle.

La somme des augmentations des sommes des carrés intraclasses au cours des différentes étapes est donc égale à la somme des carrés totale. Ainsi la somme des distances de Ward entre les classes fusionnées au cours des itérations est aussi égale à la somme des carrés totale. A chaque étape, on calcule la qualité de la typologie obtenue. On choisit la typologie obtenue à l'étape précédant une forte diminution de cette qualité, c'est-à-dire l'étape précédant la fusion de deux classes très éloignées au sens de la distance de Ward. Cette méthode est sensible aux points atypiques, et il est donc préférable de les exclure avant de l'utiliser.

```
# chargement des données
import pandas as pd

# coordonnées factorielles des villes
rowcoord = pd.read_pickle('temperature.pkl')
# données brutes
actif = pd.read_excel('temperature.xlsx', sheet_name=0, header=0, index_col=0)
sup = pd.read_excel('temperature.xlsx', sheet_name=1, header=0, index_col=0)
donnee = pd.concat([actif, sup], axis=1)
# Dimension
(n,p) = rowcoord.shape
```

1. Matrice des liens

Elle regroupe les villes les plus proches selon la méthode de Ward. C'est un indice de dissimilarité entre les classes G_a et G_b . Sa formule est donnée par :

$$D(G_a, G_b) = \frac{n_a n_b}{n_a + n_b} d^2(g_a, g_b) \quad (4.1)$$

Il faut au préalable indiquer le nombre d'axes factorielles à retenir pour réaliser la classification. L'observation de l'histogramme des valeurs propres issues de l'analyse factorielle pourraient aider à le faire. En effet, on peut considérer le nombre d'axes factorielles qui résument près de 95% de l'information contenue dans le jeu de données en l'occurrence 2 dans notre exemple. On peut également conserver toutes les dimensions car cela ne modifie pas la classification et permet de décomposer l'inertie totale de l'analyse en composantes principales.

La méthode d'agrégation de Ward consiste à minimiser à chaque étape le critère de distance. Elle est utilisée à chaque fois qu'on a un nuage projeté dans un espace euclidien.

```
# Matrice des liens
import numpy as np
from scipy.cluster.hierarchy import linkage

matrix = linkage(rowcoord,"ward")
linkmat = pd.DataFrame(matrix,index = np.arange(1,n,1),
                       columns = ["Ind 1", "Ind 2","Distance", "Poids"])
print(linkmat)
```

N°	Ind 1	Ind 2	Distance	Poids
1	2	14	0.540535	2
2	6	7	0.859403	2
3	3	5	0.877949	2
4	10	15	1.192691	3
5	8	11	1.307175	2
6	0	13	1.637345	2
7	17	18	2.230769	5
8	9	16	2.435392	3
9	4	12	2.820679	2
10	1	19	3.552803	3
11	20	22	4.111179	5
12	21	23	4.546507	7
13	24	26	6.846986	10
14	25	27	15.376817	15

TABLE 4.1 – Matrice de liens

Il est possible d'importer directement la méthode **ward** de la librairie **scipy**. Les résultats obtenus par les deux méthodes sont identiques.

```
# Méthode ward
from scipy.cluster.hierarchy import ward

wardmatrix = ward(rowcoord)
```

2. Dendrogramme et indice de niveau

Le dendrogramme donne une suite de partitions obtenues par coupures de certaines branches.

```
# Dendrogramme
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram

plt.figure(figsize = (14,9))
dendrogram(linkmat,labels = rowcoord.index, orientation = "top",
           color_threshold = 0)
plt.plot([0,150], [6,6], color = "silver", linestyle = "--", linewidth = 2)
plt.title("Dendrogram")
plt.xlabel('Villes')
plt.tick_params(axis = "x", rotation = -60)
plt.show()
```

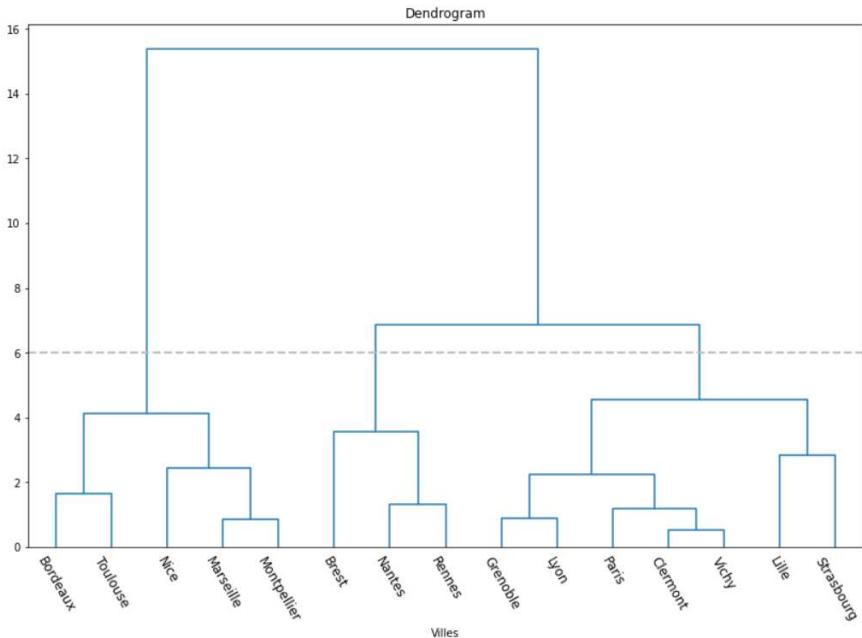


FIGURE 4.1 – Arbre hiérarchique issu de l'algorithme de Ward appliquée au tableau ??

La figure 4.1 représente l'arbre hiérarchique obtenu en appliquant l'algorithme de Ward aux données du tableau 1.2 pour calculer les distances entre les éléments. Le principe général de construction d'un arbre hiérarchique par une méthode ascendante est simple. On commence par regrouper les deux villes les plus proches. Il s'agit de Clermont et Vichy ($d^2(\text{Clermont}, \text{Vichy}) = 0.54$), ce qui est cohérent avec la position de ces dernières sur le plan factoriel (Nuage des individus en ACP Figure 1.5). On constitue ainsi le premier noeud de l'arbre. La hauteur à laquelle on relie les éléments correspond à la ressemblance entre les éléments reliés : c'est l'indice de niveau

du noeud.

A l'issue de l'agrégation de Clermont et Vichy, nous ne disposons plus que de 14 éléments à classifier : 13 villes et un groupe de 2 villes. L'algorithme regroupe ensuite Marseille et Montpellier qui ont des températures très voisines, très légèrement moins que les 2 villes précédemment agrégées ($d^2(\text{Clermont}, \text{Clermont}) = 0.54 < d^2(\text{Marseille}, \text{Montpellier}) = 0.86$). L'indice de niveau du noeud correspond à cette deuxième agrégation est plus élevé que le précédent. Et ainsi de suite, on agrège petit à petit les villes mais aussi les groupes de villes.

Si on classifie I individus, l'arbre contient $I - 1$ noeuds, qu'il est d'usage de numérotier de $I + 1$ à $2J - 1$. Les deux éléments réunis par chaque noeud sont quelquefois appelés l'un *aîné*, l'autre *benjamin*.

Le tableau 4.2 présente les indices de niveau obtenus lors de l'agrégation des classes. Elle illustre ainsi ce que l'on gagne (en inertie inter) lorsque l'on passe d'une partition donnée à la partition « immédiatement » plus fine.

Passage de	Inertie perdue
De 15 classes à 14 classes	0.01
De 14 classes à 13 classes	0.02
De 13 classes à 12 classes	0.03
De 12 classes à 11 classes	0.05
De 11 classes à 10 classes	0.06
De 10 classes à 9 classes	0.09
De 9 classes à 8 classes	0.17
De 8 classes à 7 classes	0.19
De 7 classes à 6 classes	0.26
De 6 classes à 5 classes	0.42
De 5 classes à 4 classes	0.56
De 4 classes à 3 classes	0.69
De 3 classes à 2 classes	1.56
De 2 classes à 1 classe	7.88
Total	12

TABLE 4.2 – Inerties perdues

```
# Barplot
import seaborn as sns
# Inertie perdue
inertiePerdue = np.array([7.88, 1.56, 0.69, 0.56, 0.42, 0.26, 0.19,
                         0.17, 0.19, 0.06, 0.05, 0.03, 0.02, 0.01])

fig2, axis = plt.subplots(figsize = (10,6))
sns.barplot(x = np.arange(1,n,1), y = inertiePerdue,
             color = "black")
plt.plot([0,2], [1.3,1.3], color = "black", linestyle = "--", linewidth = 2)
axis.set_xticklabels([str(i) for i in np.arange(29,15,-1)])
axis.set_title("Diagramme des indices de niveau")
axis.set_xlabel("Numéro du noeud")
```

```

axis.set_ylabel("Niveau d'agrégation")
plt.show()

```

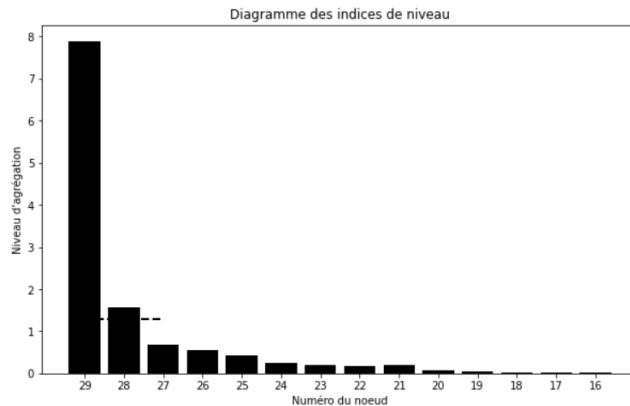


FIGURE 4.2 – Diagramme des indices de niveau

On observe une grosse perte lorsqu'on passe de 2 classes à 1 classe. En faisant la somme des pertes d'inertie inter, on retrouve l'inertie totale qui est égale à 12. La première barre (la plus grande) donne la perte d'inertie inter lorsqu'on regroupe 2 classes d'individus en une seule classe. La perte d'inertie inter est de 7,88, soit 65,68% de l'inertie totale, ce qui est très important. Cela veut dire que ce regroupement agrège des individus très différents. Il n'est pas question de regrouper ces deux classes.

La deuxième montre la perte d'inertie lorsqu'on passe de 3 classes à 2 classes. Cette perte d'inertie inter est de 1,56. Cette quantité est relativement importante également et on pourrait se demander s'il faut ou non faire ce regroupement pour passer de 3 classes à 2 classes. Les passages de 15 à 14 classes, de 14 à 13 classes, etc. du début conduisent à de très faibles pertes d'inertie inter et donc les regroupements sont naturels. La question est alors de savoir : jusqu'à quand peut-on regrouper les classes ? Et quand faut-il s'arrêter de regrouper ? En d'autres termes, combien de classes faut-il faire ?

Le trait en interrompu représente le niveau de coupure. En effet, en élevant le trait, on fait apparaître une partition en 2 classes (villes chaudes/villes froides) : L'inertie inter vaut 7,88 et l'inertie totale 12, ce qui donne un ratio (inertie inter/inertie totale) égale à 65,67%. C'est-à-dire qu'en séparant les villes en deux groupes, les 5 villes méridionales : Bordeaux, Marseille, Montpellier, Nice et Toulouse et les 10 autres villes froides : Brest, Nantes, Rennes, Clermont, Grenoble, Lille, Lyon, Paris, Strasbourg et Vichy, on récupère 65,67% d'information contenue dans le tableau de données. Ce découpage en deux classes résume assez grossièrement les ressemblances entre individus du tableau de données.

Ce pourcentage est à comparer au pourcentage de variabilité expliquée par le premier axe de l'ACP. Le premier axe de l'ACP récupère environ 79,85% de l'information du jeu de données. Avec la classification, en séparant juste les villes méridionales des 10 autres villes, on récupère 65,67% d'information. Donc un peu moins d'information qu'avec le premier axe de l'ACP. En effet, le premier axe donne une information plus fine ; il sépare Nice de Toulouse ou Bordeaux en

donnant une coordonnée plus extrême à Nice. De même, Lille est plus extrême que Nantes. Avec la classification, ces deux villes sont dans la même classe et avec le regroupement, on récupère 65,67% d'information.

En abaissant la coupure un peu légèrement, on obtient une classification en trois groupes (en séparant les villes froides en deux groupes). Les 5 villes méridionales : *Bordeaux, Marseille, Montpellier, Nice et Toulouse* ; les 3 villes les plus occidentales (à faible amplitude thermique) : *Brest, Nantes et Rennes* et 7 les autres (à forte amplitude thermique) : Clermont, Grenoble, Lille, Lyon, Paris, Strasbourg et Vichy. Le passage en 3 classes permet de ne pas perdre l'inertie perdue lors du passage de 2 classes à trois classes. On récupère l'inertie inter de 1,56, soit 13% de l'information. Et ces 13%, on peut les comparer au deuxième axe de l'ACP (18,97%), puisque ce dernier sépare les villes froides en deux groupes. Ainsi avec une classification en trois classes, on récupère 78,67% (65,67% + 13%) de la variabilité des données. Et avec l'ACP et les deux axes, on récupère 97,82% d'information.

En abaissant encore le trait, on fait apparaître successivement une partition en 4 classes, 5 classes et ainsi de suite. On augmente le rapport inertie inter/inertie intra, ce qui montre que ce dernier doit être examiné en référence au nombre de classes de la partition et au nombre total d'individus.

4.1.2 Choix du nombre de classes

Le choix du nombre de classe est très important car construire une partition avec trop peu de classe risque de conduire à des classes qui ne sont pas homogènes. Et au contraire, construire une partition avec trop de classe risque de conduire à des classes qui ne se différencient pas suffisamment. Se pose alors la question cruciale du choix du nombre de classe. Ainsi, la recherche d'une meilleure partition est primordiale. De manière intuitive, une partition d'un ensemble d'individus (villes) est bonne si :

1. à l'intérieur de chaque classe, la variabilité est faible. En d'autres termes, si la variance des individus qui composent la classe est faible pour chaque variable
2. d'une classe à l'autre la variabilité est grande. Autrement dit si, pour chaque variable, la moyenne des individus qui composent une classe varie beaucoup d'une classe à l'autre.

On peut déterminer un nombre de classe à partir de l'arbre (Figure 4.1). On aura tendance à couper l'arbre là où les branches sont assez longues. On peut aussi s'appuyer sur le diagramme des indices de niveaux (Figure 4.2) et choisir d'arrêter de regrouper des classes quand le saut est faible. Ce qui nous indique qu'on récupère assez peu d'information et qu'il n'est plus vraiment utile de regrouper certaines classes. L'allure du diagramme des indices de niveaux suggère des niveaux de coupure privilégiés, c'est-à-dire ceux qui précèdent une décroissance rapide du gain en inertie inter. Le diagramme suggère une coupure en 3 classes.

```
# Arbre avec 3 classes
nivCoup=6
plt.figure(figsize = (14,9))
dendrogram(linkmat, labels = donnee.index, distance_sort = "ward",
           orientation = "top", color_threshold = nivCoup)
plt.plot([0,150], [nivCoup,nivCoup], color = "silver", linestyle = "--",
         linewidth = 2)
plt.title("Dendrogram avec matérialisation de 3 classes")
```

```

plt.xlabel('Villes')
plt.tick_params(axis = "x", rotation = -60)
plt.show()

```

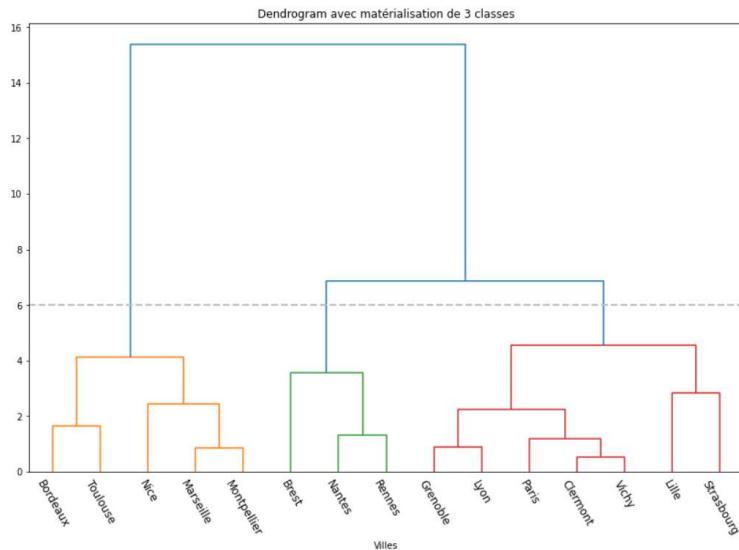


FIGURE 4.3 – Dendrogramme avec matérialisation de 3 classes

On extrait le numéro de classes de chaque observation.

villes	classe
Bordeaux	1
Brest	2
Clermont	3
Grenoble	3
Lille	3
Lyon	3
Marseille	1
Montpellier	1
Nantes	2
Nice	1
Paris	3
Rennes	2
Strasbourg	3
Toulouse	1
Vichy	3

TABLE 4.3 – Classes d'appartenance des villes

```

# Découpage en 3 classes
from scipy.cluster.hierarchy import fcluster

```

```

nbClasse = 3
cahGroupe = fcluster(linkmat, t = nivCoup, criterion = "distance")
classes = pd.DataFrame(cahGroupe, index = donnee.index, columns = ["classe"])
print(classes)

# Graphique
fig, axes = plt.subplots(figsize = (14,8)); axes.grid()
axes.axis([-8,8,-6,6])
axes.set_title("Projection des individus")
axes.set_xlabel("Dim. 1 (79.85%)")
axes.set_ylabel("Dim. 2 (18.97%)")
cdict = {'classe 1': 'green','classe 2': 'blue', 'classe 3':'red'}
for group in np.unique(classes.classe):
    idx = np.where(classes.classe==group)
    axes.scatter(rowcoord.iloc[:,0].values[idx[0]],rowcoord.iloc[:,1].values[idx[0]],
                 label = 'classe '+str(group), c = cdict['classe '+str(group)])
    for i in idx[0]:
        axes.text(rowcoord.iloc[:,0].values[i],rowcoord.iloc[:,1].values[i],
                  s=rowcoord.index[i],c = cdict['classe '+str(group)], fontsize = 11)
axes.legend()
plt.axhline(0, color='blue',linestyle="--", linewidth=0.5)
plt.axvline(0, color='blue',linestyle="--", linewidth=0.5)
plt.show()

```

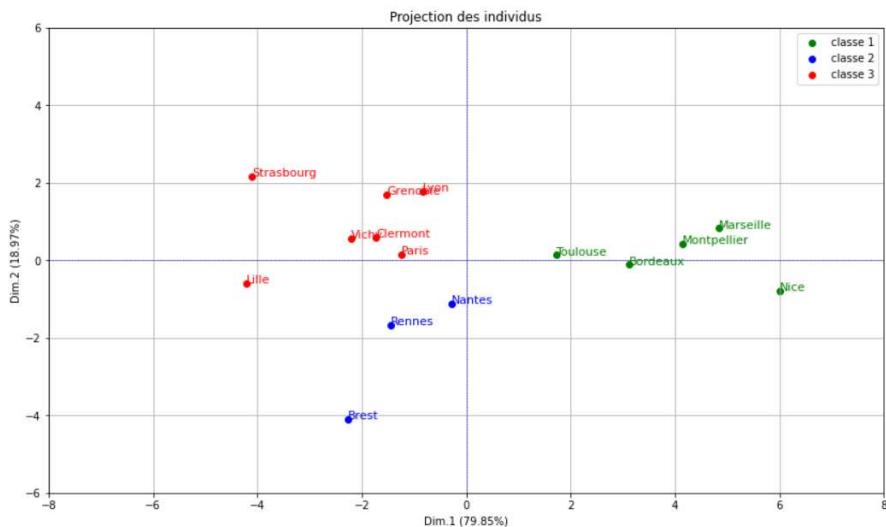


FIGURE 4.4 – Nuage des individus classés

On peut regrouper les villes par classe d'appartenance.

```

# Regroupement
dico = dict()

```

```

for i in range(1,4,1):
    for j in classes.classe:
        if i==j:
            dico[f'classe {i}'] = list(classes[classes.classe==i].index)
# Affichage
for i, keys in enumerate(dico.keys()):
    print(f'classe {i+1} : {dico[keys]} : {len(dico[keys])} éléments')

```

classes	villes	effectifs
classe 1	Bordeaux, Marseille, Montpellier, Nice, Toulouse	5
classe 2	Brest, Nantes, Rennes	3
classe 3	Clermont, Grenoble, Lille, Lyon, Paris, Strasbourg, Vichy	7

TABLE 4.4 – Classes d'appartenance des villes et effectifs par classe

Les classes ainsi formées peuvent être interprétées grâce à la comparaison moyennes par groupe et moyennes totales.

4.1.3 Interprétation des classes

1) Problématique

L'idée est de comparer les moyennes des variables conditionnellement aux groupes. L'arbre hiérarchique permet de définir chaque classe par l'énumération des individus qui la composent. Mais cela n'est pas suffisant pour connaître avec précision les caractéristiques communes des individus d'une classe.

L'idée la plus simple consiste à calculer, pour chaque variable X, la moyenne des individus de chaque classe. Pour une classe k donnée, en comparant pour chaque variable la moyenne de la classe (notée \bar{x}_k) à la moyenne générale (notée \bar{x}), on peut caractériser la classe. Mais l'indicateur $\bar{x}_k - \bar{x}$ n'est pas suffisant car il doit être relativisé par l'effectif de la classe k (notée I_k) et l'écart-type de la variable X (noté σ).

2) Valeur-test

L'idée de relativiser la quantité $\bar{x}_k - \bar{x}$ par l'écart-type général σ apparaît déjà dans le centrage et la réduction, transformation des données utilisée en préalable à l'analyse en composantes principales : on choisit l'écart-type comme unité, ce qui permet de comparer entre elles des valeurs de variables différentes.

L'idée de relativiser par l'effectif de la classe se situe sur un tout autre plan. Empiriquement, on a l'intuition que même que pour une variable qui n'a rien à voir avec la partition (cas des variables supplémentaires), la différence $\bar{x}_k - \bar{x}$ n'est jamais (en pratique) exactement nulle et risque, l'écart-type général σ étant fixé, de s'écartez d'autant plus de 0 que l'effectif de la classe est faible.

On peut montrer que la distribution des valeurs attendues de \bar{x}_k a pour moyenne \bar{x} et pour variance :

$$\sigma_{\bar{x}_k}^2 = \frac{\sigma^2}{I_k} \frac{I - I_k}{I - 1} \quad (4.2)$$

Ainsi, on « situe » \bar{x}_k dans cette distribution en calculant la **valeur-test** :

$$\frac{\bar{x}_k - \bar{x}}{\sigma_{\bar{x}_k}} = \frac{\bar{x}_k - \bar{x}}{\sigma} \times \sqrt{\frac{(I-1)I_k}{I-I_k}} \quad (4.3)$$

L'expression 4.3 montre comment l'écart $\bar{x}_k - \bar{x}$ est « relativisé » par σ et I_k . Ainsi construite, la valeur-test, à l'instar des valeurs centrées réduites, est comparable d'une variable à l'autre et d'une classe à l'autre.

```
# Concaténation des données
df = pd.concat([donnee, classes], axis = 1)
meanvar = donnee.mean(axis=0)           # moyenne générale
stdvar = donnee.std(axis=0,ddof=0)     # ecart-type général
# Moyennes conditionnelles
condmean = pd.pivot_table(df, values = donnee.columns,index=['classe'],
                           aggfunc=np.mean)[donnee.columns].T
# Effectifs par classes
nk = df.groupby(df.classe).size()
# valeur-test
u = condmean.apply(lambda x : (x-meanvar)/stdvar, axis=0)
v = u.apply(lambda x : x*np.sqrt(nk*(n-1)/(n-nk)), axis=1)
v.columns = ['classe '+str(x+1) for x in range(len(dico))]
print(v.round(2))
```

mois	classe 1	classe 2	classe 3
Jan	2.46	1.28	-3.36
Fev	2.88	0.66	-3.25
Mars	3.24	-0.25	-2.85
Avril	3.33	-1.30	-2.11
Mai	3.00	-2.02	-1.22
Juin	3.00	-2.05	-1.19
Juil	2.92	-2.18	-1.00
Août	3.18	-2.02	-1.38
Sept	3.40	-1.45	-2.05
Oct	3.32	-0.41	-2.81
Nov	2.97	0.44	-3.15
Dec	2.54	1.11	-3.28
moy	3.39	-0.75	-2.60
amp	0.50	-2.95	1.89
Lati	-2.95	1.49	1.60
Long	0.65	-2.88	1.69

TABLE 4.5 – Valeur-test

La distribution des valeurs possibles de \bar{x}_k peut être approchée par une loi Normale, ce qui permet de calculer une probabilité associée à la valeur observée. Cette probabilité est définie comme la probabilité d'obtenir, dans le cadre du modèle de tirage au hasard, une valeur de \bar{x}_k au moins aussi éloignée de \bar{x} que ne l'est la valeur de \bar{x}_k effectivement observée.

```
# Calcul des probabilités associées aux valeurs-tests
import scipy.stats as stats

prob = v.apply(lambda x : 2*(1-stats.norm(0,1).cdf(np.abs(x))))
print(prob.round(3))
```

mois	classe 1	classe 2	classe 3
Jan	0.014	0.201	0.001
Fev	0.004	0.512	0.001
Mars	0.001	0.800	0.004
Avril	0.001	0.195	0.035
Mai	0.003	0.044	0.224
Juin	0.003	0.040	0.233
Juil	0.004	0.029	0.315
Août	0.001	0.043	0.167
Sept	0.001	0.146	0.041
Oct	0.001	0.682	0.005
Nov	0.003	0.662	0.002
Dec	0.011	0.268	0.001
moy	0.001	0.456	0.009
amp	0.615	0.003	0.058
Lati	0.003	0.136	0.111
Long	0.517	0.004	0.090

TABLE 4.6 – Probabilités associées aux valeur-tests

L'intérêt de cette probabilité est de fournir un éclairage complémentaire à celui de la valeur-test. Ainsi, une valeur-test de 2 peut être appréciée en disant que l'écart de la moyenne correspondant possède environ 5 chances sur 100 d'être obtenu ou dépassé dans le cadre du modèle de tirage au hasard.

On peut à présent caractériser les différentes classes en triant les variables par valeurs-tests décroissantes.

```
# ecart-types conditionnels
columns = ['V.test','Proba','classe','général','classe','général']
condstd = pd.pivot_table(df, values = donnee.columns,index=['classe'],
                         aggfunc=lambda x:np.std(x,ddof=0))[donnee.columns].T
# Caractéristiques de la classe 1
classe1 = pd.concat([v.iloc[:,0],prob.iloc[:,0],condmean.iloc[:,0],meanvar,
                     condstd.iloc[:,0],stdvar],axis=1)
classe1.columns = columns
print(classe1.sort_values(by='V.test',ascending=False).round(2))
```

Classe 1 : Nice, Marseille, Montpellier, Bordeaux, Toulouse

variable	V.test	Proba	Moyennes		Écarts-types	
			classe	général	classe	général
Sept	3.40	0.00	19.28	16.99	0.75	1.79
moy	3.39	0.00	13.80	11.81	0.74	1.55
Avril	3.33	0.00	12.70	10.98	0.58	1.37
Oct	3.32	0.00	14.54	12.32	0.94	1.77
Mars	3.24	0.00	10.04	8.23	0.52	1.48
Août	3.18	0.00	21.90	19.57	0.79	1.94
Juin	3.00	0.00	19.80	17.83	0.73	1.73
Mai	3.00	0.00	16.08	14.43	0.69	1.45
Nov	2.97	0.00	9.88	7.93	1.00	1.74
Juil	2.92	0.00	22.10	19.83	1.00	2.06
Fev	2.88	0.00	6.80	4.83	0.94	1.81
Dec	2.54	0.01	6.66	4.85	0.90	1.89
Jan	2.46	0.01	5.78	3.97	0.92	1.94
Long	0.65	0.52	3.37	2.58	2.68	3.21
amp	0.50	0.62	16.34	15.91	0.99	2.25
Lati	-2.95	0.00	43.56	46.04	0.47	2.22

TABLE 4.7 – Caractérisation de la classe 1 par l'ensemble des variables

```
# Caractéristiques de la classe 2
classe2 = pd.concat([v.iloc[:,1],prob.iloc[:,1],condmean.iloc[:,1],meanvar,
                     condstd.iloc[:,1],stdvar],axis=1)
classe2.columns = columns
print(classe2.sort_values(by='V.test',ascending=False).round(2))
```

Classe 2 : Brest, Rennes, Nantes

variable	V.test	Proba	Moyennes		Écarts-types	
			classe	général	classe	général
Lati	1.49	0.14	47.81	46.04	0.48	2.22
Jan	1.28	0.20	5.30	3.97	0.57	1.94
Dec	1.11	0.27	5.97	4.85	0.73	1.89
Fev	0.66	0.51	5.47	4.83	0.24	1.81
Nov	0.44	0.66	8.33	7.93	0.50	1.74
Mars	-0.25	0.80	8.03	8.23	0.26	1.48
Oct	-0.41	0.68	11.93	12.32	0.25	1.77
moy	-0.75	0.46	11.20	11.81	0.38	1.55
Avril	-1.30	0.20	10.03	10.98	0.65	1.37
Sept	-1.45	0.15	15.60	16.99	0.70	1.79
Mai	-2.02	0.04	12.87	14.43	0.95	1.45
Août	-2.02	0.04	17.47	19.57	1.09	1.94
Juin	-2.05	0.04	15.93	17.83	1.16	1.73
Juil	-2.18	0.03	17.43	19.83	1.35	2.06
Long	-2.88	0.00	-2.34	2.58	1.38	3.21
amp	-2.95	0.00	12.37	15.91	1.56	2.25

TABLE 4.8 – Caractérisation de la classe 2 par l'ensemble des variables

```
# Caractéristiques de la classe 3
classe3 = pd.concat([v.iloc[:,2],prob.iloc[:,2],condmean.iloc[:,2],meanvar,
                     condstd.iloc[:,2],stdvar],axis=1)
classe3.columns = columns
print(classe3.sort_values(by='V.test',ascending=False).round(2))
```

Classe 3 : Clermont, Grenoble, Lille, Lyon, Paris, Strasbourg, Vichy

variable	V.test	Proba	Moyennes		Écarts-types	
			classe	général	classe	général
amp	1.89	0.06	17.13	15.91	1.44	2.25
Long	1.69	0.09	4.13	2.58	1.68	3.21
Lati	1.60	0.11	47.05	46.04	1.88	2.22
Juil	-1.00	0.31	19.24	19.83	1.04	2.06
Juin	-1.19	0.23	17.24	17.83	0.91	1.73
Mai	-1.22	0.22	13.93	14.43	0.74	1.45
Août	-1.38	0.17	18.80	19.57	0.88	1.94
Sept	-2.05	0.04	15.94	16.99	0.74	1.79
Avril	-2.11	0.04	10.16	10.98	0.64	1.37
moy	-2.60	0.01	10.66	11.81	0.62	1.55
Oct	-2.81	0.00	10.90	12.32	0.66	1.77
Mars	-2.85	0.00	7.03	8.23	0.81	1.48
Nov	-3.15	0.00	6.36	7.93	0.65	1.74
Fev	-3.25	0.00	3.16	4.83	0.76	1.81
Dec	-3.28	0.00	3.07	4.85	0.91	1.89
Jan	-3.36	0.00	2.11	3.97	0.88	1.94

TABLE 4.9 – Caractérisation de la classe 3 par l'ensemble des variables

4.2 La méthode de partitionnement : KMeans

Encore appelées méthodes d'agrégation autour des centres mobiles, elles ne nécessitent pas la construction d'un arbre hiérarchique.

4.2.1 Algorithme d'agrégation autour des centres mobiles

L'algorithme des kmeans nécessite de connaître le nombre de classes Q que l'on doit construire.

- Étape 1 : On choisit Q centres de classes au hasard. Pour notre jeu de données nous allons prendre Q=3, pour construire donc une partition en 3 classes.
- Étape 2 : On affecte tous les individus au centre des classes les plus proches. On répartit ainsi les individus dans les différentes classes.
- Étape 3 : On calcule les centres de gravité de chacune des classes. On obtient ainsi de nouveaux centres de gravités et on reprend la procédure à l'étape 2 et ainsi de suite.

Si à une certaine étape de la procédure, les classes ne changent plus et les centres de gravité ne bougent plus, l'algorithme a alors convergé. Cependant, l'algorithme des kmeans possèdent deux défauts :

- Le premier est qu'il faut connaître le nombre de classes a priori
- Le second est que la partition dépend de l'initialisation et du choix des centres de classes au hasard.

On peut consolider la partition obtenue par la CAH. En effet, la partition obtenue par cette dernière en coupant un arbre hiérarchique n'est pas optimale parce que l'on prend en compte toute une hiérarchie entre les individus et les groupes d'individus. On peut alors consolider la partition en utilisant la méthode des kmeans.

On part de la partition obtenue par le découpage de l'arbre hiérarchique comme l'initialisation dans l'algorithme des kmeans et on va itérer quelques étapes des kmeans jusqu'à convergence. Cela améliore nécessairement la partition puisque par construction, à chaque étape des kmeans, le critère (l'inertie intra) diminue. L'inconvénient par contre est que les liens hiérarchiques entre individus sont perdus. En effet, certains individus peuvent changer de classes par rapport à la partition obtenue depuis l'arbre et donc on perd complètement la hiérarchie.

4.2.2 Mise en oeuvre de la méthode des kmeans

Comme mentionné précédemment, la méthode des kmeans est utilisée en complément de la CAH faite précédemment. Elle permettra d'avoir une meilleure partition, vue que la première classification peut permettre de sélectionner des centres initiaux qui pourront faciliter la convergence de l'algorithme. Dans l'exemple précédent, nous avons regrouper nos individus en trois groupes. Pour chaque groupe, nous allons essayer de trouver l'individu le plus proche du centre de gravité de la classe. Ces individus seront appelés parangons et ils serviront à l'initialisation de la méthode kmeans. Pour ce faire, commençons d'abord par ajouter un variable classe au tableau des coordonnées factorielles. Elle nous sera utile dans l'extraction des parangons.

```
# Concaténation coordonnées-groupes
coordclasse=pd.concat([rowcoord, classes], axis=1)
```

Le code suivant permettra de trouver ses individus ainsi que leurs coordonnées. Pour l'exécuter, il faudra au préalable installer le package « dfply » permettant de manipuler des données (équivalent à dplyr sur R).

```
# Extraction des parangons et de leurs coordonnées
from dfply import *

nbclasse = 3
parangons=[]
for k in range(1,nbclasse+1):
    group = coordclasse >> row_slice(X.classe==k) >> select(~X.classe)

    baryClass=np.array(group.mean())

    distClass=group.values
    for i in range(distClass.shape[0]):
        for j in range(distClass.shape[1]):
            distClass[i,j]=(distClass[i,j]- baryClass[j])**2

    distClass= np.sum(distClass, axis=1)
```

```

distClass=pd.DataFrame({'Distance_centre': distClass})
distClass.index=group.index #parangons
distClass= distClass >> arrange(X.Distance_centre, ascending=True)
print('classe {} CAH :\n'.format(k),distClass,'\\n')
parangons.append(distClass.index[0])

```

classe	Villes	Distance centre
classe 1	Montpellier	0.203433
	Bordeaux	1.312107
	Marseille	1.453357
	Nice	5.058424
	Toulouse	5.098880
classe 2	Rennes	0.437930
	Nantes	2.520157
	Brest	4.207469
classe 3	Vichy	0.190009
	Clermont	0.451521
	Grenoble	1.406543
	Paris	1.798945
	Lyon	2.836851
	Strasbourg	5.136617
	Lille	6.223900

TABLE 4.10 – Distance au centre de gravité de la classe

Un individu parangon est un individu dont les coordonnées sont les plus proches du centre de gravité du groupe. Le profil de cet individu caractérise alors le groupe auquel il appartient. Nous allons afficher pour chaque classe les individus parangons.

```

# Individus proches des parangons
parangon = pd.DataFrame(np.transpose(parangons), columns = ['Parangons'],
                       index = ['Classe '+str(x+1) for x in np.arange(nbclasse)])
print(parangon)

```

classe	parangons	Distance au centre
classe 1	Montpellier	0.203433
classe 2	Rennes	0.437930
classe 3	Vichy	0.190009

On extrait les coordonnées factorielles des individus proches des parangons car elles seront nécessaires à l'initialisation de l'algorithme KMeans obtenues. Nous pouvons à présent lancer le partitionnement.

```

# k-means sur les coordonées factorielles des individus parangons
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters= nbclasse,init= rowcoord.loc[parangons,:],
                 n_init=1).fit(rowcoord)

```

Le partitionnement lancé, on récupère les classes d'appartenance de nos villes classifiées selon l'algorithme des KMeans.

```
#index triés des groupes et affichage des observations et leurs groupes
idk = np.argsort(kmeans.labels_)
kmeansclasses = pd.DataFrame(kmeans.labels_[idk]+1, index = donnee.index[idk],
                               columns = ["classe"])
print(kmeansclasses)
```

ville	classe
Bordeaux	1
Marseille	1
Montpellier	1
Nice	1
Toulouse	1
Brest	2
Nantes	2
Rennes	2
Clermont	3
Grenoble	3
Lille	3
Lyon	3
Paris	3
Strasbourg	3
Vichy	3

TABLE 4.11 – Classes d'appartenance des villes selon l'algorithme des KMeans

On calcule la distance des villes aux centre de gravité des différentes classes.

Villes	classe 1	classe 2	classe 3
Bordeaux	1.145	5.001	5.534
Brest	7.519	2.051	5.008
Clermont	5.720	2.926	0.672
Grenoble	5.746	4.021	1.186
Lille	8.224	3.389	2.495
Lyon	5.094	4.114	1.684
Marseille	1.206	6.926	7.110
Montpellier	0.451	6.126	6.434
Nantes	4.431	1.588	2.853
Nice	2.249	7.500	8.451
Paris	5.229	2.475	1.341
Rennes	5.697	0.662	2.726
Strasbourg	8.346	5.273	2.266
Toulouse	2.258	3.934	4.092
Vichy	6.195	3.019	0.436

TABLE 4.12 – Distance aux centres de gravité des classes

```
# Distances aux centres de classes
distance = pd.DataFrame(kmeans.transform(rowcoord), index = donnee.index,
                        columns = ['classe '+str(x+1) for x in np.arange(nbclasse)])
print(distance.round(3))
```

Une fois les nouvelles classes obtenues, il est légitime de se poser la question de savoir si des individus ont changée de classe d'une classification à l'autre.

```
#correspondance avec les groupes de la CAH
crosstabmat = pd.crosstab(cahGroupe, 1+kmeans.labels_)
print(crosstabmat)
```

		KMeans		
		classe 1	classe 2	classe 3
CAH	classe 1	5	0	0
	classe 2	0	3	0
classe 3	0	0	7	

TABLE 4.13 – Matrice de croisement

Dans notre exemple, on voit que les classes sont restées inchangées. Ainsi, représenter les nouvelles classes sur le plan factoriel n'apporterait pas d'information nouvelle, dû à la stabilité des classes.

4.2.3 Aide à la détection du nombre adéquat de groupes

KMeans, à la différence de la classification ascendante hiérarchique, ne fournit pas d'outils d'aide à la détection du nombre de classes. Il faut le programmer sous python ou utiliser des procédures proposées par des packages dédiées. Le schéma est souvent le même : on fait varier le nombre de groupes et on surveille l'évaluation d'un indicateur de qualité de la solution c'est-à-dire l'amplitude des individus à être plus proches de ses congénères du même groupe que des individus des autres groupes.

Dans ce qui suit, on calcule deux grandeurs : Elbow Method et La métrique silhouette pour différents nombres de groupes issus de la méthode des centres mobiles.

1. Elbow Method

La méthode Elbow permet de détecter une zone de coude dans la minimisation du coût (inertia_). Nous allons entraîner l'algorithme des KMeans sur un nombre de classes compris entre 1 et 6 et calculer le coût lié.

```
# Elbow method
inertia = []
k_range = range(1,7)
for k in k_range:
    model = KMeans(n_clusters = k).fit(rowcoord)
    inertia.append(model.inertia_)
```

On représente l'ensemble des valeurs inertia obtenues.

```
# Graphique
plt.figure(figsize=(12,8))
plt.grid()
plt.title("Elbow point")
plt.plot(k_range, inertia)
plt.xticks(k_range)
plt.xlabel("Nombre de clusters")
plt.ylabel("Coût du modèle (inertia)")
plt.show()
```

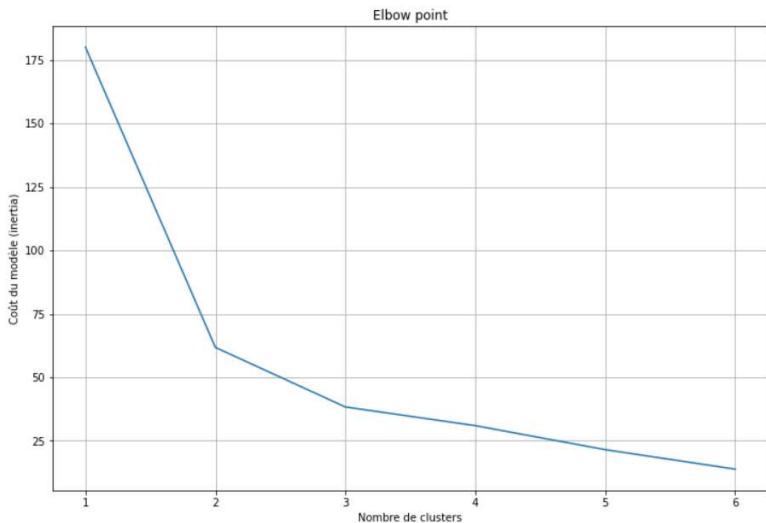


FIGURE 4.5 – Elbow point

A partir du graphique 4.5, on voit qu'une cassure s'effectue sensiblement au niveau de 2 clusters. On peut retrouver ce point grâce à la méthode KneeLocator.

```
# elbow point
from kneed import KneeLocator

kl = KneeLocator(k_range, inertia, curve = "convex", direction = "decreasing")
elbowpoint = kl.elbow
print("elbow point : %.i" % (elbowpoint))

elbow point : 2
```

Cela nous retourne la valeur 2 correspondant ainsi au nombre de clusters nécessaires.

2. Coefficient de silhouette

Pour chaque point, son coefficient de silhouette est la différence entre la distance moyenne avec les points du même groupe que lui (cohésion) et la distance moyenne avec les points des autres groupes voisins (séparation).

- Si cette différence est négative, le point est en moyenne plus proche du groupe voisin que du sien : il est donc mal classé ;
- A l'inverse, si cette différence est positive, le point est en moyenne plus proche de son groupe que du groupe voisin : il est donc bien classé

Le coefficient de silhouette proprement dit est la moyenne du coefficient de silhouette pour tous les points.

```
# Librairie pour évaluation des partition
from sklearn.metrics import silhouette_score

k_range = range(1,7)
silhouettecoef = []
for k in k_range:
    model = KMeans(n_clusters = k+1).fit(rowcoord)
    score = silhouette_score(rowcoord, model.labels_, random_state = 2022)
    silhouettecoef.append(score)

# Graphique
plt.figure(figsize=(12,8))
plt.grid()
plt.title("Silhouette")
plt.plot(k_range, silhouettecoef)
plt.xticks(k_range)
plt.xlabel("Nombre de clusters")
plt.ylabel("Silhouette Coefficient")
plt.show()
```

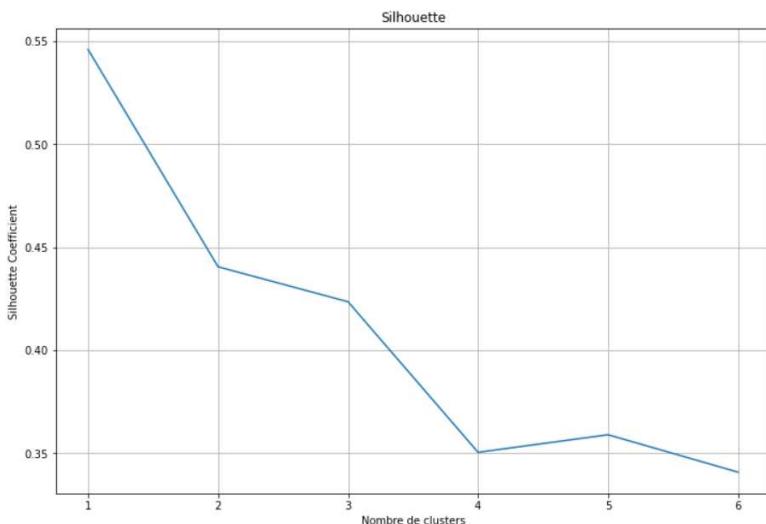


FIGURE 4.6 – Silhouette Coefficient

CHAPITRE 5

ANALYSE FACTORIELLE MULTIPLE (AFM)

Sommaire

5.1	Données - objectifs	154
5.2	Équilibre et ACP globale	157

Dans ce chapitre, nous nous intéressons à une méthode permettant d'étudier des tableaux de données plus complexes où un ensemble d'individus est décrit par des variables structurées en groupe et provenant éventuellement de différentes sources d'informations. L'intérêt de la méthode sera d'analyser globalement le tableau de données, mais aussi de comparer l'information apportée par les différentes sources d'information.

Introduite par [Pag13], l'analyse factorielle multiple fournit des résultats sur les individus et les variables comme l'analyse en composantes principales pour les variables quantitatives ou comme l'analyse factorielle des correspondances multiples pour les variables qualitatives. La spécificité et la richesse de l'analyse factorielle multiple est de prendre en compte plusieurs groupes de variables.

5.1 Données - objectifs

5.1.1 Exemple illustratif

Nous commençons cette section par décrire sur quel type de données nous travaillons et quelles sont les problématiques associées. Présentons tout d'abord le jeu à partir duquel nous allons présenter la méthode. Nous utiliserons le jeu de données `wine` dans le package `FactoMineR`. Ce jeu de données porte sur une évaluation sensorielle des vins par différents juges.

```
# Chargement des données
import pandas as pd

donnee = pd.read_csv('wine.txt',sep='\\')
```

Les données contiennent 21 lignes (vins, individus) et 31 colonnes (variables) : les deux premières colonnes sont des variables catégorielles : label (Saumur, Bourgueil ou Chinon) et soil (Reference, Env1, Env2 or Env4). Les 29 colonnes suivantes sont numériques (variables sensorielles continues). Pour chaque vin, la valeur est le score moyen pour tous les juges. L'objectif de

cette étude est d'analyser les caractéristiques des vins.

La méthode appropriée pour caractériser les vins par des variables continues est l'Analyse en Composantes Principales. Cependant, le tableau de données est structuré en différents groupes de variables :

- Un groupe qualitatif (variables label et soil).
- Un groupe concernant l'odeur avant agitation (variables Odor.Intensity.before.shaking, Aroma.quality.Fruity.before.shaking, Flower.before.shaking et Spice.before.shaking)
- Un groupe concernant l'évaluation visuelle (variables Visual.intensity, Nuance et Surface.feeling)
- Un groupe concernant l'odeur après agitation (variables Odor.Intensity, Quality.of.odour, Fruity, Flower, Spice, Plante, Phenolic, Aroma.intensity, Aroma.persistency et Aroma.quality)
- Un groupe concernant le goût (variables Attack.intensity, Acidity, Astringency, Alcohol, Balance, Smooth, Bitterness, Intensity et Harmony)
- Et un dernier groupe concernant une appréciation globale (variables Overall.quality et Typical)

	Label	Soil	Odor.intensity. Aroma.quality. before.shaking before.shaking ...	Visual. intensity	Nuance ...	Odor. Intensity	Quality. of.odour ...	Attack. intensity	Acidity ...	Overall quality	Typical
2EL	Saumur	Env1	3.074 3 ...	4.321 4 ...	3.407 3.308 ...	2.963 2.107 ...	3.393 3.25 ...	3.393 2.107 ...	3.214 3.036 ...	3.393 3.25 ...	3.393 3.25 ...
1CHA	Saumur	Env1	2.964 2.821 ...	3.222 3 ...	3.37 3 ...	3.036 2.107 ...	3.222 2.179 ...	3.222 2.179 ...	3.536 3.179 ...	3.214 3.036 ...	3.536 3.179 ...
1FON	Bourgueuil	Env1	2.857 2.929 ...	3.536 3.393 ...	3.25 2.929 ...	3.222 2.179 ...	3.222 2.179 ...	3.222 2.179 ...	3.536 3.179 ...	3.214 3.036 ...	3.536 3.179 ...
1VAU	Chinon	Env2	2.808 2.593 ...	2.893 2.786 ...	3.16 2.88 ...	2.704 3.179 ...	2.704 3.179 ...	2.704 3.179 ...	2.464 2.25 ...	2.464 2.25 ...	2.464 2.25 ...
1DAM	Saumur	Reference	3.607 3.429 ...	4.393 4.036 ...	3.536 3.36 ...	3.464 2.571 ...	3.464 2.571 ...	3.464 2.571 ...	3.741 3.444 ...	3.741 3.444 ...	3.741 3.444 ...
2BOU	Bourgueuil	Reference	2.857 3.111 ...	4.464 4.259 ...	3.179 3.385 ...	3.286 2.393 ...	3.286 2.393 ...	3.286 2.393 ...	3.643 3.393 ...	3.643 3.393 ...	3.643 3.393 ...
1BOI	Bourgueuil	Reference	3.214 3.222 ...	4.143 3.929 ...	3.429 3.5 ...	3.393 2.607 ...	3.393 2.607 ...	3.393 2.607 ...	3.714 3.357 ...	3.714 3.357 ...	3.714 3.357 ...
3EL	Saumur	Env1	3.12 2.852 ...	4.214 3.857 ...	3.654 3.077 ...	3.25 2.179 ...	3.25 2.179 ...	3.25 2.179 ...	3.393 3.071 ...	3.393 3.071 ...	3.393 3.071 ...
DOM1	Chinon	Env1	2.857 2.815 ...	4.037 3.893 ...	3.357 3.346 ...	3.286 2.286 ...	3.286 2.286 ...	3.286 2.286 ...	3.2 3.5 ...	3.2 3.5 ...	3.2 3.5 ...
1TUR	Saumur	Env2	2.893 3 ...	3.704 3.407 ...	3.222 3.259 ...	2.893 2.357 ...	2.893 2.357 ...	2.893 2.357 ...	3.179 2.964 ...	3.179 2.964 ...	3.179 2.964 ...
4EL	Saumur	Env2	3.25 3.286 ...	3.857 3.643 ...	3.607 3.385 ...	3.321 2.429 ...	3.321 2.429 ...	3.321 2.429 ...	3.571 3.5 ...	3.571 3.5 ...	3.571 3.5 ...
PER1	Saumur	Env2	3.393 3.179 ...	4.714 4.5 ...	3.481 3.385 ...	3.357 2.429 ...	3.357 2.429 ...	3.357 2.429 ...	3.148 3.556 ...	3.148 3.556 ...	3.148 3.556 ...
2DAM	Saumur	Reference	3.179 3.286 ...	4.222 4.071 ...	3.481 3.423 ...	3.393 2.286 ...	3.393 2.286 ...	3.393 2.286 ...	3.571 3.929 ...	3.571 3.929 ...	3.571 3.929 ...
1POY	Saumur	Reference	3.071 3.107 ...	4.714 4.536 ...	3.357 3.444 ...	3.519 2.111 ...	3.519 2.111 ...	3.519 2.111 ...	3.929 3.481 ...	3.929 3.481 ...	3.929 3.481 ...
1ING	Bourgueuil	Env1	3.107 3.143 ...	4.071 3.893 ...	3.357 3.37 ...	3.185 2.286 ...	3.185 2.286 ...	3.185 2.286 ...	3.643 3.296 ...	3.643 3.296 ...	3.643 3.296 ...
1BEN	Bourgueuil	Reference	2.929 3.179 ...	3.889 3.429 ...	3.286 3.308 ...	3.393 2.393 ...	3.393 2.393 ...	3.393 2.393 ...	3.75 3.571 ...	3.75 3.571 ...	3.75 3.571 ...
2BEA	Chinon	Reference	3.036 3.179 ...	3.786 3.607 ...	3.444 3.5 ...	3.071 2.571 ...	3.071 2.571 ...	3.071 2.571 ...	3.536 3.269 ...	3.536 3.269 ...	3.536 3.269 ...
1ROC	Chinon	Env2	3.071 2.926 ...	3.679 3.393 ...	3.37 3.36 ...	3.071 2.393 ...	3.071 2.393 ...	3.071 2.393 ...	3.464 3.444 ...	3.464 3.444 ...	3.464 3.444 ...
2ING	Bourgueuil	Env1	2.643 2.786 ...	2.607 2.536 ...	2.889 2.8 ...	2.179 2.25 ...	2.179 2.25 ...	2.179 2.25 ...	2.37 2.321 ...	2.37 2.321 ...	2.37 2.321 ...
T1	Saumur	Env4	3.696 3.192 ...	4.321 4 ...	3.737 3.08 ...	2.963 2.407 ...	2.963 2.407 ...	2.963 2.407 ...	2.643 2.571 ...	2.643 2.571 ...	2.643 2.571 ...
T2	Saumur	Env4	3.708 2.926 ...	4.321 4.107 ...	3.727 2.885 ...	3.333 2.571 ...	3.333 2.571 ...	3.333 2.571 ...	2.852 2.75 ...	2.852 2.75 ...	2.852 2.75 ...

FIGURE 5.1 – Source de l'image, FactoMineR <http://factominer.free.fr>

De nouveaux objectifs apparaissent comme comparer les groupes de variables (deux groupes de variables sont proches si deux vins proches l'un de l'autre du point de vue du premier groupe de variables le sont aussi du point de vue du deuxième) et mettre en évidence une typologie des groupes ou comparer de façon simultanée les typologies des vins vus par chaque groupe de variables pris un par un.

On va étudier les profils de vins selon l'évaluation sensorielle. On utilisera comme groupes actifs les groupes odor, visual, odor after shaking et taste et comme groupes illustratifs les groupes origin et overall.

5.1.2 Structure du jeu de données

La structure de notre jeu de données exemples est donc la suivante : un même ensemble d'individus est décrit par plusieurs groupes de variables. Nous considérons que nous avons I individus dans chaque groupe et que nous disposons de J groupes de variables. Dans chacun de ces groupes, nous avons K_j variables.

		Variables						
		X_1		X_j		X_J		
Individus	1	K_1		1	K_j		1	K_J
	i							
	I							

TABLE 5.1 – Exemple de tableau d'une AFM

La valeur prise par l'individu i pour la variable k est notée x_{ik} . Dans chaque groupe, les variables peuvent être quantitatives ou qualitatives. J représente le nombre de sous-tableaux ; K_j représente le nombre de variables du groupe j ; X_j est le tableau associé au groupe j et X le tableau complet $X = (X_1 | \dots | X_J)$.

Dans de nombreux domaines d'application, on trouve ce type de tableau, avec un même ensemble d'individus décrit par plusieurs groupes de variables. En effet, les groupes de variables peuvent correspondre à différentes sources d'information. En voici quelques exemple issus de divers domaines d'application :

1. En génomique, un ensemble de patients atteints de cancer du cerveau est décrit selon 3 sources d'information différentes et éventuellement complémentaires :

- Des mesures d'ADN correspondant à des mesures sur les gènes ;
- Des mesures d'expression
- Des mesures sur les protéines.

L'objectif est alors de comprendre quels sont les gènes qui entrent en jeu dans la maladie et qui sont différentiellement extrêmes selon que le patient est malade ou non et selon le type de tumeur. L'objectif est également de savoir si les 3 sources d'information sont concordantes ou si une des sources apporte des informations différentes.

2. Un autre exemple concerne une enquête menée auprès de jeunes étudiants. Plusieurs groupes de questions leur ont été posées sur leur consommation de produits tels que l'alcool ou les drogues douces, sur leurs conditions psychologiques, sur la qualité de leur sommeil et enfin des questions sur leur signalétique. Toutes les questions étaient qualitatives. L'objectif est ici de comprendre globalement le lien entre la qualité du sommeil, les conditions psychologiques et leur consommation de drogues. On ne s'intéresse pas aux liens variable par variables mais plutôt à des ressemblances plus globales.
3. En économie, cela peut correspondre à l'ensemble des indicateurs économiques, d'une période à l'autre ou d'une année à l'autre.

Les exemples de tableaux multiples sont de plus en plus nombreux car d'une part il est de plus en plus facile de recueillir et stocker des données, et d'autre part, on cherche de plus en plus souvent à analyser des problèmes plus complexes mettant en jeu plusieurs sources d'information.

5.1.3 Objectifs

Dans la majorité des exemples, les problématiques seront les suivantes : dans un premier temps, étudier et décrire l'ensemble des individus à l'aide de toutes les variables et décrire les relations entre les différentes variables d'un même groupe ou d'un groupe à l'autre. Ces objectifs coïncident avec ceux de l'analyse en composantes principales ou de l'analyse des correspondances multiples.

Ici, nous allons également profiter de la structure en groupes sur les variables pour étudier globalement les ressemblances et différences entre groupes : est-ce que les variables d'un groupe apportent la même information que les variables des autres groupes ou bien y a-t-il une information commune et une information spécifique apportée par le groupe et celle apportée par les autres groupes. Pour ce faire, on peut voir si un individu particulier est décrit de la même manière par tous les groupes ou bien si un des groupes le décrit de façon très spécifique. Enfin, on pourra comparer la typologie des individus obtenue par chaque groupe.

Un point crucial dans l'analyse multi tableaux est celui de l'équilibre de l'information. L'information d'un groupe ne doit pas écraser (ou masquer) l'information apportée par les autres groupes et il faut donc veiller à équilibrer l'information apportée par chacun des groupes.

5.2 Équilibre et ACP globale

Dans cette section, nous allons voir comment équilibrer les groupes et comment mettre en œuvre cet équilibre dans une analyse.

5.2.1 Importance d'équilibrer l'influence de chaque groupe

On rappelle que nous avons déjà chercher à équilibrer l'influence des variables dans une analyse factorielle. En effet, lorsqu'on construit une analyse en composantes principales normée, on équilibre l'influence de chaque variable pour éviter que les variables qui ont la plus grande variance aient plus d'influence dans l'analyse, i.e plus d'influence dans le calcul des distances entre individus. De la même façon, avec plusieurs groupes de variables, on veut chercher à équilibrer les groupes de variables pour éviter que certains groupes aient une contribution trop importante trop importante dans le calcul des distances entre individus.

Une première idée, qui se rapproche de celle de l'ACP, consiste à équilibrer l'influence de chaque groupe de variable en divisant la l'inertie du groupe. En effet, l'inertie correspond bien à une variance dans un cadre multidimensionnel. On pourrait donc pondérer les groupes de sorte que chaque groupe ait une inertie équivalente. Pour ce faire, il suffit de pondérer chaque variable par l'inertie totale du groupe auquel elle appartient. Avec une telle pondération, si deux groupes n'ont pas le même nombre de variables, le groupe ayant le moins de variables aura la même influence que le groupe ayant plus de variables. De ce point de vue, l'équilibre est bien respecté.

Cependant, cette pondération ne tient pas compte de la méthode utilisée ici : l'analyse factorielle. En effet, l'analyse factorielle est sensible à la répartition de l'inertie d'un groupe à l'autre.

Groupe 1	Groupe 2	Groupe 3
8 variables fortement corrélées	3 variables orthogonales	3 variables orthogonales

TABLE 5.2 – Exemple fictif de tableau en AFM

Dans le tableau 5.2, le premier groupe est composée de 8 variables fortement corrélées. Le deuxième groupe est composée de 3 variables orthogonales et enfin le troisième groupe est identique au deuxième. Avec la pondération par l'inertie totale de chaque groupe, la première dimension du groupe 1 aura un poids proche de 1 (ce groupe est unidimensionnel et toute l'inertie est quasiment sur une seule dimension), tandis que la première dimension des groupes 2 et 3 aura un poids de 1/3 (si l'inertie totale du groupe est de 1, chaque variable représente 1/3 de l'inertie). Par conséquent, la première dimension de l'analyse globale coïncide avec la première dimension du groupe 1. Or on recherche un équilibre pour trouver des structures communes d'un groupe à l'autre, donc on aimerait mettre en évidence ici la structure commune aux groupes 2 et 3. Cette idée est donc rejetée car jugée mauvaise.

Une deuxième idée consiste à équilibrer les groupes, non plus en fonction de l'inertie totale, mais en fonction de l'inertie maximum d'une dimension. Ainsi, on évite qu'un groupe unidimensionnel ait trop d'importance dans l'analyse, mais ne restreint pas la richesse d'un groupe multidimensionnel, i.e d'un groupe qui a plusieurs dimensions de variabilité. Pour construire un tel équilibre, on divise les valeurs d'une variable par la plus grande valeur propre du groupe auquel elle appartient (on divise par la racine carré de la plus grande valeur propre). Cette pondération permet dans l'exemple du tableau 5.2, de mettre en évidence la structure commune engendrée par les groupes 2 et 3.

Cet exemple fictif met en évidence une situation relativement fréquente en pratique : des groupes unidimensionnels et d'autres multidimensionnels. L'équilibre proposé évite de donner trop d'importance aux groupes unidimensionnels, groupes qui contenant une information moins riche.

Équilibre des groupes

5.2.2 L'AFM : une ACP pondérée

Nous avons trouvé une pondération qui nous convient, il ne reste plus qu'à rechercher les principales dimensions factorielles comme nous l'avons vu pour l'ACP, l'ACM ou encore l'afc. La difficulté réside bien dans le choix de la pondération. La pondération que nous avons trouvé revient à utiliser pour chaque variable, un poids égal à l'inverse de la première valeur propre du groupe auquel il appartient. Ainsi, l'AFM est une ACP pondérée. Donc la première étape de l'analyse consiste à calculer la première valeur propre de chaque groupe. Ensuite, pondérer une variable par l'inverse de l'inertie de la première valeur propre du groupe auquel elle appartient. Cela revient à diviser chaque variable par la racine carré de la première valeur propre de son groupe :

$$\left[\frac{X_1}{\sqrt{\lambda_1^1}}, \frac{X_2}{\sqrt{\lambda_1^2}}, \dots, \frac{X_j}{\sqrt{\lambda_1^j}} \right] \quad (5.1)$$

Dans la formule 5.1, X_j correspond au tableau j centré ou centré et réduit selon que les variables du groupe j sont normées ou non. Enfin, il suffit de faire l'ACP de ce tableau qui juxtapose les variables pondérées de tous les tableaux.

ACP pondérée

Du fait de l'obtention des résultats différents de ceux fournis par le package FactoMineR, nous n'avons pas pu terminé ce chapitre et laissons le soins au lecteur de se référer à des livres liés à l'analyse des données.

CHAPITRE 6

ANALYSE FACTORIELLE DES DONNÉES MIXTES (AFDM)

Sommaire

6.1	Données	161
6.2	Représentation des nuages N_I et N_K	166
6.3	Mise en œuvre de l'AFDM	168
6.4	Aide à l'interprétation	176
6.5	Éléments supplémentaires	179

Dans les chapitres précédents, lorsqu'on disposait à la fois des variables qualitatives et quantitatives dans notre jeu de données, une première approche consistait à utiliser soit l'analyse en composantes principales sur toutes les variables quantitatives en mettant en supplémentaires les variables qualitatives ; soit l'analyse des correspondances multiples sur toutes les variables qualitatives en mettant en supplémentaires les variables quantitatives. Cependant cette solution n'est pas adaptée parce qu'une seule partie de l'information seulement participe à la construction des facteurs. Par conséquent, les résultats sont faussés.

Une seconde approche consiste à découper en classes les variables quantitatives et de passer à une analyse des correspondances multiples. Néanmoins, cette solution est peu avantage lorsque le nombre d'observations est faible. En effet, l'analyse des correspondances multiples donne des résultats peu stables dans ce cas. Ou encore, lorsque le nombre de variables catégorielles est faible par rapport aux variables quantitatives. Cette solution reste anodin car il faut définir le nombre d'intervalles pour chaque variable à traiter, choisir un algorithme pour obtenir les bornes de discréétisation, etc... Le processus est donc susceptible de dénaturer l'information traitée.

Une troisième approche consiste à remplacer les variables catégorielles par une série d'indicatrices via un codage disjonctif complet. L'information n'est pas dénaturée car il y a une bijection exacte entre les variables initiales et les variables recodées. Attention, il n'est pas très judicieux de traiter directement avec une analyse en composantes principales des données mélangeant des colonnes de valeurs numériques (continues) et des indicatrices 0/1. Les dispersions étant différentes, il y a déséquilibre dans l'influence des variables, les résultats peuvent être biaisés.

C'est dans ce cadre que s'inscrit la méthode d'Analyse Factorielle de Données Mixtes en abrégée AFDM. Elle traite les tableaux individus \times variables, lesquels sont composés d'un mixte de quantitatives et de qualitatives. On y retrouve les résultats de l'analyse en composantes principales normée lorsque toutes les variables sont continues et ceux de l'analyse des correspondances multiples lorsqu'elles sont qualitatives.

6.1 Données

L'Analyse en Factorielles des Données Mixtes s'intéresse à des tableaux de données rectangulaires de mesures X avec en lignes des individus et en colonnes des variables qui sont de nature quantitatives et qualitatives :

		Variables		
		quantitatives	qualitatives	
Individus	1	k	K_1	1
	i	x_{ik}		x_{iq}
	I			

TABLE 6.1 – Tableau AFDM

Les I individus sont décrits par K_1 variables quantitatives (que nous supposons centrées réduites) et Q variables qualitatives. La q^e variable qualitative présente K_q modalités et on note $K_2 = \sum_{q=1}^Q K_q$ l'ensemble des modalités des Q variables qualitatives. Soit $K = K_1 + K_2$ le nombre total de variables quantitatives et de variables indicatrices. x_{ik} et x_{iq} représentent respectivement la valeur centrée réduite de la variable quantitative k et la modalité pour la variable qualitative q prise par l'individu i .

6.1.1 Exemple illustratif

Du tableau 6.1, on passe au tableau 6.2 suivant :

		Variables		
		quantitatives	qualitatives	
Individus	1	k	K_1	1
	i	x_{ik}		y_{ik_q}
	I			

TABLE 6.2 – Tableau AFDM récodé

Nos effectuons un codage disjonctif complet sur nos Q variables qualitatives et on note y_{ik_q} qui prend la valeur 1 si l'individu i possède la modalité k_q de la modalité q et 0 sinon. On note p_{k_q} la proportion des individus possédant la modalité k_q .

Pour illustrer ce chapitre, nous utiliserons la base de données « Auto 2005 » accessible sur la page de cours de Pierre - Louis Gonzales. Cette base de données comporte 38 modèles de véhicules décrits par 9 variables quantitatives (puissance, cylindrée, vitesses, longueur, largeur, hauteur, poids, COS2.prix) et 3 variables qualitatives (origine avec 3 modalités : France, Europe, Autres ; carburant avec 2 modalités : diesel, essence ; type4X4 avec 2 modalités : oui, non). Nous importons notre jeu de données depuis notre espace de travail.

```
# Chargement des données
import pandas as pd

donnee = pd.read_table("autos.txt", index_col = 0)
```

Les questions que l'on se pose sont les suivantes : Quelles sont les automobiles qui de ressemblent (proximité entre les individus) ? Sur quelles caractéristiques sont fondées les ressemblances / dissemblances ? Quelles sont les relations entre les variables ? Et les relations entre les modalités, entre les modalités et les variables quantitatives ?

On sauvegarde dans deux variables I et J le nombre d'individus (nombre de lignes ou nombre de véhicules) et les nombres de variables (quantitatives et qualitatives).

```
# Dimension
I = donnee.shape[0]
J = donnee.shape[1]
print("Nombre d'observations : %.i" %(I))
print("Nombre de variable : %.i" %(J))

Nombre d'observations : 38
Nombre de variable : 12
```

On sépare notre jeu de données en deux groupes : variables quantitatives et variables qualitatives.

```
# Qual variable
qual= donnee[["origine", "carburant", "type4X4"]]
# variables quantitatives
quant = donnee.drop(["origine", "carburant", "type4X4"], axis = 1)
```

6.1.2 Statistiques sur les variables quantitatives

1) Indicateurs de tendances centrales et de dispersion

Nous calculons quelques statistiques sur nos variables quantitatives.

```
# variables quantitatives
quant = donnee.drop(["origine", "carburant", "type4X4"], axis = 1)
# statistiques descriptives
stats1 = quant.describe(include = "all")
print(stats1.round(2))
```

	puissance	cylindree	vitesse	longueur	largeur	hauteur	poids	CO2	prix
count	38.00	38.00	38.00	38.00	38.00	38.00	38.00	38.00	38.00
mean	154.45	2191.61	200.89	436.39	177.55	151.82	1427.61	193.11	29493.16
std	69.13	950.08	30.76	45.57	9.45	11.49	347.75	54.24	15788.91
min	54.00	998.00	150.00	344.00	159.00	134.00	840.00	113.00	8070.00
25%	100.50	1595.75	173.75	393.75	170.25	144.00	1208.00	143.75	18210.00
50%	147.50	1997.00	200.50	444.50	177.00	147.00	1405.00	189.50	26975.00
75%	204.00	2479.25	224.00	473.25	183.75	158.75	1595.00	230.50	37187.50
max	340.00	5654.00	250.00	506.00	203.00	185.00	2495.00	295.00	78340.00

TABLE 6.3 – Statistiques descriptives sur les variables quantitatives

2) Matrice des corrélations

On calcule la matrice des corrélations de nos variables afin de mesurer leur liaison linéaire.

```
# matrice des corrélations
corr = quant.corr(method = "pearson")
print(corr.round(2))
```

	puissance	cylindree	vitesse	longueur	largeur	hauteur	poids	CO2	prix
puissance	1.00								
cylindree	0.84	1.00							
vitesse	0.85	0.58	1.00						
longueur	0.79	0.74	0.75	1.00					
largeur	0.70	0.72	0.60	0.87	1.00				
hauteur	0.03	0.26	-0.32	0.09	0.23	1.00			
poids	0.74	0.79	0.48	0.82	0.82	0.52	1.00		
CO2	0.90	0.75	0.65	0.73	0.68	0.18	0.75	1.00	
prix	0.85	0.86	0.65	0.82	0.85	0.24	0.86	0.81	1.00

TABLE 6.4 – Matrice des corrélations

```
# Graphique
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize = (12,8))
sns.heatmap(corr,xticklabels=corr.columns,yticklabels = corr.columns,
            vmin = -1, vmax = +1, center = 0,
            cmap = 'Blues',linewidths=0.5, annot=True)
plt.title('Heatmap des corrélations entre variables',fontsize=12)
plt.xlabel('Variables',fontsize=12)
plt.ylabel('Variables',fontsize=12)
plt.show()
```

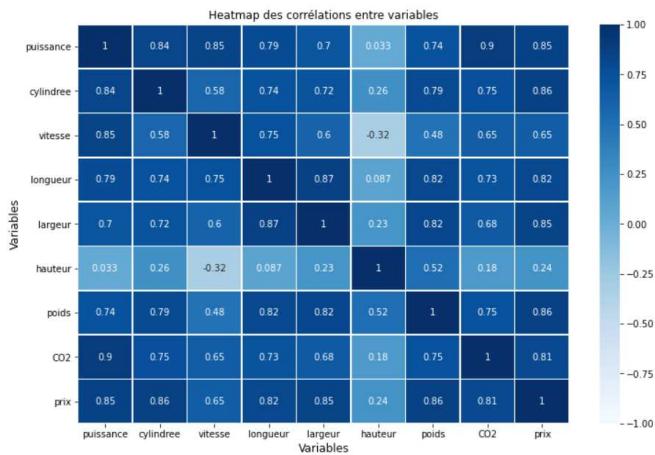


FIGURE 6.1 – Heatmap des corrélations

6.1.3 Statistiques sur les variables qualitatives

On refait les mêmes calculs avec nos variables qualitatives.

1) Indicateurs de tendance centrale

```
# variables qualitatives
qual= donnee[["origine", "carburant", "type4X4"]]
stats2 = qual.describe(include = "all")
print(stats2)
```

	origine	carburant	type4X4
count	38	38	38
unique	3	2	2
top	Europe	Essence	non
freq	15	21	33

TABLE 6.5 – Statistiques sur les variables qualitatives

2) Structure des variables qualitatives

Il s'agit ici d'avoir une distribution de chaque variable suivant ses modalités en termes d'effectifs et de proportion.

```
# Variable Origine
count1 = qual.origine.value_counts()
prop1 = qual.origine.value_counts(normalize = True).round(2)
origin = pd.concat([count1, prop1], axis=1)
origin.columns = ["effectif", "pourcentage"]
# Variable Carburant
count2 = qual.carburant.value_counts()
```

```

prop2 = qual.carburant.value_counts(normalize = True).round(2)
carbur = pd.concat([count2, prop2], axis=1)
carbur.columns = ["effectif", "pourcentage"]
# Variable Type
count3 = qual["type4X4"].value_counts()
prop3 = qual["type4X4"].value_counts(normalize = True).round(2)
typevoit = pd.concat([count3, prop3], axis=1)
typevoit.columns = ["effectif", "pourcentage"]
# concaténation
varqual = pd.concat([origin, carbur, typevoit],axis=0)
print(varqual)

```

Variables	Modalités	effectif	pourcentage
Origine	Europe	15	0.39
	France	13	0.34
	Autres	10	0.26
Carburant	Essence	21	0.55
	Diesel	17	0.45
type4×4	non	33	0.87
	oui	5	0.13

TABLE 6.6 – Répartition des modalités

```

# Diagrammes en barres
for i,name in enumerate(qual.columns):
    plt.subplot(1, 3, i+1)
    qual[name].value_counts().plot.bar(figsize=(12,5))
    plt.title(name)
    plt.xticks(rotation= 0)
    plt.tight_layout()
    plt.suptitle("Diagramme en barres", fontsize=12)
plt.show()

```

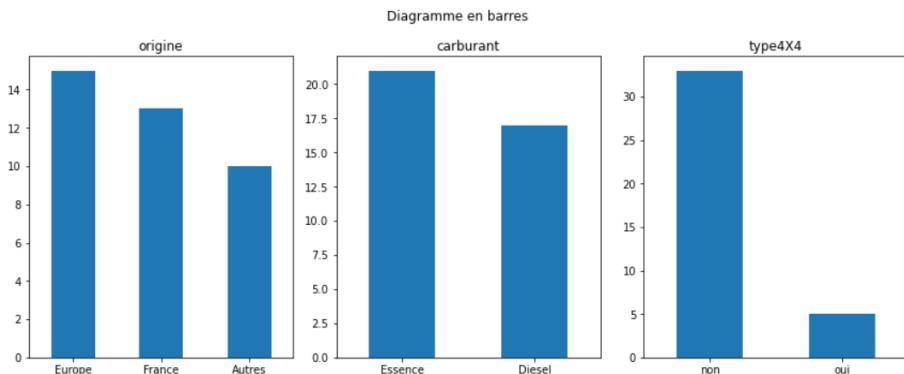


FIGURE 6.2 – Diagrammes en barres

Nous devons au préalable centrer et réduire nos variables et transformer nos variables qualitative en utilisant un codage disjonctif complet.

```
# Centrage et réduction des données
def StandardScaler(x):
    # x est vecteur
    return (x - x.mean())/x.std(ddof=0)

# Application
Z1 = quant.transform(StandardScaler)
# Codage en 0/1 (TDC)
dummies = pd.get_dummies(qual, prefix = "",prefix_sep = "")


```

On calcule les proportions associées à chaque modalité des variables qualitatives.

```
# proportions des modalités
modweight = dummies.mean(axis = 0)
print(modweight.round(3))


```

modalités	Autres	Europe	France	Diesel	Essence	non	oui
proportion	0.263	0.395	0.342	0.447	0.553	0.868	0.132

TABLE 6.7 – Proportions des modalités

6.2 Représentation des nuages N_I et N_K

6.2.1 Représentation des individus dans N_K

A partir du tableau 6.2, on a vu qu'un individu i est caractérisé par $K = K_1 + K_2$ variables quantitatives et variables indicatrices, par conséquent il appartient à un espace de dimension K . L'espace \mathbb{R}^K est menu de la métrique euclidienne diagonale des poids des colonnes (1 pour les variables quantitatives et p_{k_q} pour les modalités). Ainsi, la distance (au carré) entre les individus i et j s'écrit :

$$d^2(i,j) = \sum_{k \in K_1} (x_{ik} - x_{jk})^2 + \sum_{q \in Q} \sum_{k \in K_q} \frac{1}{p_{k_q}} (y_{ik_q} - y_{jk_q})^2 \quad (6.1)$$

Dans cette équation, les variables quantitatives contribuent exactement comme dans l'analyse en composantes principales et les variables qualitatives le font exactement comme elles le font dans l'analyse des correspondances multiples.

```
# Distance entre les individus
import numpy as np
def distance1(x,y):
    return np.sum((x-y)**2)

def distance2(x,y,z):
    return np.sum(1/z*(x-y)**2)

# Calcul de la distance
```

```

rowdist = pd.DataFrame(np.zeros(shape=(I,I),dtype=float),index = donnee.index,
                      columns = donnee.index)
for i in range(I):
    for j in range(i+1,I):
        rowdist.iloc[i,j] = distance1(Z1.iloc[i,:],
                                      Z1.iloc[j,:])+distance2(dummies.iloc[i,:],
                                      dummies.iloc[j,:],
                                      modweight)

# heatmap
plt.figure(figsize = (12,8))
sns.heatmap(rowdist,xticklabels=rowdist.columns,yticklabels = rowdist.columns,
            cmap = 'Blues',linewidths=0.5)
plt.title('Distance entre les voitures',fontsize=12)
plt.show()

```

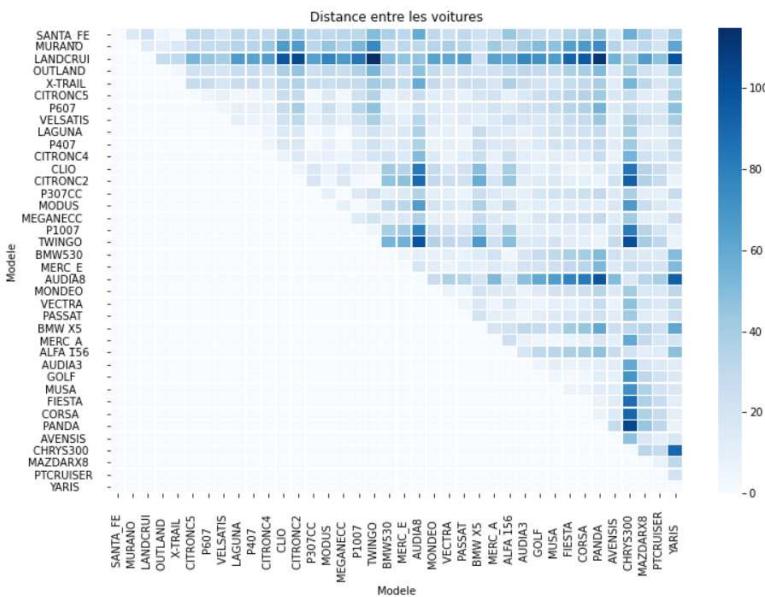


FIGURE 6.3 – Distance entre les voitures

Un cas particulier important est celui de la distance entre un individu et le centre de gravité du nuage. Ce centre de gravité est confondu avec l'origine O dès lors que les variables sont centrées. Pour les indicatrices, compte tenu de la division par $\sqrt{p_{kq}}$, la moyenne de la colonne k_q vaut $\sqrt{p_{kq}}$. On obtient finalement :

$$d^2(i, O) = \sum_{k \in K_1} x_{ik}^2 + \sum_{q \in Q} \sum_{k_q \in K_q} \frac{1}{p_{kq}} (y_{ikq} - p_{kq})^2 = \sum_{k \in K_1} x_{ik}^2 + \sum_{q \in Q} \frac{1 - p_{q(i)}}{p_{q(i)}} \quad (6.2)$$

où $q(i)$ est la modalité de la variable q possédée par i et $p_{q(i)}$ la proportion associée à $q(i)$.

```
# Distance entre un individu i et l'origine du nuage
row1 = Z1.apply(lambda x:np.sum(x**2),axis=1)
```

```

row2 = dummies.apply(lambda x:np.sum(1/modweight*(x-modweight)**2),axis = 1)
rowdisto=row1+row2
print(rowdisto.round(3))

```

6.2.2 Représentation des variables dans N_I

Chaque variable quantitative/indicatrice appartient à un espace à I dimensions. Cet espace est muni de la métrique diagonale des poids des individus notée D définie par :

$$D(k,l) = \begin{cases} p_k & \text{si } k = l \\ 0 & \text{si } k \neq l \end{cases}$$

Généralement, en analyse factorielle (ACP, ACM), on suppose a priori que les individus ont le même poids ($1/I$), par conséquent $D = \frac{1}{I}\mathbb{I}_d$ où \mathbb{I}_d est la matrice d'identité de dimension I . Comme en analyse des composantes principales, les variables quantitatives sont représentées par des vecteurs de longueur 1.

Comme en analyse des correspondances multiples, la variable qualitative q est représentée par le Nuage N_q de les $K_q - 1$ modalités, ensemble des fonctions sur I centrées et constantes sur les classes de la partition définie par q . Pour que N_q ait, dans une ACP non normée), les mêmes propriétés inertielles que dans une ACM, il faut affecter à l'indicatrice k_q le poids $1/p_{k_q}$. Comme les programmes d'ACP usuels ne permettent pas l'introduction directe de poids de colonnes, on préférera diviser les valeurs y_{ik_q} par $\sqrt{p_{k_q}}$, ce qu'on appelle le codage-ACP de la variable qualitative.

En procédant ainsi, nous obtenons en particulier la propriété fondamentale suivante de l'ACM : l'inertie projetée de N_q sur une variable centrée x est égale au rapport de corrélation $\eta^2(q,x)$ ¹ entre q et x .

```

# Standardisation des modalités
Z2 = dummies.apply(lambda x: x/np.sqrt(modweight), axis = 1)
# Concatenation
base = pd.concat([Z1,Z2], axis = 1)

```

6.3 Mise en œuvre de l'AFDM

6.3.1 L'inertie maximale

L'objectif de toute analyse factorielle consiste en la recherche d'un facteur F_1 qui soit le plus lié possible avec les variables originelles. Si la liaison n'est pas parfaite, on cherche un second facteur qui explique l'information résiduelle (non prise en compte par le facteur 1), et ainsi de suite.

Nous avons vu en ACP que la valeur propre de la première dimension correspond à :

$$1. \eta^2(q,x) = \sum_{k_q \in q} \left\langle k_q, \frac{x}{||x||} \right\rangle^2 = \sum_{k_q \in q} \left(\sum_i \left[\frac{1}{I} \frac{y_{ik_q} - p_{k_q}}{\sqrt{p_{k_q}}} \times \frac{x_i}{s_x} \right] \right)^2 = \frac{\sum_{k_q \in q} p_{k_q} \bar{x}_{k_q}^2}{s_x^2}$$

$$\lambda_1 = \sum_j \rho^2(F_1, X_j) \quad (6.3)$$

où ρ représente le coefficient de corrélation (c'est pour cela que dans le cercle des corrélations, on se concentre sur les variables proches du bord du cercle).

Nous avons également vu, dans le cadre de l'ACM, que la valeur propre de la première dimension correspond à :

$$\lambda_1 = \sum_j \eta^2(F_1, X_j) \quad (6.4)$$

où η^2 représente le carré du rapport de corrélation. On veut que les modalités d'une variable soient le plus écartés possible les uns des autres.

En AFDM, en recherchant la direction ν de \mathbb{R}^I qui rend maximum l'inertie projetée du nuage N_K (comportant à la fois les variables quantitatives et les indicatrices), on rend maximum le critère :

$$\sum_{k \in K_1} \rho^2(k, \nu) + \sum_{q \in Q} \eta^2(q, \nu) \quad (6.5)$$

L'influence d'une variable doit être raisonnée en fonction de la dimension du sous-espace qu'elle engendre. Ainsi, dans l'espace \mathbb{R}^I :

- une variable quantitative est représentée par un vecteur associé à une inertie de 1 ;
- une variable qualitative à K_q modalités est représentée par K_q vecteurs engendrant un sous-espace E_q de dimension $K_q - 1$, l'ensemble étant associé à une inertie de $K_q - 1$.

6.3.2 AFDM sous forme d'ACP sur données transformées

Dans le tableau 6.2, nous avons $K = K_1 + K_2$ colonnes. Mais nous avons introduit artificiellement de la colinéarité. En effet, la somme des indicatrices d'une variable qualitative vaut systématiquement 1. De ce fait, le nombre de valeurs propres non nulles est en réalité égale à $K_1 + \sum_{q \in Q} (K_q - 1) = K_1 + K_2 - Q = K - Q$. Disposant de cette informations, nous fixons le nombre d'axe maximal à $13 = 9 + [(3 - 1) + (2 - 1) + (2 - 1)]$. Si nous ne le faisons pas, l'ACP ne saura pas que nous lui avons passé une variante très particulière d'un tableau comprenant de nombreuses redondances et produira 3 facteurs en trop avec une variance nulle.

```
# Nombre d'axe maximal
fmax = base.shape[1] - qual.shape[1]
```

Nous demandons également à l'algorithme de ne pas centrer et réduire les variables ;

```
# Mise en oeuvre de l'AFDM
from fanalysis.pca import PCA

model = PCA(n_components = fmax, std_unit=False, row_labels=base.index,
            col_labels=base.columns)
model.fit(base.values)
```

Nous affichons les valeurs propres.

```
# Valeurs propres
columns = ['valeur propre','pourcentage d'inertie','pourcentage d'inertie cumulée']
index = ['Dim.{}'.format(x+1) for x in range(fmax)]
Eigen = pd.DataFrame(np.transpose(model.eig_),
                      index=index,columns = columns)
Eigen.index.name = 'Dimension'
# Affichage
print(Eigen.round(3))
```

Dimension	valeur propre	pourcentage d'inertie	pourcentage d'inertie cumulée
Dim.1	6.602	50.787	50.787
Dim.2	2.510	19.308	70.095
Dim.3	1.305	10.038	80.132
Dim.4	0.867	6.667	86.800
Dim.5	0.558	4.289	91.089
Dim.6	0.390	2.997	94.085
Dim.7	0.268	2.059	96.145
Dim.8	0.172	1.324	97.468
Dim.9	0.140	1.077	98.545
Dim.10	0.097	0.744	99.289
Dim.11	0.051	0.391	99.680
Dim.12	0.031	0.239	99.920
Dim.13	0.010	0.080	100.000

TABLE 6.8 – Valeurs propres AFDM

On constate effectivement que les 13 premiers axes expliquent 100% de l'inertie totale.

```
# Histogramme des valeurs propres
model.plot_eigenvalues(figsize = (8,6))
```

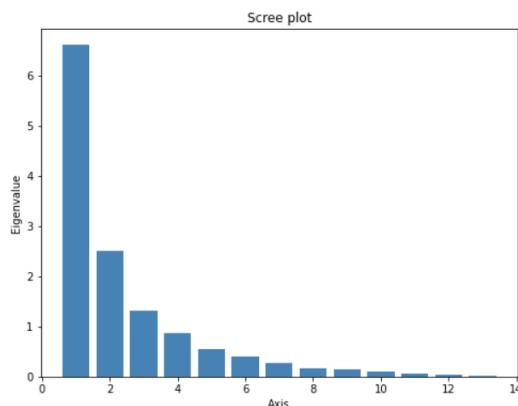


FIGURE 6.4 – Histogramme des valeurs propres

Nous constatons que la première dimension contribue à elle seule à plus de 50% de l'inertie globale du nuage. Les axes 1 et 2 contribuent à 70% de l'inertie totale.

6.3.3 Coordonnées des individus

Du point de vue des individus, les valeurs retournées par la classe PCA sont justes. On récupère les coordonnées factorielles des individus calculés par l'algorithme. L'attribut « `.row_coord_` » nous fournit directement.

```
# Coordonnées des individus
rowcoord = pd.DataFrame(model.row_coord_,index = base.index,columns = index)
print(rowcoord.iloc[:, :2].round(3))

# Nuage des individus sur les axes 1 et 2
famd_row_plot(data=rowcoord,eigen=Eigen.iloc[:, 0],axeij=0,axeij=1,figsize=(12,8))
```

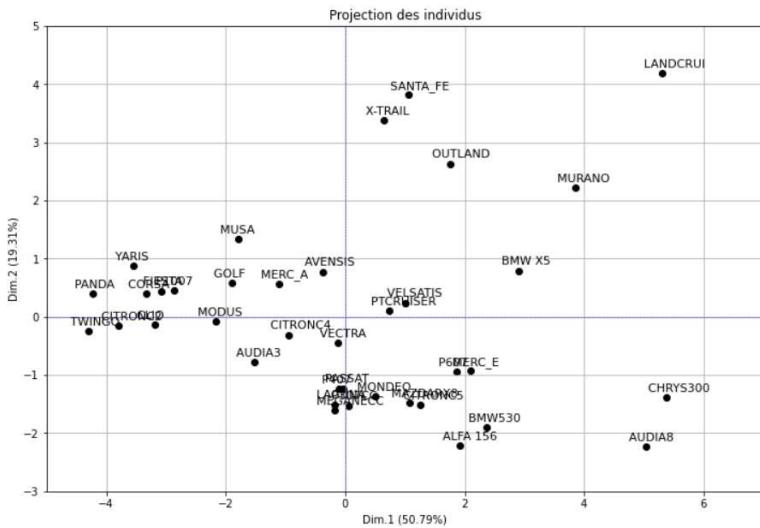


FIGURE 6.5 – Nuage des individus

On rappelle que : $\forall \alpha, \quad \lambda_\alpha = \frac{1}{I} \sum_{i=1}^I F_\alpha^2(i)$

```
# Valeur propre  
valprop = rowcoord.apply(lambda x : x**2/I, axis = 0).sum()  
print(valprop.round(3))
```

6.3.4 Coordonnées des variables - modalités

On récupère les coordonnées des variables (qualitatives et quantitatives). Pour les modalités des variables qualitatives, certaines corrections doivent être apportées afin d'avoir les bonnes coordonnées.

```

# Coordonnées des variables-modalités
varmodcoord = pd.DataFrame(model.col_coord_, index = base.columns,
                             columns = index)
varmodcoord.index.name = 'variable'
print(varmodcoord.iloc[:,2].round(3))

```

variable	Dim.1	Dim.2
puissance	0.916	-0.271
cylindree	0.888	-0.039
vitesse	0.703	-0.598
longueur	0.899	-0.142
largeur	0.876	-0.039
hauteur	0.288	0.846
poids	0.915	0.249
CO2	0.891	-0.090
prix	0.940	-0.062

modalité	Dim.1	Dim.2
Autres	0.317	0.489
Europe	-0.038	-0.174
France	-0.237	-0.243
Diesel	0.011	0.407
Essence	-0.010	-0.366
non	-0.139	-0.289
oui	0.356	0.743

TABLE 6.9 – Coordonnées des variables - modalités

On peut visualiser le sens du lien des variables quantitatives avec les axes à travers un cercle de corrélation.

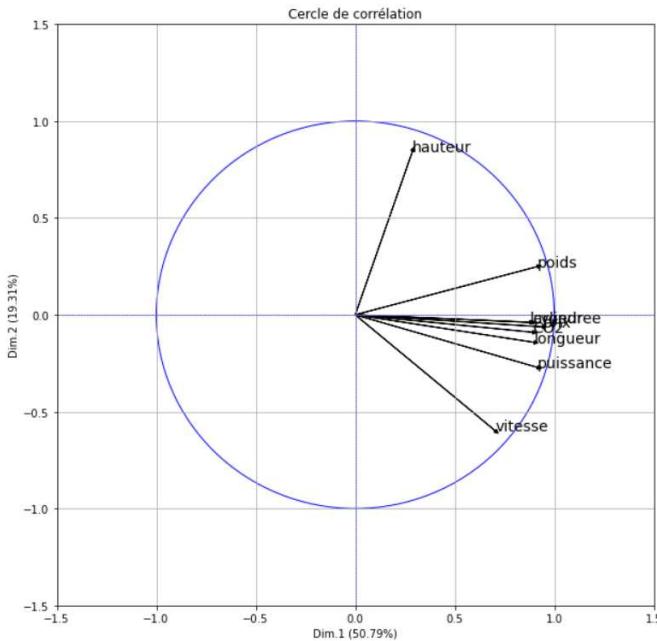


FIGURE 6.6 – Cercle des corrélations

Nous notons un effet taille fort sur le premier axe : les grosses cylindrées sont puissantes, chères, polluantes, et imposantes (longues et larges). Sur le second axe, sans tenir compte des

variables qualitatives, nous notons une opposition entre la hauteur des véhicules et leur vitesse c'est-à-dire à « taille » (au sens du premier axe) égale, les véhicules hauts s'opposent aux rapides. Les variables origine, carburant et type4X4 devraient préciser l'idée.

Pour les modalités des variables qualitatives, nous devons apporter une correction afin d'avoir les bonnes valeurs des coordonnées factorielles. On récupère les coordonnées fournies pour les modalités.

```
# Coordonnées des modalités extraites de PCA
coordmod = pd.DataFrame(np.zeros((fmax,dummies.shape[1])),index=index,
                        columns = dummies.columns).T
for name in dummies.columns:
    coordmod.loc[name,:] = varmodcoord.loc[name,:]
```

Si $u_\alpha(k_q)$ est la valeur de coordonnée fournie par la classe PCA, alors la coordonnée de la modalité k_q sur l'axe α est :

$$G_\alpha(k_q) = u_\alpha(k_q) \sqrt{\frac{\lambda_\alpha}{p_{k_q}}}$$

```
# Coordonnées des modalités corrigées
modcoord = coordmod.apply(lambda x : x/np.sqrt(modweight),
                           axis = 0)*np.sqrt(Eigen.iloc[:,0])
print(modcoord.iloc[:,2].T.round(3))
```

	Autres	Europe	France	Diesel	Essence	non	oui
Dim.1	1.588	-0.156	-1.041	0.044	-0.036	-0.382	2.524
Dim.2	1.512	-0.438	-0.658	0.965	-0.781	-0.492	3.247

TABLE 6.10 – Coordonnées corrigées des modalités

On peut vérifier les valeurs des coordonnées des modalités en calculant les moyennes conditionnelles pour nos variables qualitatives sur les deux premiers axes factoriels.

```
# Concaténation
coord = pd.concat([rowcoord, qual], axis =1)
# Calcul des moyennes conditionnelles
origincoord = pd.pivot_table(coord, values = ['Dim.1','Dim.2'],index = ["origine"],
                               aggfunc = np.mean)
carbucoord = pd.pivot_table(coord, values = ['Dim.1','Dim.2'], index = ["carburant"],
                             aggfunc = np.mean)
type4coord = pd.pivot_table(coord, values = ['Dim.1','Dim.2'], index = ["type4X4"],
                             aggfunc = np.mean)
# Affichage
df = pd.concat([origincoord,carbucoord,type4coord],axis=0)
print(df.round(3).T)
```

	Autres	Europe	France	Diesel	Essence	non	oui
Dim.1	1.588	-0.156	-1.041	0.044	-0.036	-0.382	2.524
Dim.2	1.512	-0.438	-0.658	0.965	-0.781	-0.492	3.247

TABLE 6.11 – Moyennes conditionnelles des variables qualitatives

On constate que les moyennes conditionnelles calculées à partir des coordonnées des individus sont identiques aux coordonnées modifiées $G_\alpha(k)$ des modalités. Ces moyennes représentent les positions des modalités sur les axes factoriels.

Nuage des individus avec modalités

```
fAMD_rowsup_plot(data1 = rowcoord,data2 = modcoord,eigen = Eigen.iloc[:,0],
                  axeI = 0,axeJ = 1,figsize=(12,8))
```

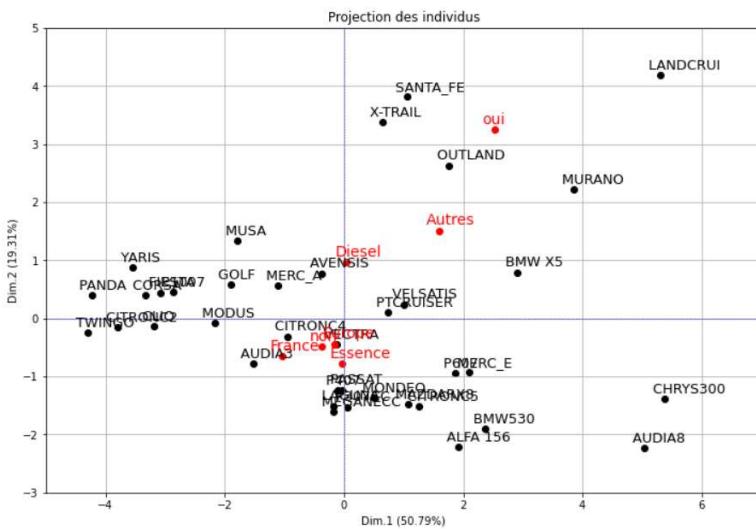


FIGURE 6.7 – Nuage des individus avec les modalités des variables qualitatives

6.3.5 Relations de transition

On applique ici les formules générales de l'ACP au tableau codé.

1) Relation de R^K vers R^I

Soit $G_\alpha(k)$ la coordonnée de la colonne k sur l'axe de range α

a) Cas d'une variable quantitative

On a la formule suivante

$$G_\alpha(k) = \frac{1}{\sqrt{\lambda_\alpha}} \frac{1}{I} \sum_{i=1}^I x_{ik} F_\alpha(i) = \rho(k, F_\alpha) \quad (6.6)$$

```
# relation de transition - coordonnées des variables
transition1 = Z1.T.dot(rowcoord)/(I*np.sqrt(Eigen.iloc[:,0]))
print(transition1.iloc[:,[0,1]].T.round(2))
```

a) Cas d'une modalité k_q de la variable q ayant la fréquence relative p_{k_q}

Puisque F_α est centré, on la formule suivante :

$$\begin{aligned} u_\alpha(k_q) &= \frac{1}{\sqrt{\lambda_\alpha}} \sum_{i=1}^I \frac{1}{I} \left(\frac{y_{ik_q}}{\sqrt{p_{k_q}}} - \sqrt{p_{k_q}} \right) F_\alpha(i) \\ &= \frac{1}{\sqrt{\lambda_\alpha}} \sum_{i=1}^I \frac{1}{I} \frac{y_{ik_q}}{\sqrt{p_{k_q}}} F_\alpha(i) = \frac{\sqrt{p_{k_q}}}{\sqrt{\lambda_\alpha}} F_\alpha(k_q) \end{aligned}$$

La représentation des indicatrices transformées ($u_\alpha(k_q)$) ne semble pas présenter d'avantages par rapport à celle des centres de gravité ($F_\alpha(k_q)$)

```
# relation de transition - coordonnées des modalités
transition2 = Z2.T.dot(rowcoord)/(I*np.sqrt(Eigen.iloc[:,0]))
print(transition2.iloc[:,[0,1]].T.round(3))
```

2) Relation de transition de R^K dans R_I

Cette relation est fondamentale en ACM où elle exprime la position d'un individu par rapport aux modalités qu'il possède. Elle est rarement explicitée en ACP mais est sous-jacente aux interprétations. Pour l'AFDM, elles s'écrivent :

$$F_\alpha(i) = \frac{I}{\sqrt{\lambda_\alpha}} \sum_{k \in K_1} x_{ik} G_s(k) + \frac{1}{\sqrt{\lambda_\alpha}} \sum_{k_q \in K_2} \frac{1}{I} \left(\frac{y_{ik_q}}{\sqrt{p_{k_q}}} - \sqrt{p_{k_q}} \right) G_\alpha(k_q) \quad (6.7)$$

où K_1 et K_2 représentent respectivement le groupe des variables quantitatives et qualitatives.

Le premier membre est celui de l'ACP usuelle. Il exprime qu'un individu se trouve globalement du côté des variables pour lesquelles il a une valeur au-dessus de la moyenne et à l'opposé des variables pour lesquelles il a une valeur au-dessous de la moyenne.

Le second membre peut s'écrire en fonction de $F_s(k_q)$, grâce à relation précédente :

$$\frac{1}{\sqrt{\lambda_\alpha}} \sum_{k_q \in K_2} (y_{ik_q} - p_{k_q}) F_\alpha(k_q) = \frac{1}{\sqrt{\lambda_\alpha}} \sum_{k_q \in K_2} y_{ik_q} F_\alpha(k_q) \quad (6.8)$$

Il exprime qu'un individu est, au coefficient λ_s près, au barycentre des modalités qu'il possède. Finalement, un individu se trouve à la fois du côté des variables pour lesquelles il a une forte valeur et du côté des modalités qu'il possède.

```
# Relation de transition - coordonnées des individus
transition3 = base.dot(varmodcoord)/np.sqrt(Eigen.iloc[:,0])
print(transition3.iloc[:,2].round(3))
```

6.4 Aide à l'interprétation

6.4.1 Analyse du point de vue des individus

1) Cosinus carré des individus

Il mesure la qualité de représentation de l'individu i sur l'axe α . Il peut être cumulable sur α .

$$COS_{\alpha}^2(i) = \frac{F_{\alpha}^2(i)}{\sum_{j=1}^n F_{\alpha}^2(j)} \quad (6.9)$$

```
# Cosinus carré des individus
rowcos2 = rowcoord.apply(lambda x : x**2/rowdisto, axis=0)
#Affichage
print(rowcos2.iloc[:, :2].round(3))
```

2) Contributions des individus

La contribution d'un individu i mesure l'influence de cet individu dans la construction de l'axe α .

$$CTR_{\alpha}(i) = \frac{1}{\lambda_{\alpha}} \frac{1}{I} F_{\alpha}^2(i) \quad (6.10)$$

```
# contributions factorielles des individus
rowcontrib = rowcoord.apply(lambda x: 100*x**2/(I*Eigen.iloc[:, 0]), axis=1)
# Affichage
print(rowcontrib.iloc[:, :2].round(3))
```

En ce qui concerne l'analyse du point de vue des variables, nous allons séparer les variables quantitatives des modalités des variables qualitatives.

6.4.2 Analyse du point de vue des variables quantitatives

1) Cosinus carré des variables quantitatives

Elle mesure la qualité de représentation des variables quantitatives.

$$COS_{\alpha}(k) = G_{\alpha}^2(k), \quad \forall k \in K_1 \quad (6.11)$$

```
# Cosinus carré des variables quantitatives
cos2varquant = quantcoord.apply(lambda x : x**2, axis = 1)
print(cos2varquant.iloc[:, :2].round(3))
```

2) Contribution des variables quantitatives

Elle mesure l'importance du lien des variables avec les axes.

$$CTR_{\alpha}(k) = \frac{G_{\alpha}^2(k)}{\lambda_{\alpha}}, \quad \forall k \in K_1 \quad (6.12)$$

```
# Contributions des variables quantitatives
varquantcontrib = quantcoord.apply(lambda x : 100*x**2/Eigen.iloc[:, 0], axis = 1)
print(varquantcontrib.iloc[:, :2].round(3))
```

6.4.3 Analyse du point de vue des modalités

1) Cosinus carré des modalités

Il mesure la qualité de représentation des modalités.

$$COS_{\alpha}^2(k_q) = G_{\alpha}^2(k_q) \quad (6.13)$$

2) Contributions des modalités

Elle mesure l'importance du lien des modalités avec les axes. On a :

$$CTR_{\alpha}(k) = \frac{u_{\alpha}^2(k_q)}{\lambda_{\alpha}} \quad (6.14)$$

```
# Contributions des modalités
modcontrib= coordmod.apply(lambda x : 100*x**2/Eigen.iloc[:,0], axis = 1)
print(modcontrib.iloc[:,2].round(3))
```

3) Valeur test

Elle permet de tester la significativité de l'écart par rapport à l'origine.

$$VT_{\alpha}(k_q) = \frac{1}{\sqrt{\lambda_{\alpha}}} u_{\alpha}(k_q) \times \sqrt{\frac{(I-1)I_{k_q}}{(I-I_{k_q})}} \quad (6.15)$$

```
# Valeur test des modalités
test = modcoord.apply(lambda x : x*np.sqrt(((I-1)*rowmarge)/(I-rowmarge)))
vtest = test.apply(lambda x : x/np.sqrt(Eigen.iloc[:,0]),axis=1)
print(vtest.iloc[:,2].round(3))
```

6.4.4 Analyse du point de vue des variables qualitatives

1) Cosinus carré des variables qualitatives

Pour avoir le cosinus carré des variables, on applique les étapes suivantes : d'abord, on élève au carré les coordonnées de ses modalités ; ensuite, on fait la somme des valeurs élevées au carré. En d'autres termes, si $u_{\alpha}(k_q)$ est la coordonnée de la modalité q , alors on a :

$$COS_{\alpha}^2(q) = \sum_{k_q \in K_q} u_{\alpha}^2(k_q) = \eta^2(F_{\alpha}, X_q) \quad (6.16)$$

On récupère par groupe de variable les coordonnées de ses modalités et on élève au carré en faisant la somme.

```
# cosinus carrés des variables
varqualcos2 = pd.DataFrame(np.zeros((qual.shape[1],fmax)),columns=index,
                           index = qual.columns)
for name in qual.columns:
    varqualcos2.loc[name,:] = coordmod.loc[np.unique(qual[[name]]),
                                             :].apply(lambda x:x**2,axis=1).sum()
print(varqualcos2.iloc[:,2].T.round(4))
```

variable	origine	carburant	type4X4
Dim.1	0.1581	0.0002	0.1462
Dim.2	0.3286	0.3002	0.6365

TABLE 6.12 – Cosinus carrés des variables qualitatives

On joint les cosinus carrés des deux types de variables.

```
# Concaténation
varcos2 = pd.concat([varquantcos2,varqualcos2], axis = 0)
print(varcos2.iloc[:, :2].round(3))
```

variable	Dim.1	Dim.2
puissance	0.839	0.074
cylindree	0.789	0.002
vitesse	0.495	0.358
longueur	0.807	0.020
largeur	0.767	0.001
hauteur	0.083	0.716
poids	0.838	0.062
CO2	0.795	0.008
prix	0.884	0.004
origine	0.158	0.329
carburant	0.000	0.300
type4X4	0.146	0.637

TABLE 6.13 – Caption

Ce tableau à une particularité car la somme des valeurs en colonnes est égale à la valeur propre associée à l'axe car il permet de vérifier la relation suivante

$$\forall \alpha, \quad \lambda_\alpha = \sum_{j=1}^{K_1} \tau^2(F_\alpha, X_j) + \sum_{j=K_1+1}^{K_2} \eta^2(F_\alpha, X_j) \quad (6.17)$$

```
# Nuage des cosinus carrés des variables
famd_rowsup_plot(data1 = varquantcos2,data2 = varqualcos2,eigen = Eigen.iloc[:, 0],
                  axe1 = 0,axej = 1,figsize=(12,8))
```

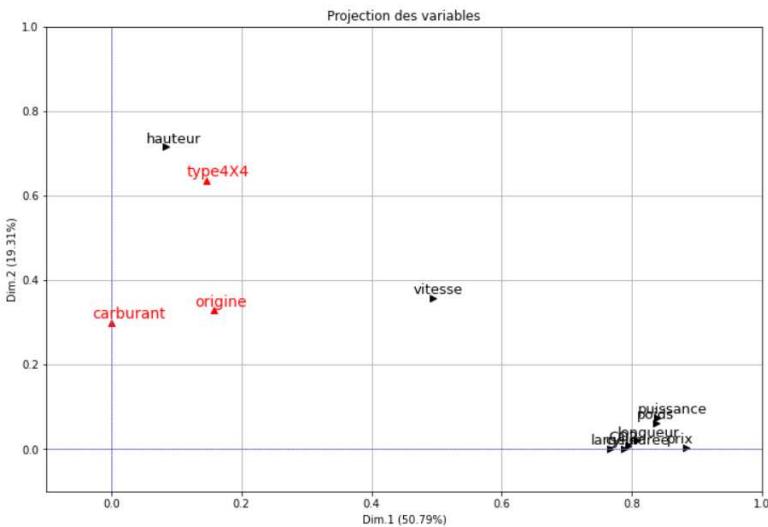


FIGURE 6.8 – Graphe des variables

2) Contribution des variables qualitatives

Elle mesure l'importance du lien des variables avec les facteurs. Elle correspond à la somme des contributions de ces modalités.

$$CTR_\alpha(j) = 100 \times \sum_{k_q \in K_q} CTR_\alpha(k_q) = 100 \times \frac{\text{COS}_\alpha(j)}{\lambda_\alpha} \quad (6.18)$$

```
# Contribution des variables qualitatives
varqualcontrib = varqualcos2.apply(lambda x : 100*x/Eigen.iloc[:,0], axis = 1)
print(varqualcontrib.iloc[:,2].T.round(3))
```

variable	origine	carburant	type4X4
Dim.1	2.395	0.004	2.215
Dim.2	13.093	11.959	25.359

TABLE 6.14 – Contribution des variables qualitatives

6.5 Éléments supplémentaires

6.5.1 Individus supplémentaires

Plusieurs raisons possibles peuvent faire en sorte que le modélisateur puisse mettre certains individus en supplémentaires . Il peut s'agir des individus collectés après coup que l'on aimerait situer par rapport à ceux de l'échantillon d'apprentissage (les individus actifs); des individus appartenant à une population différente (ou spécifique) que l'on souhaite positionner; des observations s'avérant atypiques ou trop influents dans l'AFDM que l'on a préféré écarter. On veut maintenant juger de leur positionnement par rapport aux individus actifs.

```

# chargement des individus supplémentaire
indsup = pd.read_csv('autos_ind_sup.txt',sep='\t',index_col=0)
print(indsup)

      puissance cylindree vitesse longueur largeur hauteur poids \
CLIO          339        4188     227      351      192      181    2235
RENAULT        183        4439     160      387      186      137    1177
CITROEN         88        4161     231      367      182      139     947
TOYOTA          83        1880     232      373      195      149    1229
MERCEDES       311        5361     188      375      159      145     874

      CO2   prix origine carburant type4X4
CLIO     166 77810 France Diesel non
RENAULT    228 31681 France Essence oui
CITROEN    185 51161 Europe Essence non
TOYOTA     118 51233 Autres Diesel non
MERCEDES   339 72876 Europe Diesel oui

```

On sépare notre jeu de données en variables quantitatives et variables qualitatives.

```

# Séparation
quantsup = indsup[quant.columns]
qualsup = indsup[qual.columns]

```

Nous devons préparer notre base de données en standardisant les variables quantitatives et en pondérant le TDC des variables qualitatives par la racine carré de leur poids. La standardisation des variables quantitatives consiste à soustraire et diviser les variables quantitatives des individus supplémentaires par la moyenne et l'écart-type calculé sur les individus actifs. Il en est de même pour les modalités des variables qualitatives.

```

# standardisation des variables quantitatives
meanquant = quant.mean(axis=0)
stdquant = quant.std(ddof = 0, axis=0)
Z3 = quantsup.apply(lambda x : (x-meanquant)/stdquant, axis=1)

```

```

# Standardisation des modalités
meanqual = dummiessup.mean(axis=0)
stdqual = dummiessup.std(ddof = 0, axis=0)
dummiessup = pd.get_dummies(qualsup, prefix = "", prefix_sep = "")
Z4 = dummiessup.apply(lambda x: (x-meanqual)/stdqual, axis = 1)

```

```

# concaténation
Zsup = pd.concat([Z3,Z4],axis=1)

```

En utilisant une formule de transition, la coordonnée d'un individu supplémentaire i' sur l'axe de rang α s'écrit :

$$F_\alpha(i') = \frac{I}{\sqrt{\lambda_\alpha}} \sum_{k \in K_1} x_{i'k} G_s(k) + \frac{1}{\sqrt{\lambda_\alpha}} \sum_{k_q \in K_2} \frac{1}{I} \left(\frac{y_{i'k_q}}{\sqrt{p_{k_q}}} - \sqrt{p_{k_q}} \right) G_\alpha(k_q) \quad (6.19)$$

```
# Coordonnées des individus supplémentaires
rowsupcoord=Zsup.dot(varmodcoord)/np.sqrt(Eigen.iloc[:,0])
print(rowsupcoord.iloc[:,2].round(3))
```

	Dim.1	Dim.2
CLIO	3.631	0.923
RENAULT	0.702	0.526
CITROEN	-0.004	-1.905
TOYOTA	-0.054	0.688
MERCEDES	2.516	1.023

TABLE 6.15 – Coordonnées des individus supplémentaires

La classe PCA de fanalysis dispose d'une méthode `.transform()` qui permet d'obtenir les coordonnées des individus supplémentaires.

```
# Extraction des coordonnées
indsupcoord = model.transform(Zsup)
indsupcoord.columns = index
print(indsupcoord.iloc[:,2].round(3))
```

On représente dans le nuage des individus, les individus supplémentaires.

```
# Nuage des individus avec modalités
famd_rowsup_plot(data1 = rowcoord,data2 = rowsupcoord,eigen = Eigen.iloc[:,0],
                  axe1 = 0,axej = 1,figsize=(12,8))
```

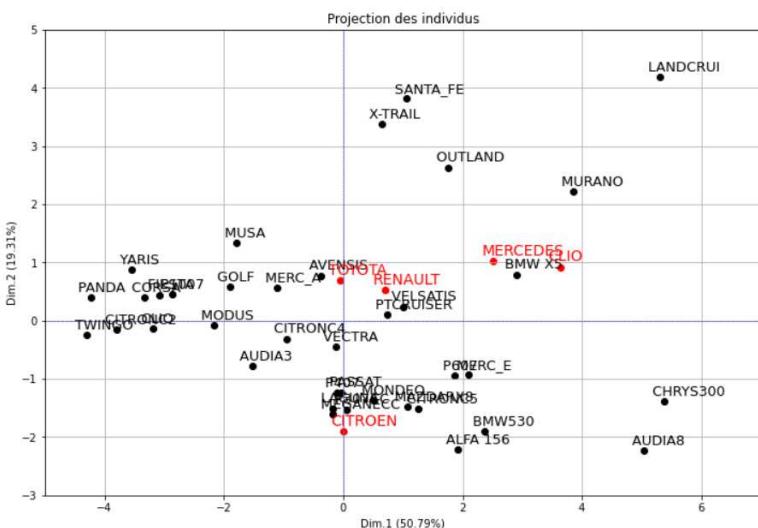


FIGURE 6.9 – Nuage des individus actifs et supplémentaires

6.5.2 Variables supplémentaires

Ce sont des variables non utilisées lors de la construction des axes mais exploitées après coup pour mieux comprendre/commenter les résultats.

1) Variables illustratives quantitatives

Pour des variables quantitatives supplémentaires, on calcule les corrélations avec les facteurs, c'est-à-dire le coefficient de corrélation entre les coordonnées des I individus sur les facteurs et les valeurs prises par les variables illustratives.

$$\rho_{x_{k'}}(F_\alpha) = \frac{\frac{1}{I} \sum_{i=1}^I (x_{ik'} - \bar{x}_{k'}) (F_\alpha(i) - \bar{F}_\alpha)}{\sigma_{x_{k'}} \times \sigma_{F_\alpha}} = \frac{\frac{1}{I} \sum_{i=1}^I F_\alpha(i) (x_{ik'} - \bar{x}_{k'})}{\sigma_{x_{k'}} \times \sqrt{\lambda_\alpha}}, \quad \forall k' \quad (6.20)$$

Il est possible de les placer dans le cercle des corrélations.

```
# chargement des données
varquantsup = pd.read_excel('autos_varquant_sup.xls', index_col=0)
```

On calcule les corrélations entre les deux variables et les axes factoriels (nous calculons uniquement pour les axes 1 et 2).

```
# Corrélation avec les axes 1 et 2
varquantsupcoord = pd.DataFrame(np.zeros((varquantsup.shape[1], 2)),
                                   columns = ['Dim.1', 'Dim.2'],
                                   index = varquantsup.columns)

for name in varquantsup.columns:
    varquantsupcoord.loc[name, :] = np.corrcoef(varquantsup[name],
                                                rowcoord.iloc[:, :2],
                                                rowvar = False)[0, 1:]

print(varquantsupcoord.round(3))
```

	Dim.1	Dim.2
age	0.242	-0.132
kilometrage	0.124	-0.009

TABLE 6.16 – Corrélation des variables supplémentaires avec les axes 1 et 2

On représente dans le cercle de corrélation, nos variables age et kilometrage.

```
# Nuage des variables actives et illustratives
famd_varsup_plot(data1 = quantcoord, data2 = varquantsupcoord, eigen=Eigen.iloc[:, 0],
                  axe1=0, axe2=1, figsize=(10, 10))
```

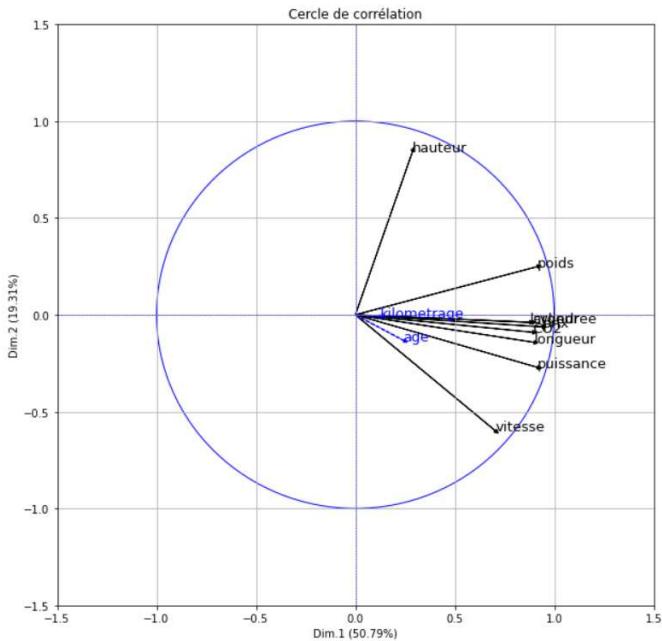


FIGURE 6.10 – Cercle des corrélations des variables actives et supplémentaires

2) Variables illustratives qualitatives

Nous disposons d'une variable qualitative, syntaxe, liée à la politique fiscale de l'administration. Comment la situer ?

```
# chargement des données
varqualsup = pd.read_excel('autos_varqual_sup.xls', index_col=0)
```

On calcule les moyennes conditionnelles de la variable syntaxe sur les axes factoriels.

```
# Concaténation
concatenate = pd.concat([rowcoord, varqualsup], axis = 1)
# Moyennes conditionnelles sur les axes 1 et 2
varqualsupcoord = pd.pivot_table(concatenate, values = ['Dim.1','Dim.2'],
                                 index = ["syntaxe"], aggfunc = np.mean)
print(varqualsupcoord.iloc[:, :2])
```

On calcule les valeurs-tests des modalités de la variable syntaxe.

```
# Valeur test des modalités
supmarge = pd.get_dummies(varqualsup, prefix = "", prefix_sep = "").sum(axis=0)
testsup = varqualsupcoord.apply(lambda x:x*np.sqrt(((I-1)*supmarge)/(I-supmarge)))
vtestsup = testsup.apply(lambda x : x/np.sqrt(Eigen.iloc[:, 0][:2]),axis=1)
print(vtestsup.round(3))
```

syntaxe	Dim.1	Dim.2
non	1.201	-2.129
oui	-3.925	2.989

TABLE 6.17 – Valeurs-tests des modalités supplémentaires

Résultats

	Coordonnées			Contributions			Cosinus carrés		
	Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3
puissance	0.916	-0.271	0.126	12.704	2.936	1.220	0.839	0.074	0.016
cylindree	0.888	-0.039	-0.046	11.951	0.061	0.160	0.789	0.002	0.002
vitesse	0.703	-0.598	-0.039	7.495	14.263	0.115	0.495	0.358	0.001
longueur	0.899	-0.142	-0.130	12.229	0.804	1.292	0.807	0.020	0.017
largeur	0.876	-0.039	-0.187	11.621	0.060	2.693	0.767	0.001	0.035
hauteur	0.288	0.846	-0.028	1.260	28.519	0.059	0.083	0.716	0.001
poids	0.915	0.249	-0.099	12.693	2.469	0.757	0.838	0.062	0.010
CO2	0.891	-0.090	0.327	12.037	0.323	8.202	0.795	0.008	0.107
prix	0.940	-0.062	-0.126	13.397	0.154	1.218	0.884	0.004	0.016

TABLE 6.18 – Analyse des variables quantitatives

modalité	Coordonnées			Contributions			Valeur-test		
	Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3
Autres	1.588	1.512	0.896	1.522	9.544	12.409	2.246	3.468	2.851
Europe	-0.156	-0.438	-1.096	0.022	1.200	27.832	-0.298	-1.357	-4.712
France	-1.041	-0.658	0.575	0.851	2.349	6.643	-1.778	-1.821	2.208
Diesel	0.044	0.965	-0.837	0.002	6.609	18.405	0.094	3.333	-4.010
Essence	-0.036	-0.781	0.678	0.002	5.350	14.899	-0.094	-3.333	4.010
non	-0.382	-0.492	-0.103	0.291	3.337	0.539	-2.326	-4.853	-1.406
oui	2.524	3.247	0.679	1.923	22.022	3.558	2.326	4.853	1.406

TABLE 6.19 – Analyse des modalités

Modele	Distro ²	Poids	Inertie	Coordonnées			Contributions			Cosinus		
				Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3	Dim.1	Dim.2	Dim.3
SANTAFE	16.928	0.026	0.445	1.061	3.820	0.296	0.448	15.302	0.176	0.066	0.862	0.005
MURANO	24.599	0.026	0.647	3.855	2.214	1.803	5.924	5.141	6.555	0.604	0.199	0.132
LANDCRUI	48.584	0.026	1.279	5.309	4.192	0.095	11.233	18.420	0.018	0.580	0.362	0.000
OUTLAND	14.438	0.026	0.380	1.760	2.634	0.731	1.234	7.273	1.077	0.215	0.480	0.037
X-TRAIL	13.344	0.026	0.351	0.637	3.376	0.468	0.162	11.951	0.443	0.030	0.854	0.016
CITRONC5	6.400	0.026	0.168	1.264	-1.519	1.003	0.637	2.420	2.028	0.250	0.361	0.157
P607	8.946	0.026	0.235	1.861	-0.945	-0.485	1.380	0.937	0.473	0.387	0.100	0.026
VELSATIS	6.778	0.026	0.178	1.006	0.243	-0.740	0.403	0.062	1.105	0.149	0.009	0.081
LAGUNA	4.278	0.026	0.113	-0.176	-1.505	0.916	0.012	2.374	1.691	0.007	0.529	0.196
P407	4.371	0.026	0.115	-0.099	-1.243	0.755	0.004	1.619	1.149	0.002	0.353	0.130
CITRONC4	4.854	0.026	0.128	-0.956	-0.309	-0.492	0.365	0.100	0.489	0.188	0.020	0.050
CLIO	13.678	0.026	0.360	-3.185	-0.128	-0.153	4.044	0.017	0.047	0.742	0.001	0.002
CITRONC2	16.641	0.026	0.438	-3.793	-0.155	1.192	5.736	0.025	2.866	0.865	0.001	0.085
P307CC	4.465	0.026	0.118	0.062	-1.524	1.033	0.002	2.436	2.151	0.001	0.520	0.239
MODUS	7.993	0.026	0.210	-2.174	-0.073	1.107	1.883	0.006	2.471	0.591	0.001	0.153
MEGANECC	4.508	0.026	0.119	-0.180	-1.599	0.900	0.013	2.682	1.632	0.007	0.567	0.180
P1007	11.967	0.026	0.315	-2.863	0.448	1.077	3.267	0.210	2.340	0.685	0.017	0.097
TWINGO	21.549	0.026	0.567	-4.304	-0.240	1.363	7.385	0.061	3.747	0.860	0.003	0.086
BMW530	10.780	0.026	0.284	2.362	-1.901	-0.541	2.223	3.788	0.591	0.517	0.335	0.027
MERC_E	10.235	0.026	0.269	2.092	-0.932	-2.021	1.744	0.911	8.240	0.428	0.085	0.399
AUDIA8	34.807	0.026	0.916	5.044	-2.234	-0.887	10.142	5.231	1.586	0.731	0.143	0.023
MONDEO	7.382	0.026	0.194	0.498	-1.364	-0.752	0.099	1.949	1.140	0.034	0.252	0.077
VECTRA	4.345	0.026	0.114	-0.131	-0.451	-1.825	0.007	0.214	6.718	0.004	0.047	0.767
PASSAT	4.033	0.026	0.106	-0.038	-1.229	-0.393	0.001	1.585	0.312	0.000	0.375	0.038
BMW X5	15.815	0.026	0.416	2.909	0.793	-1.942	3.373	0.659	7.607	0.535	0.040	0.239
MERC_A	6.006	0.026	0.158	-1.115	0.573	-1.672	0.495	0.345	5.636	0.207	0.055	0.465
ALFA 156	12.796	0.026	0.337	1.926	-2.211	0.137	1.479	5.124	0.038	0.290	0.382	0.001
AUDIA3	5.385	0.026	0.142	-1.530	-0.775	-0.396	0.933	0.629	0.316	0.435	0.111	0.029
GOLF	7.793	0.026	0.205	-1.905	0.576	-1.706	1.447	0.348	5.869	0.466	0.043	0.373
MUSA	9.337	0.026	0.246	-1.794	1.343	-1.552	1.282	1.892	4.858	0.345	0.193	0.258
FIESTA	12.922	0.026	0.340	-3.078	0.444	-1.553	3.776	0.206	4.864	0.733	0.015	0.187
CORSA	14.386	0.026	0.379	-3.341	0.397	-1.388	4.450	0.166	3.884	0.776	0.011	0.134
PANDA	21.969	0.026	0.578	-4.237	0.404	0.056	7.156	0.171	0.006	0.817	0.007	0.000
AVENSISS	5.647	0.026	0.149	-0.379	0.775	-0.289	0.057	0.630	0.169	0.025	0.106	0.015
CHRYSS300	38.330	0.026	1.009	5.375	-1.393	1.124	11.516	2.034	2.549	0.754	0.051	0.033
MAZDARX8	12.644	0.026	0.333	1.070	-1.474	1.746	0.456	2.278	6.147	0.091	0.172	0.241
PTCRUISER	6.263	0.026	0.165	0.736	0.100	1.555	0.216	0.010	4.876	0.086	0.002	0.386
YARIS	18.801	0.026	0.495	-3.547	0.871	1.432	5.014	0.796	4.136	0.669	0.040	0.109

TABLE 6.20 – Analyse des individus

CHAPITRE 7

ANOVA À 1 FACTEUR

Sommaire

7.1 Exemple introductif	186
7.2 Le modèle d'analyse de la variance à 1 facteur	188

7.1 Exemple introductif

On dispose de 30 chambres de même type avec leur frais de location. On dispose également des quartiers pour les chambres : A, B et C. Les frais de location des chambres se trouvent dans le tableau ci-après :

Quartier	Frais de location
A	20, 15, 21, 20, 15, 16, 18, 17, 15, 12, 21, 20
B	50, 60, 65, 45, 40, 70, 65, 30
C	35, 35, 32, 40, 45, 25, 20, 21, 30, 29

TABLE 7.1 – Fais de location par quartier

On cherche à expliquer le loyer (variable quantitative) par le type de quartier (variable qualitative).

```
# Create dataset
A = [20, 15, 21, 20, 15, 16, 18, 17, 15, 12, 21, 20]
B = [50, 60, 65, 45, 40, 70, 65, 30]
C = [35, 35, 32, 40, 45, 25, 20, 21, 30, 29]
```

Nos listes créées n'ont pas la même longueur.

```
# longueur des listes
for x, y in zip(['A','B','C'], [len(A), len(B),len(C)]):
    print(f'La liste {x} contient {y} éléments.')
```

La liste A contient 12 éléments.
La liste B contient 8 éléments.
La liste C contient 10 éléments.

Nous créons notre variable quartier qui contiendra le type de quartier du logement.

```
# Regroupement des quartiers
```

```
q_A = ['A']*len(A)
```

```
q_B = ['B']*len(B)
```

```
q_C = ['C']*len(C)
```

```
# Somme
```

```
frais = A + B + C
```

```
quartier = q_A + q_B + q_C
```

On crée notre dataframe

```
# création du dataframe
```

```
import pandas as pd
```

```
df = pd.DataFrame({'quartier': quartier, 'montant': frais})
```

```
print(df)
```

	quartier	montant
0	A	20
1	A	15
2	A	21
3	A	20
4	A	15
5	A	16
6	A	18
7	A	17
8	A	15
9	A	12
10	A	21
11	A	20
12	B	50
13	B	60
14	B	65
15	B	45
16	B	40
17	B	70
18	B	65
19	B	30
20	C	35
21	C	35
22	C	32
23	C	40
24	C	45
25	C	25
26	C	20
27	C	21
28	C	30
29	C	29

Représentation graphique

On visualise le boxplot de notre variable par quartier.

```
# Boxplot
import matplotlib.pyplot as plt
import seaborn as sns
ax = sns.boxplot(x='quartier', y='montant', data=df)
ax = sns.swarmplot(x='quartier', y='montant', data=df, color="#7d0013")
plt.title('Boxplot des frais de location par quartier')
plt.show()
```

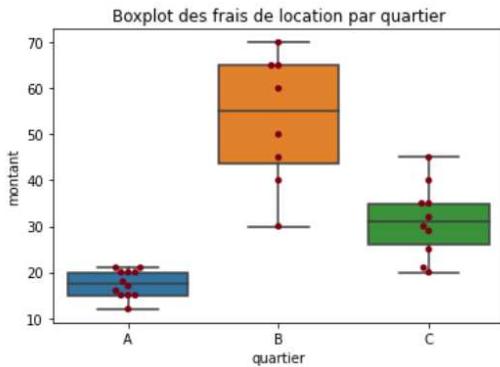


FIGURE 7.1 – Boxplot des frais de location par quartier

7.2 Le modèle d'analyse de la variance à 1 facteur

7.2.1 Notations

On appellera :

- y : la variable quantitative ;
- y_{ik} : la valeur de la variable y pour l'observation i du groupe k ;
- K : le nombre total de groupe ;
- n_k : le nombre d'observations dans le groupe k ;
- n : le nombre total d'observations. Il correspond à $n = \sum_{k=1}^K n_k$

On a les indicateurs de tendances centrales suivantes :

a) Moyenne par groupes

La moyenne par groupe est calculée avec la formule :

$$\bar{y}_k = \frac{1}{n_k} \sum_{i=1}^{n_k} y_{ik}, \quad \forall k$$

C'est la moyenne de y dans le groupe k .

b) Moyenne générale

La moyenne générale ou moyenne globale est calculée avec la formule ci-après :

$$\begin{aligned}\bar{y} &= \frac{1}{n} \sum_{k=1}^K \sum_{i=1}^{n_k} y_{ik} \\ &= \frac{1}{n} \sum_{k=1}^K n_k \bar{y}_k\end{aligned}$$

Pour nos données, on a :

```
# Moyenne par groupe et moyenne générale
import numpy as np
mean = pd.pivot_table(df, values='montant', columns=['quartier'], aggfunc=np.mean)
mean.loc[:, 'gen. mean'] = np.mean(df['montant'])
print(mean)

quartier      A      B      C  gen. mean
montant    17.5  53.125  31.2   31.566667
```

Remarque :

Le moyenne globale est une moyenne des moyennes des groupes pondérées par leurs effectifs.

On calcule également les indicateurs de dispersion.

c) Variance du groupe k

La variance du groupe k est calculée comme suit :

$$\sigma_k^2(y) = \frac{1}{n_k} \sum_{i=1}^{n_k} (y_{ik} - \bar{y}_k)^2 \quad \forall k$$

C'est la variance de y à l'intérieur du groupe k .

d) Variance globale

Sa formule est :

$$\sigma^2(y) = \frac{1}{n} \sum_{k=1}^K \sum_{i=1}^{n_k} (y_{ik} - \bar{y})^2$$

```
# Variance par groupe et variance globale
variance = pd.pivot_table(df, values='montant', columns=['quartier'], aggfunc=np.var)
variance.loc[:, 'gen. variance'] = np.var(df['montant'])
print(variance)

quartier      A      B      C  gen. variance
montant  8.636364  199.553571  63.511111  271.912222
```

7.2.2 Le modèle et ses hypothèses

Le modèle ANOVA à 1 facteur s'écrit simplement :

$$y_{ik} = \alpha + \beta_k + \varepsilon_{ik} \quad (7.1)$$

Autrement dit, le montant du loyer est la résultante de trois effets :

- Un effet commun à tous les chambres : α
- Un effet spécifique au groupe et commun aux individus du même groupe : β_k
- Un effet de perturbation propre à chaque individu : ε_{ik}

Remarque :

Le modèle a trop de paramètres inconnus pour être estimable de manière bien déterminée. On rajoute donc la contrainte suivante :

$$\sum_{k=1}^K \frac{n_k}{n} \beta_k = 0 \quad (7.2)$$

C'est-à-dire que les effets spécifiques sont en moyenne nuls. Ils sont vus comme des effets différentiels à l'effet global.

Les ε_{ik} sont considérés comme des perturbations aléatoires. Ils matérialisent l'effet éventuel de tous les facteurs influençant y que l'on n'a pas pris en compte dans le modèle.s

Hypothèse du modèle

On suppose que les ε_{ik} sont des aléas de moyenne nulle, indépendants et identiquement distribués et supposés gaussien.

7.2.3 Les estimateurs du modèle

Les estimateurs des différents modèles sont calculés de la manière suivante :

- Effet commun : $\hat{\beta} = \bar{y}$;
- Effet spécifique du groupe k : $\hat{\beta}_k = \hat{y}_k - \bar{y}$;
- Perturbation : $\hat{\varepsilon}_{ik} = y_{ik} - \bar{y}_k$

```

# Estimation des paramètres
hat_b = np.mean(df['montant'])
print('Estimation des paramètres \nalpha : {}'.format(round(hat_b,3)))
for value in ['A','B','C']:
    val = np.mean(df[df.quartier ==value]['montant']) - hat_b
    print(f'beta_{value} : {round(val,3)}')

```

Estimation des paramètres
alpha : 31.567
beta_A : -14.067
beta_B : 21.558
beta_C : -0.367

7.2.4 Équation de décomposition de la variance

Elle consiste à écrire la variance totale sous forme de variance intra groupe et variance inter groupe.

$$\begin{aligned}
\sigma^2(y) &= \frac{1}{n} \sum_{k=1}^K \sum_{i=1}^{n_k} (y_{ik} - \bar{y})^2 = \frac{1}{n} \sum_{k=1}^K \sum_{i=1}^{n_k} [(y_{ik} - \bar{y}_k) + (\bar{y}_k - \bar{y})]^2 \\
&= \frac{1}{n} \sum_{k=1}^K \sum_{i=1}^{n_k} [(y_{ik} - \bar{y}_k)^2 + (\bar{y}_k - \bar{y})^2 + 2(y_{ik} - \bar{y}_k)(\bar{y}_k - \bar{y})]^2 \\
&= \frac{1}{n} \sum_{k=1}^K \sum_{i=1}^{n_k} (y_{ik} - \bar{y}_k)^2 + \frac{1}{n} \sum_{k=1}^K \sum_{i=1}^{n_k} (\bar{y}_k - \bar{y})^2 + \frac{2}{n} \sum_{k=1}^K \sum_{i=1}^{n_k} (y_{ik} - \bar{y}_k)(\bar{y}_k - \bar{y}) \\
&= \frac{1}{n} \sum_{k=1}^K n_k \left(\frac{1}{n_k} \sum_{i=1}^{n_k} (y_{ik} - \bar{y}_k)^2 \right) + \frac{1}{n} \sum_{k=1}^K n_k (\bar{y}_k - \bar{y})^2 + \frac{2}{n} \sum_{k=1}^K (\bar{y}_k - \bar{y}) \underbrace{\left[\sum_{i=1}^{n_k} (y_{ik} - \bar{y}_k) \right]}_{=0} \\
&= \frac{1}{n} \sum_{k=1}^K n_k \sigma_k^2(y) + \frac{1}{n} \sum_{k=1}^K n_k (\bar{y}_k - \bar{y})^2
\end{aligned}$$

On fait la remarque suivante :

- $\frac{1}{n} \sum_{k=1}^K n_k \sigma_k^2(y)$ est la moyenne des variances internes aux groupes pondérées par leurs effectifs.
Il s'agit donc d'une variance interne moyenne ou **variance dans les groupes**.
- $\frac{1}{n} \sum_{k=1}^K n_k (\bar{y}_k - \bar{y})^2$ est la variance des moyennes des différents groupes, soit une **variance entre les groupes**.

Ainsi, on a :

$$\text{variance totale} = \text{variance inter groupe} + \text{variance intra groupe} \quad (7.3)$$

où :

- **La variance intra** représente la moyenne des variances (conditionnelles) et quantifie la part de variabilité intrinsèque de y : $\sigma_{\text{intra}}^2(y) = \frac{1}{n} \sum_{k=1}^K n_k \sigma_k^2(y)$
- **La variance inter** représente la variance des moyennes (conditionnelles) et mesure l'hétérogénéité des populations : $\sigma_{\text{inter}}^2(y) = \frac{1}{n} \sum_{k=1}^K n_k (\bar{y}_k - \bar{y})^2$

```
# Equation de décomposition de la variance
def variance_decomposition(categories, values):
    cat = np.unique(categories, return_inverse=True)[1]
    values = np.array(values)

    ssw = 0
    ssb = 0
    for i in np.unique(cat):
        subgroup = values[np.argwhere(cat == i).flatten()]
        ssw += np.sum((subgroup - np.mean(subgroup))**2)
        ssb += len(subgroup)*(np.mean(subgroup) - np.mean(values))**2

    return ssw/values.shape[0], ssb/values.shape[0]
ssw, ssb = variance_decomposition(df['quartier'], df['montant'])
print(f'variance intra : {round(ssw,3)}\nvariance inter : {round(ssb,3)}')

variance intra : 68.783
variance inter : 203.13
```

7.2.5 Le critère et le test

L'idée est la suivante :

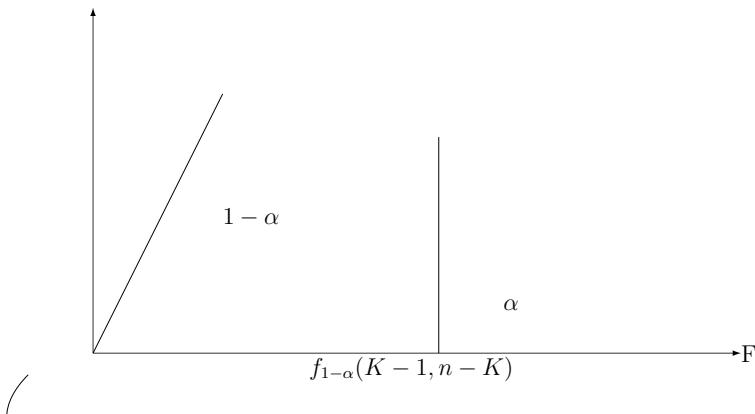
- Si le rapport $\frac{\text{variance inter}}{\text{variance intra}}$ est **grand**, on admet la significativité de l'influence dépisté du facteur groupe sur la variable y .

Formulation du test

Sous l'hypothèse que le facteur de groupe n'induit aucune différence sur y entre les groupes (c'est-à-dire *aucun effet spécifique*, autrement dit $\beta_k = 0$), la statistique de test est :

$$F_{\text{stat}} = \frac{\frac{1}{K-1} \times \text{variance inter}}{\frac{1}{n-K} \times \text{variance intra}} \quad (7.4)$$

Cette statistique suit une loi de Fisher à $K-1$ et $n-K$ degrés de liberté noté $F(K-1, n-K)$. On compare cette statistique F au fractile d'ordre 95% de la loi de Fisher $F_{0.95}(K-1, n-K)$



On voit que

- Si F_{stat} tombe dans la région 1 : l'influence du groupe n'est pas significative ;
- Si F_{stat} tombe dans la zone 2 : l'influence du groupe est significative.

Règle de décision :

- Si $F_{stat} > F_{1-\alpha}(K-1, n-K)$: on rejette H_0 , c'est-à-dire l'influence du groupe est significative
- Si $F_{stat} \leq F_{1-\alpha}(K-1, n-K)$: on accepte H_0 , c'est-à-dire l'influence du groupe n'est pas significative.

Source de variation	degré de liberté	Somme des carrés	Carré moyen	Stat. f	Significativité
Inter	$K - 1$	$n \times \sigma_{\text{inter}}^2(y)$	$\frac{n \times \sigma_{\text{inter}}^2(y)}{K - 1}$	F_{stat}	$pvalue$
Intra	$n - K$	$n \times \sigma_{\text{intra}}^2(y)$	$\frac{n \times \sigma_{\text{intra}}^2(y)}{n - K}$		
Totale	$nK - 1$	$n \times \sigma^2(y)$			

TABLE 7.2 – Équation de décomposition de la variance

```
# Calcul de F stat
K = len(np.unique(df['quartier']))
n = df.shape[0]

# numérateur et dénominateur
num = ssb/(K-1)
den = ssw/(n-K)
f_stat = num/den

# calcul de la pvalue
from scipy.stats import f
dd1 = K-1
dd2 = n-K
```

```

pvalue = f.sf(f_stat, dd11, dd12)
print('F stat : %.3f' % (f_stat))
print('pvalue : %.3f' % (pvalue))

F stat : 39.868
pvalue : 0.000

```

Nous avons une pvalue significative ($p < 0.05$). En conclusion, il y a une différence significative entre les quartiers.

Mise en oeuvre avec statsmodels

```

# get ANOVA table as R like output
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Ordinary Least Squares (OLS) model
model = ols('montant ~ quartier', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
# output (ANOVA F and p value)
print(anova_table)

      sum_sq      df          F      PR(>F)
quartier  6093.891667    2.0  39.868444  8.733022e-09
Residual   2063.475000   27.0        NaN           NaN

```

[Statsmodels](#) nous renvoie la table réduite identique à celle du tableau 7.2. On voit que les résultats sont identiques à ceux calculés manuellement.

CHAPITRE 8

ANALYSE FACTORIELLE DISCRIMINANTE (AFD)

Sommaire

8.1	Données - Problématique	195
8.2	Mise en œuvre de l'AFD	200
8.3	Évaluation globale du modèle	210

L'Analyse Factorielle Discriminante (AFD) forme avec l'Analyse de la Variance (ANOVA) à un ou plusieurs facteurs, un corpus de méthodes d'études des rapports entre variables quantitatives et variables qualitatives. On suppose d'une part qu'un certain nombre d'unités statistiques (appelées individus) sont décrites des variables quantitatives et d'autre part, on dispose sur ces mêmes individus, de groupement en classes (ou d'observations qualitatives). Pour chacune de ces méthodes, une variable est censée être explicable à partir des autres. En Analyse de la Variance, on a qu'une seule variable quantitative, et c'est elle que l'on cherche à expliquer à partir des variables de classe (qualitatives), tandis qu'en analyse factorielles discriminante, c'est la variable de classe qui est seule, et que l'on essaie d'expliquer à partir des variables quantitatives. En d'autre termes, dans les deux méthodes, on cherche les rapports pouvant exister entre la dispersion des individus du point de vue des variables quantitatives et les regroupements de ces individus en classe.

En analyse factorielle discriminante, la variable classe est censément explicable, donc prédictible, à partir des variables quantitatives. Une fois que les rapports entre ces variables (qualitative et quantitatives) sont dégagés, on cherchera, connaissant la description d'un nouvel individu à l'aide des variables quantitatives, à en prédire la classe de manière plausible.

8.1 Données - Problématique

8.1.1 Données

On considère une population est divisée en K classes au moyen d'un critère qualitatif y . Chaque individu de la population est décrit par J variables numériques X_1, \dots, X_J . L'analyse factorielle discriminante (ou *canonical discriminant analysis*) permet de mettre en évidence les différences

entre les classes au niveau des variables X_1, \dots, X_J et de visualiser les données lorsqu'il y a plus de deux classes. Ce tableau décrit I individus à l'aide de J variables explicatives X_1, \dots, X_J et une variable à expliquer y . x_{ij} est la valeur prise par l'individu i pour la variable explicative j . y_i est la modalité prise par l'individu i pour la variable y .

	cible	Variables explicatives		
		X_1	X_j	X_p
Individus	1			
	i	y_i	...	x_{ij}
	I	

TABLE 8.1 – Données AFD

Nous allons illustrer ce chapitre à travers l'exemple des Vins de Bordeaux [voir [Ten07]].

On cherche à relier la qualité des vins de Bordeaux à des caractéristiques météorologiques. La variable à expliquer y est la qualité du vin et prend 3 modalités : 1 = bon, 2 = moyen et 3 = médiocre. Les variables explicatives de la qualité du vin sont les suivantes : X_1 (Somme des températures moyennes journalières($^{\circ}\text{C}$)), X_2 (Durée d'insolation (h)), X_3 (Nombre de jours de grande chaleur) et X_4 (Hauteur des pluies (mm)).

```
# Chargement des données
import pandas as pd

data = pd.read_excel('wine_quality.xls', index_col=1)
# Suppression Obs.
donnee = data.drop(['Obs.'], axis=1)

# Dimension
import numpy as np
I = donnee.shape[0]
J = donnee.shape[1]-1
K = len(np.unique(donnee['Qualite']))
```

La variable qualitative y partage la population en K sous - groupes de tailles respectives I_1, \dots, I_K . et on note x_{ij}^k la valeur prise par l'individu i du groupe k pour la variable j . \bar{x}_j^k est la moyenne de la variable j dans le groupe k ($k = 1, \dots, K$) et \bar{x}_j la moyenne de la variable j sur l'ensemble de l'échantillon.

```
# Effectifs et proportions par groupes
effy = donnee['Qualite'].value_counts()
propy = donnee['Qualite'].value_counts(normalize=True)
df = pd.concat([effy, propy.round(3)], axis=1)
df.columns = ['effectifs', 'proportions']
display(df)
```

modalité	effectifs	proportions
Mediocre	12	0.353
Bon	11	0.324
Moyen	11	0.324

TABLE 8.2 – Distribution de la variable Qualité

Obs.	Année	Température	Soleil	Chaleur	Pluie	Qualité
1	1924	3064	1201	10	361	Moyen
2	1925	3000	1053	11	338	Mediocre
3	1926	3155	1133	19	393	Moyen
4	1927	3085	970	4	467	Mediocre
5	1928	3245	1258	36	294	Bon
6	1929	3267	1386	35	225	Bon
7	1930	3080	966	13	417	Mediocre
8	1931	2974	1189	12	488	Mediocre
9	1932	3038	1103	14	677	Mediocre
10	1933	3318	1310	29	427	Moyen
11	1934	3317	1362	25	326	Bon
12	1935	3182	1171	28	326	Mediocre
13	1936	2998	1102	9	349	Mediocre
14	1937	3221	1424	21	382	Bon
15	1938	3019	1230	16	275	Moyen
16	1939	3022	1285	9	303	Moyen
17	1940	3094	1329	11	339	Moyen
18	1941	3009	1210	15	536	Mediocre
19	1942	3227	1331	21	414	Moyen
20	1943	3308	1366	24	282	Bon
21	1944	3212	1289	17	302	Moyen
22	1945	3361	1444	25	253	Bon
23	1946	3061	1175	12	261	Moyen
24	1947	3478	1317	42	259	Bon
25	1948	3126	1248	11	315	Moyen
26	1949	3458	1508	43	286	Bon
27	1950	3252	1361	26	346	Moyen
28	1951	3052	1186	14	443	Mediocre
29	1952	3270	1399	24	306	Bon
30	1953	3198	1259	20	367	Bon
31	1954	2904	1164	6	311	Mediocre
32	1955	3247	1277	19	375	Bon
33	1956	3083	1195	5	441	Mediocre
34	1957	3043	1208	14	371	Mediocre

TABLE 8.3 – Qualité des vins de Bordeaux

On calcule les moyennes et les écarts-types par groupe pour chacune des variables.

```
# Moyenne et écart - type par groupe
import numpy as np
```

```

meanstd = pd.pivot_table(donnee, values = donnee.columns[:-1],
                         index = donnee.columns[-1], aggfunc = [np.mean,np.std])
meanstd.T.round(3)

```

	Qualité	Bon	Moyen	Médiocre
mean	Chaleur	28.545	16.455	12.083
std	Pluie	305.000	339.636	430.333
mean	Soleil	1363.636	1262.909	1126.417
std	Température	3306.364	3140.909	3037.333
mean	Chaleur	8.802	6.729	6.302
std	Pluie	52.293	54.986	104.846
mean	Soleil	80.306	71.941	88.393
std	Température	92.057	100.045	69.339

TABLE 8.4 – Moyennes et écarts-types par niveau de qualité

On calcule également les moyennes et les écarts-types globaux.

```

# Moyennes et écarts-types globaux
meanvar = donnee[donnee.columns[:-1]].mean(axis=0)
stdvar = donnee[donnee.columns[:-1]].std(ddof=1, axis=0)
df2 = pd.concat([meanvar.round(3), stdvar.round(3)], axis=1)
df2.columns = ['moyenne', 'écart-type']
display(df2.T)

```

	Température	Soleil	Chaleur	Pluie
moyenne	3157.882	1247.324	18.824	360.441
écart-type	141.184	126.623	10.017	91.402

TABLE 8.5 – Moyennes et écarts-types globaux

8.1.2 Analyse descriptives des variables

Nous mesurons le pouvoir discriminant de chaque variables X_j en utilisant l'analyse de la variance à un facteur (cf chapitre 7). Pour cela, nous utilisons le rapport de corrélation définit par :

$$\eta^2(X_j, y) = \frac{\text{Somme des carrés interclasses}}{\text{Somme des carrés totale}} \quad (8.1)$$

```

# Pouvoir discriminant
def Eta2(categories, value):
    K = len(np.unique(categories, return_inverse=True)[0])
    n = value.shape[0]

    cat = np.unique(categories, return_inverse=True)[1]
    values = np.array(value)

```

```

scintra = 0
scinter = 0
for i in np.unique(cat):
    subgroup = values[np.argwhere(cat == i).flatten()]
    scintra += np.sum((subgroup-np.mean(subgroup))**2)
    scinter += len(subgroup)*(np.mean(subgroup)-np.mean(values))**2

eta2 = scinter/(scintra+scintra)
f_stat = (scinter/(K-1))/(scintra/(n-K))
# calcul de la pvalue
from scipy.stats import f
pvalue = np.round(f.sf(f_stat, K-1, n-K),4)
return dict({'Sum. Intra':round(scintra,4), 'Sum. Inter':round(scinter,4),
            'Eta2':round(eta2,4), 'F':round(f_stat,4), 'pvalue':pvalue})

# Application
eta2 = dict()
for name in donnee.columns[:-1]:
    eta2[name]=Eta2(donnee['Qualité'], donnee[name])
print(pd.DataFrame(eta2).T)

```

	Sum. Intra	Sum. Inter	Eta2	F	pvalue
Température	237722.1212	420067.4082	0.6386	27.3893	0.0000
Soleil	202192.3712	326909.0700	0.6179	25.0607	0.0000
Chaleur	1664.3712	1646.5700	0.4973	15.3342	0.0000
Pluie	178499.2121	97191.1702	0.3525	8.4396	0.0012

TABLE 8.6 – Rapport de corrélation

On visualise les boxplots pour les différentes variables.

```

# Boxplot
import seaborn as sns
import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 2, figsize=(12,8))
fig.suptitle('Boxplot', fontsize=13)
sns.boxplot(ax=axes[0, 0], data=donnee, x='Qualité', y=donnee.columns[0])
sns.boxplot(ax=axes[0, 1], data=donnee, x='Qualité', y=donnee.columns[1])
sns.boxplot(ax=axes[1, 0], data=donnee, x='Qualité', y=donnee.columns[2])
sns.boxplot(ax=axes[1, 1], data=donnee, x='Qualité', y=donnee.columns[3])
plt.show()

```

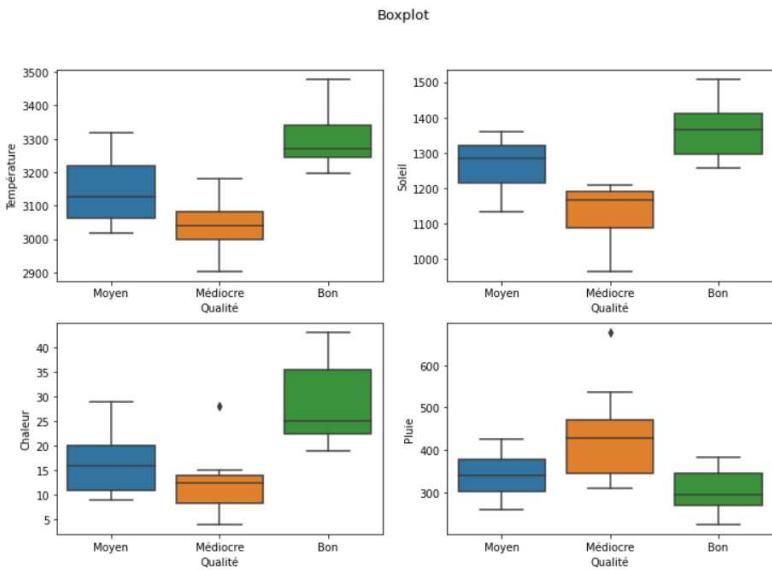


FIGURE 8.1 – Boxplot des variables explicatives

8.1.3 Problématique

L'analyse factorielle discriminante ou analyse discriminante descriptive permet de caractériser de manière multidimensionnelle l'appartenance des individus à des groupes pré définis, ceci à l'aide de plusieurs variables explicatives prises de façon simultanée. En effet, il s'agit de construire un nouveau système de représentation qui permet de mettre en évidence ces groupes. Les objectifs de l'analyse factorielle discriminante sont double :

1. Descriptif : Mettre en évidence les caractéristiques qui permettent de distinguer au mieux les groupes ;
2. Prédicatif : Classer automatiquement un nouvel individu (l'affecter à un groupe) à partir de ses caractéristiques

8.2 Mise en œuvre de l'AFD

Le principe de l'analyse factorielle discriminante consiste à trouver une succession de combinaisons linéaires des variables initiales (variables discriminantes deux à deux orthogonales) qui permet de distinguer au mieux (au sens des barycentres) les groupes [voir [\[Rak\]](#)].

Si on note $X = (X_1, \dots, X_J)$ la matrice des variables explicatives (supposées centrées) et Y la matrice formée des K variables indicatrices des modalités de y , alors l'analyse factorielle discriminante consiste à rechercher une première variable composite Z_1 telle que :

$$Z_1 = a_{11}X_1 + a_{12}X_2 + \dots + a_{1J}X_J = Xa_1 \quad (8.2)$$

ayant un rapport de corrélation $\eta^2(Z_1, y)$ maximum $a_l = (a_{l1}, \dots, a_{lJ})'$ le vecteur-colonne des coefficients. Ensuite, une deuxième variable composite Z_2 (non corrélée à Z_1) définie par :

$$Z_2 = a_{21}X_1 + a_{22}X_2 + \cdots + a_{2J}X_J = Xa_2 \quad (8.3)$$

maximisant $\eta^2(Z_1, y)$, et ainsi de suite.

Les Z_l sont appelées *variables discriminantes* et leur nombre est au plus égal à $K - 1$.

8.2.1 Recherche de variables discriminantes

On adoptera les notations suivantes :

- $x_i^k = (x_{i1}^k, \dots, x_{iJ}^k)'$ le vecteur-colonne des valeurs prises par l'individu i du groupe k ;
- $\bar{x}^k = (\bar{x}_1^k, \dots, \bar{x}_J^k)'$ le vecteur-colonne des moyennes des variables pour le groupe k ;
- $\bar{x} = (\bar{x}_1, \dots, \bar{x}_J)'$ le vecteur-colonne des moyennes des variables sur tout l'échantillon ;
- $T = \sum_{k=1}^K \sum_{i=1}^{I_k} (x_i^k - \bar{x})(x_i^k - \bar{x})'$ la matrice des sommes des carrés et produits totales ($T=Total$) ;
- $B = \sum_{k=1}^K I_k (\bar{x}^k - \bar{x})(\bar{x}^k - \bar{x})'$ la matrice des sommes des carrés et produits interclasses ($B=Between$)
- $W = \sum_{k=1}^K \sum_{i=1}^{I_k} (x_i^k - \bar{x}^k)(x_i^k - \bar{x}^k)'$ la matrice des sommes des carrés et produits intraclassees ($W=Within$)

La formule de décomposition de la variance permet d'écrire :

$$T = B + W \quad (8.4)$$

Sous forme matricielle et en tenant compte du fait que les variables X_j sont centrées, nous avons les écritures suivantes pour T, B et W :

$$T = X'X, \quad B = X'Y(Y'Y)^{-1}Y'X, \quad W = X' \left(I - Y(Y'Y)^{-1}Y' \right) X \quad (8.5)$$

```
# Centrage des données
def centrage(x):
    # x est un vecteur
    return (x-x.mean())
# Matrices X et Y
X = donnee[donnee.columns[:-1]].transform(centrage)
Y = pd.get_dummies(donnee[donnee.columns[-1]], prefix="", prefix_sep="")
# Calcul des matrices T, W et B
T = X.T.dot(X)
w = np.dot(Y.dot(np.linalg.inv(Y.T.dot(Y))), Y.T)
B = pd.DataFrame(np.dot(X.T.dot(w), X), index=donnee.columns[:-1],
                 columns=donnee.columns[:-1])
W = pd.DataFrame(np.dot(X.T.dot(np.identity(I)-w), X), index=donnee.columns[:-1],
```

```

        columns = donnee.columns[:-1])
display(T.round(3))
display(B.round(3))
display(W.round(3))

```

Matrice		Température	Soleil	Chaleur	Pluie
T	Température	657789.529	420250.294	40372.294	-174435.235
	Soleil	420250.294	529101.441	27056.941	-180803.853
	Chaleur	40372.294	27056.941	3310.941	-12119.353
	Pluie	-174435.235	-180803.853	-12119.353	275690.382
B	Température	420067.408	361965.597	26071.355	-187772.538
	Soleil	361965.597	326909.070	21811.721	-175905.823
	Chaleur	26071.355	21811.721	1646.570	-11039.838
	Pluie	-187772.538	-175905.823	-11039.838	97191.170
W	Température	237722.121	58284.697	14300.939	13337.303
	Soleil	58284.697	202192.371	5245.220	-4898.030
	Chaleur	14300.939	5245.220	1664.371	-1079.515
	Pluie	13337.303	-4898.030	-1079.515	178499.212

TABLE 8.7 – Matrices T, B et W

La variable Z_1 définie par l'équation 8.2 prend la valeur $z_{1i}^k = a_1' x_i^k$ pour l'individu i dans le groupe k . $\bar{z}_1^k = a_1' \bar{x}^k$ et $\bar{z}_1 = a_1' \bar{x}$ sont respectivement les moyennes de Z_1 par classe et globale. Ainsi, en réécrivant Z_1 , nous avons l'équation suivante :

$$\sum_{k=1}^K \sum_{i=1}^{I_k} (z_{1i}^k - \bar{z}_1)^2 = \sum_{k=1}^K I_k (\bar{z}_1^k - \bar{z}_1)^2 + \sum_{k=1}^K \sum_{i=1}^{I_k} (z_{1i}^k - \bar{z}_1^k)^2 \quad (8.6)$$

Et en utilisant les définitions des matrices T , B et W , nous avons :

- $\sum_{k=1}^K \sum_{i=1}^{I_k} (z_{1i}^k - \bar{z}_1)^2 = a_1' T a_1$ la somme des carrés totale ;
- $\sum_{k=1}^K I_k (\bar{z}_1^k - \bar{z}_1)^2 = a_1' B a_1$ la somme des carrés interclasses ;
- $\sum_{k=1}^K \sum_{i=1}^{I_k} (z_{1i}^k - \bar{z}_1^k)^2 = a_1' W a_1$ la somme des carrés intraclasses.

L'analyse discriminante descriptive consiste à chercher le vecteur de coefficients « a » qui permet de définir une axe qui maximise le rapport de corrélation avec y

$$\max_{a_1} \eta^2(Z_1 = X a_1, y) \Leftrightarrow \max_{a_1} \frac{a_1^T B a_1}{a_1^T T a_1} \quad (8.7)$$

Ce programme est équivalent au programme suivant :

$$\begin{cases} \max_{a_1} a_1^T B a_1 \\ \text{s.c } a_1^T T a_1 = 1 \end{cases}$$

Le Lagrangien associé s'écrit :

$$L(a_1, \lambda) = a_1^T B a_1 - \lambda (a_1^T T a_1 - 1) \quad (8.8)$$

Ce programme admet une solution a_1 comme vecteur propre de la matrice $T^{-1}B$ associé à la plus grande valeur propre $\lambda_1 = \eta_1^2$. La racine carrée de η_1^2 s'appelle la première corrélation canonique. Ainsi de suite, on obtient les autres variables discriminantes $Z_l = X a_l$ en considérant les vecteurs propres de la matrice $T^{-1}B$ associés aux valeurs propres η_l^2 suivantes, rangées par ordre décroissant. B étant de rang au plus égal à $K - 1$, il y a au plus $K - 1$ valeurs non nulles.

```
# Diagonalisation de la matrice TinuB
H = min(K-1,J)
eigen,vector= np.linalg.eig(np.dot(np.linalg.inv(T),B))
display(eigen[:,H].round(3))
display(vector[:,H].round(3))
```

Remarquons que la matrice $T^{-1}B$ de dimension $J \times J$ n'est pas symétrique.

	axe 1	axe 2
η_l^2	0.766	0.122
a_l	-0.288 -0.228 0.909 0.197	0 -0.042 0.998 0.048

TABLE 8.8 – Valeurs propres et vecteurs

Nous avons

$$\lambda_1 = \eta^2(Z_1, y) = 0.766 \quad (8.9)$$

$$\lambda_2 = \eta^2(Z_2, y) = 0.122 \quad (8.10)$$

Un autre approche détermination des vecteurs propres est de résoudre le programme suivant :

$$\max_{a_1} \frac{a_1^T B a_1}{a_1^T W a_1} \quad (8.11)$$

Ainsi, les vecteurs propres a_l de $T^{-1}B$ associés aux valeurs propres η_l^2 sont aussi les vecteurs propres de la matrice $W^{-1}B$ associées aux valeurs propres :

$$\delta_l = \frac{\eta_l^2}{1 - \eta_1^2}, \quad \forall l = 1, \dots, K - 1 \quad (8.12)$$

```
# Diagonalisation de la matrice WinuB
eigen2,vector2 = np.linalg.eig(np.dot(np.linalg.inv(W),B))
display(eigen2[:,H].round(3))
display(vector2[:,H].round(3))
```

	axe 1	axe 2
δ_l	3.279	0.139
a_l	-0.288	0
	-0.228	-0.042
	0.909	0.998
	0.197	0.048

TABLE 8.9 – Delta et vecteurs

Remarquons que ces deux approches ci-dessus sont décrites dans la plupart des ouvrages en français, mais elles ne correspondent pas aux résultats retournées par certains logiciels (lda() de MASS sous R, Proc Candisc sous SAS,etc.). En effet, nous devons apporter une correction sur la matrice à diagonaliser $T^{-1}B$ ou $W^{-1}B$.

La matrice des covariances interclasses peut être exprimée par le produit d'une matrice C par sa transposée ($B = CC^T$), où C est de dimension $J \times K$ de termes :

$$c_{jk} = \sqrt{\frac{I_k}{I}} (\bar{x}_{kj} - \bar{x}_j) \quad (8.13)$$

Création de la matrice C

```
mean = meanstd['mean']
C = pd.DataFrame(np.zeros((J,K)),index=donnee.columns[:-1],
                  columns =['Mediocre','Bon','Moyen'])
for k in C.columns:
    for j in donnee.columns[:-1]:
        C.loc[j,k]=np.sqrt(propety.loc[k])*(mean.loc[k,j]-meanvar[j])
display(C.round(3))
```

	Mediocre	Bon	Moyen
Temperature	-71.617	84.456	-9.654
Soleil	-71.829	66.158	8.865
Chaleur	-4.004	5.530	-1.347
Pluie	41.522	-31.535	-11.834

TABLE 8.10 – Matrice C

Cette modification nous amène à considérer la matrice $C^T T^{-1} C$ de format $K \times K$ comme matrice à diagonaliser.

Matrice à diagonaliser

```
M = np.dot(np.dot(C.T,np.linalg.inv(T/I)),C)
display(M.round(3))
```

0.408	-0.363	-0.063
-0.363	0.396	-0.018
-0.063	-0.018	0.084

TABLE 8.11 – Matrice $C^T T^{-1} C$

Cette matrice $C^T T^{-1} C$ est symétrique. Ces valeurs propres sont identiques à celles de $T^{-1} B$ et on note b_l les vecteurs propres associés à $C^T T^{-1} C$.

```
# Diagonalisation de la matrice M
eigen3,vector3 = np.linalg.eig(M)
eigen3 = np.array([eigen3[0],eigen3[2]])
vector3 = np.transpose(np.array([vector3[:,0],vector3[:,2]]))
display(eigen3.round(3))
display(vector3.round(3))
```

	axe 1	axe 2
	-0.714	-0.37
b_l	0.698	-0.435
	0.048	0.821

TABLE 8.12 – Vecteurs propres b_l

Pour obtenir les vecteurs propres a_1 des coefficients canoniques, nous appliquons deux transformations successives :

- Première transformation : $\beta_l = (T^{-1} C) b_l$.
- Deuxième transformation : $a_l = \beta_l \sqrt{\frac{I - K}{I \times \lambda_l \times (1 - \lambda_l)}}$

```
# vecteur beta
beta = pd.DataFrame(np.dot(np.dot(np.linalg.inv(T/I),C),vector3),
                     index = donnee.columns[:-1],
                     columns = ['LD'+str(x+1) for x in range(H)])
# coefficients sans constante
etaz = beta.apply(lambda x : np.sqrt((I-K)/I)*x/np.sqrt(eigen3*(1-eigen3)),
                  axis=1)
display(etaz.round(5))
```

	LD1	LD2
Temperature	0.00857	0.00005
Soleil	0.00677	0.00533
Chaleur	-0.02705	-0.12764
Pluie	-0.00587	-0.00617

TABLE 8.13 – Coefficients des variables discriminantes

Pour obtenir les constantes associées, on applique la formule suivante :

$$a_{l0} = - \sum_{j=1}^J a_{lj} \bar{x}_j \quad (8.14)$$

```
# constante du modèle
intercept = -etaz.T.dot(meanvar)
display(intercept.round(5))
```

	LD1	LD2
constante	-32.87628	-2.16528

TABLE 8.14 – Constantes associées aux variables discriminantes

Si nous posons X_1 : Température, X_2 : Soleil, X_3 : Chaleur et X_4 : Pluie, les 2 fonctions discriminantes canoniques sont :

$$Z_1 = 0.00857 \times X_1 + 0.00677 \times X_2 - 0.02705 \times X_3 - 0.00587 \times X_4 - 32.87628 \quad (8.15)$$

$$Z_2 = 0.00005 \times X_1 + 0.00533 \times X_2 - 0.12764 \times X_3 - 0.00617 \times X_4 - 2.16528 \quad (8.16)$$

Nous calculons les coordonnées des individus.

```
# Coordonnées des individus
vsQual = donnee['Qualite']
rowcoord = X.dot(etaz)
display(rowcoord.head())
```

On représente graphiquement les individus en les étiquetant selon leur groupe d'appartenance.

```
# Nuage des individus sur les axes 1 et 2
lda_plot(data=rowcoord, vsqual=vsQual, eigen=eigen3, figsize=(12,8))
```

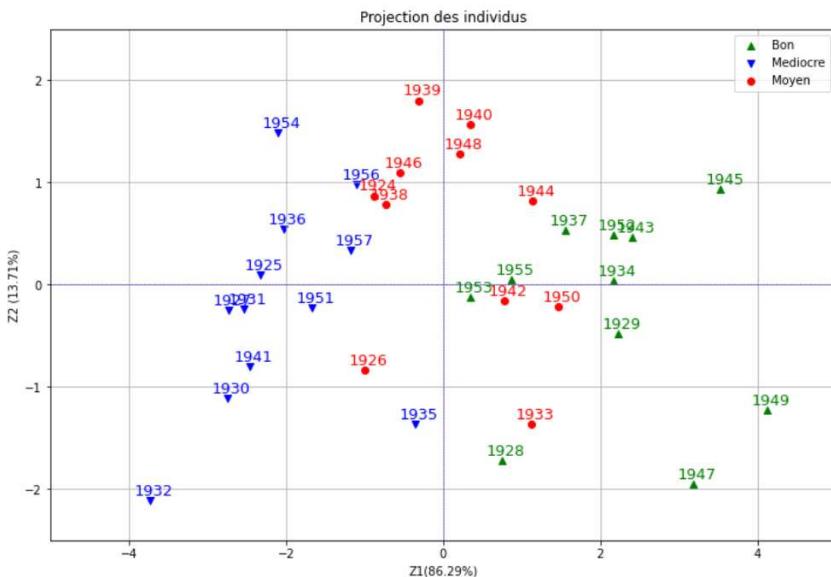


FIGURE 8.2 – Représentation des individus dans le plan factoriel

8.2.2 Les différents types de corrélation en AFD

En analyse factorielle discriminante, on considère trois types de corrélation :

1. La corrélation totale

C'est la corrélation linéaire usuelle entre X_j et Z_l définie par :

$$\text{Corr}(X_j, Z_l) = \frac{\text{Cov}(X_j, Z_l)}{\sigma_{X_j} \sigma_{Z_l}} \quad (8.17)$$

```
# Corrélation totale
corrT = pd.DataFrame(np.zeros((J,H)),index = donnee.columns[:-1],
                      columns = ['LD'+str(x+1) for x in range(H)])
for name in donnee.columns[:-1]:
    for name2 in corrT.columns:
        corrT.loc[name,name2]=np.corrcoef(donnee[name],rowcoord.loc[:,name2])[0,1]
# Affichage
display(corrT.T.round(3))
```

	Temperature	Soleil	Chaleur	Pluie
LD1	0.901	0.897	0.771	-0.663
LD2	-0.375	0.116	-0.590	-0.361

TABLE 8.15 – Corrélation totale entre X_j et Z_l

```
# Scatter plot
df = pd.concat([donnee[donnee.columns[:-1]],rowcoord['LD1']],axis=1)
fig, axes = plt.subplots(2, 2, figsize=(12,7))
fig.suptitle('Scatter plot', fontsize=13)
sns.scatterplot(ax=axes[0, 0],data=df,x='Temperature', y='LD1')
sns.scatterplot(ax=axes[0, 1],data=df,x='Soleil', y='LD1')
sns.scatterplot(ax=axes[1, 0],data=df,x='Chaleur', y='LD1')
sns.scatterplot(ax=axes[1, 1],data=df,x='Pluie', y='LD1')
plt.show()
```

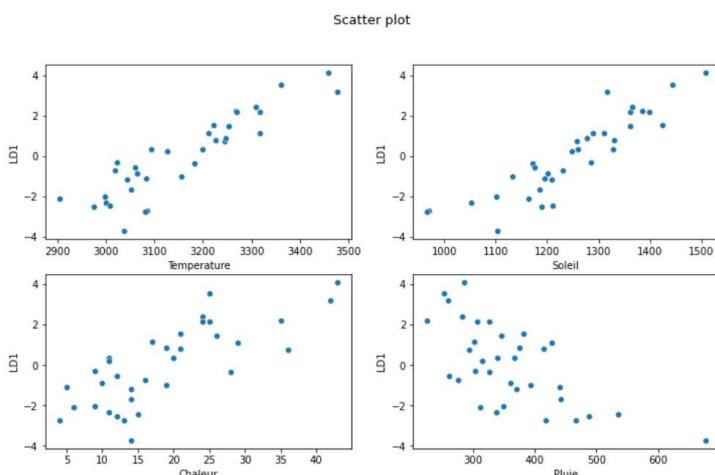


FIGURE 8.3 – Nuage de points entre variables explicatives et le premier axe factoriel

2. Corrélation intraclasses

La corrélation intraclasses correspond à la corrélation partielle entre deux variables X_j et X_p conditionnellement à la variable qualitative y . Sa formule est donnée par :

$$\text{Corr}(X_j, X_p | y) = \frac{\sum_{k=1}^K \sum_{i=1}^{I_k} (x_{ij}^k - \bar{x}_j^k)(x_{ip}^k - \bar{x}_p^k)}{\sqrt{\sum_{k=1}^K \sum_{i=1}^{I_k} (x_{ij}^k - \bar{x}_j^k)^2} \sqrt{\sum_{k=1}^K \sum_{i=1}^{I_k} (x_{ip}^k - \bar{x}_p^k)^2}} \quad (8.18)$$

Cette corrélation est qualifiée de corrélation intraclasses parce qu'elle calculée sur les données centrées sur les moyennes dans classes plutôt que sur les moyennes générales. Afin d'appliquer la formule définie ci-dessus, nous devons au préalable calculer les centres des classes.

```
# Centres de classes dans le plan factoriel
G=mean[donnee.columns[:-1]].dot(etaz)
G = G.apply(lambda x : x+intercept,axis=1)
display(G.T.round(3))
```

Qualité	Bon	Mediocre	Moyen
LD1	2.122	-2.079	0.146
LD2	-0.272	-0.221	0.513

TABLE 8.16 – Centre des classes du plan factoriel

On peut à présent appliquer notre formule.

```
# Corrélation intraclasses
def withincorrcoef(name,lda):
    y = donnee[['Qualité']]
    var = pd.DataFrame(np.zeros((I,1)),columns = [name],
                        index=donnée.index)
    zbar = pd.DataFrame(np.zeros((I,1)),columns = [lda],
                        index=donnée.index)
    mb = pd.concat([y,var],axis=1)
    cb = pd.concat([y,zbar],axis=1)
    for i in range(I):
        if mb.iloc[i,0]=='Moyen':
            mb.iloc[i,1] = mean[name]['Moyen']
            cb.iloc[i,1] = G[lda]['Moyen']
        elif mb.iloc[i,0]=='Bon':
            mb.iloc[i,1] = mean[name]['Bon']
            cb.iloc[i,1] = G[lda]['Bon']
        elif mb.iloc[i,0]=='Mediocre':
            mb.iloc[i,1] = mean[name]['Mediocre']
            cb.iloc[i,1] = G[lda]['Mediocre']
    X = donnee[name] - mb[name]
    Y = rowcoord.iloc[:,0]- cb[lda]
    corr = np.corrcoef(X,Y)[0,1]
```

```

    return corr

# Application
corrW = pd.DataFrame(np.zeros((J,H)),index = donnee.columns[:-1],
                     columns = ['LD'+str(x+1) for x in range(H)])
for name in donnee.columns[:-1]:
    for name2 in corrW.columns:
        corrW.loc[name,name2]=withinccorcoef(name,name2)
# Affichage
display(corrW.T.round(3))

```

	Temperature	Soleil	Chaleur	Pluie
LD1	0.724	0.701	0.525	-0.398
LD2	0.345	0.334	0.250	-0.189

TABLE 8.17 – Corrélation intraclasses entre X_j et Z_l

3. Corrélation interclasses

La corrélation interclasses est la corrélation calculée sur les centres de gravité des classes pondérées par les effectifs :

$$\text{Corr}_{\text{interclasses}}(X_j, X_p) = \frac{\sum_{k=1}^K I_k(\bar{x}_j^k - \bar{x}_j)(\bar{x}_p^k - \bar{x}_p)}{\sqrt{\sum_{k=1}^K I_k(\bar{x}_j^k - \bar{x}_j)^2} \sqrt{\sum_{k=1}^K I_k(\bar{x}_p^k - \bar{x}_p)^2}} \quad (8.19)$$

```

# Corrélation interclasses
def betweenccorcoef(name,lda,weights):
    def m(x, w):
        return np.average(x,weights=w)

    def cov(x, y, w):
        return np.sum(w * (x - m(x, w)) * (y - m(y, w))) / np.sum(w)

    def corr(x, y, w):
        return cov(x, y, w) / np.sqrt(cov(x, x, w) * cov(y, y, w))

    return corr(mean[name], G[lda],weights)

# Application
corrB = pd.DataFrame(np.zeros((J,H)),index = donnee.columns[:-1],
                     columns = ['LD'+str(x+1) for x in range(H)])
for name in donnee.columns[:-1]:
    for name2 in corrB.columns:
        corrB.loc[name,name2]=betweenccorcoef(name,name2,prop)
# Affichage
display(corrB.T.round(3))

```

	Temperature	Soleil	Chaleur	Pluie
LD1	0.987	0.999	0.957	-0.977
LD2	-0.163	0.051	-0.291	-0.211

TABLE 8.18 – Corrélation interclasses entre X_j et Z_l

8.3 Évaluation globale du modèle

8.3.1 Distance entre centre classes

Sa formule est :

$$d^2(y_1, y_2) = \sum_{h=1}^H (\bar{Z}_{1h} - \bar{Z}_{2h})^2 = (\bar{Z}_1 - \bar{Z}_2)(\bar{Z}_1 - \bar{Z}_2)^T \quad (8.20)$$

```
# Distance entre barycentres
disto = pd.DataFrame(np.zeros((K,K)),index=G.index,
                      columns= G.index)
for k in range(K-1):
    for l in range(k+1,K):
        disto.iloc[k,l] = np.sum((G.iloc[k,:]-G.iloc[l,:])**2)
        disto.iloc[l,k] = disto.iloc[k,l]
# Affichage
display(disto.round(3))
```

Qualité	Bon	Médiocre	Moyen
Bon	0.000	17.653	4.519
Médiocre	17.653	0.000	5.492
Moyen	4.519	5.492	0.000

TABLE 8.19 – Distances entre barycentres

```
# Représentation dans le plan factoriel
disto_plot(data=G,vsqual = G.index,eigen=eigen3,dist=disto,figsize=(12,8))
```

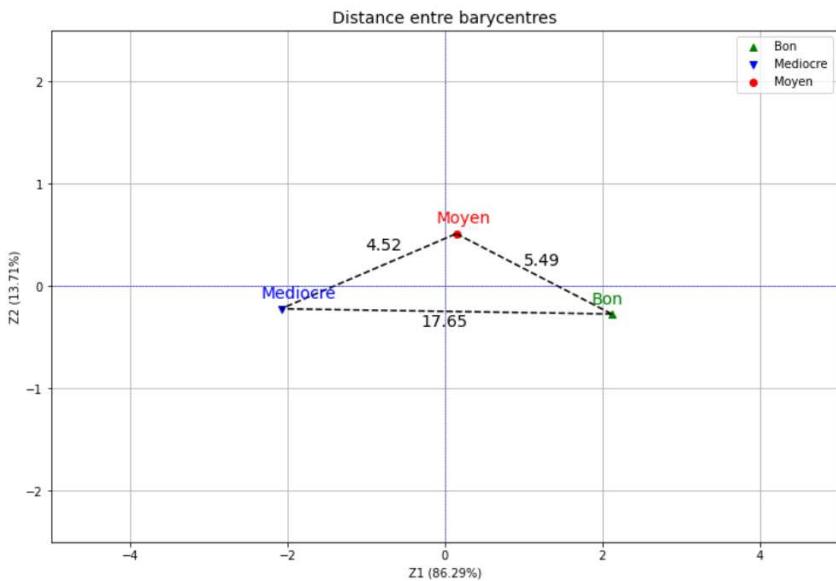


FIGURE 8.4 – Distances (au carré) entre barycentres

8.3.2 Pouvoir discriminant des facteurs

Nous pouvons calculer la contribution d'un facteur dans la discrimination des classes sous la forme de « proportion » d'explication :

$$\frac{\delta_l}{\sum_{h=1}^H \delta_h} \quad (8.21)$$

```
# Pouvoir discriminant des facteurs
lambdah = pd.DataFrame(eigen3,columns = ['lambda'],
                       index= ['LD'+str(x+1) for x in range(H)]).T
# deltah
deltah = lambdah.apply(lambda x : x/(1-x),axis=0)
deltah.index = ['delta']
# Contribution
proph = deltah.apply(lambda x : x/np.sum(x),axis=1)
proph.index = ['proportion of trace']
df = pd.concat([lambdah,deltah,proph],axis=0)
display(df.T.round(4))
```

	lambda	delta	proportion of trace
LD1	0.7663	3.2789	0.9595
LD2	0.1217	0.1386	0.0405

TABLE 8.20 – Pouvoir discriminant des facteurs

8.3.3 Calcul du Lambda de Wilks

Le lambda de Wilks est l'indicateur privilégié pour l'évaluation statistique du modèle. Il indique dans quelle mesure les centres de classes sont distincts les uns des autres dans l'espace de représentation. Il varie entre 0 et 1 :

- Vers 0, le modèle sera bon parce les nuages sont bien distincts
- Vers 1, les nuages sont confondus, il est difficile de discerner les individus appartenant à des classes différentes.

Les matrices de variance-covariance intraclasses et totales sont utiles pour le calcul du **lambda** (Λ) de **Wilks**. Il est égal au rapport :

$$\Lambda = \frac{\det(W)}{\det(T)} \quad (8.22)$$

Cette approche est basée sur le ratio des dispersions intraclasses et totales.

```
# Lambda de Wilks
LW = np.linalg.det(W)/np.linalg.det(T)
print('Lambda de Wilks : %.4f'%(LW))

Lambda de Wilks : 0.2053
```

Une seconde approche consiste à utiliser à la formule suivante :

$$\Lambda = \prod_{l=1}^H (1 - \eta_l^2) \quad (8.23)$$

```
# Autre approche - lambda Wilks
LW2 = np.prod(1-eigen3)
print('Lambda de Wilks : %.4f'%(LW2))

Lambda de Wilks : 0.2053
```

Dans notre cas, $\Lambda = 0.2053$, c'est plutôt bon signe. Cependant, la table des valeurs critiques de la loi de Wilks étant rarement disponible dans les logiciels, il faut se tourner vers les transformation de Bartlett ou de Rao qui suivent respectivement une loi du KHI-2 et de Fisher.

8.3.4 Transformation de Bartlett

Elle est moins précise que celle Rao sur les petits effectifs. Lorsque I est grand en revanche, elle suffit amplement. Sous HO, la statistique du test s'écrit :

$$B = - \left(I - 1 - \frac{J + K}{2} \right) \times \log(\Lambda) \quad (8.24)$$

suit approximativement une loi du χ^2 à $J \times (K - 1)$ degrés de liberté.

```

# Statistique de test de Bartlett
B = -(I - 1 - ((J+K)/2))*np.log(LW)
print("Statistique B de Bartlett : %.4f" % (B))

# degré de liberté
ddl = J * (K-1)
print("Degré de liberté : %.d" % (ddl))

# p-value
import scipy as sp
print("P-value du test : %.4f" % (1-sp.stats.chi2.cdf(B,ddl)))

```

Informations	valeur
Statistique B de Bartlett	46.7122
Degré de liberté	8
P-value du test	0.0000

TABLE 8.21 – Transformation de Bartlett

8.3.5 Transformation de RAO

Elle est à privilégier sur les petits effectifs. Mais la formule de la statistique de test est plus difficile à mettre en oeuvre. Les **formules** sont accessibles dans la documentation en ligne de SAS. On peut dès lors rendre compte de la significativité globale du modèle.

```

# DDL numérateur
ddlNum = J * (K-1)
print("DDL numérateur : %.d" % (ddlNum))

# valeur intermédiaire pour calcul du ddl dénominateur
temp = J**2 + (K-1)**2 - 5
temp = np.where(temp > 0,np.sqrt(((J**2) * ((K-1)**2) - 4)/temp),1)

# ddl dénominateur
ddlDenom = (2*I - J - K - 2)/2 * temp - (ddlNum - 2)/2
print("DDL dénominateur : %.d" % (ddlDenom))

#stat de test
FRao = LW***(1/temp)
FRao = ((1-FRao)/FRao)*(ddlDenom/ddlNum)
print("Statistique F de Rao : %.4f" % (FRao))

#p-value
print("P-value du test : %.4f" % (1-sp.stats.f.cdf(FRao,
                                                    ddlNum,ddlDenom)))

```

Informations	valeur
DDL numérateur	8
DDL dénominateur	56
Statistique F de Rao	8.4505
P-value du test	0.0000

TABLE 8.22 – Transformation de RAO

Le Lambda de Wilks, la transformation de Bartlett et celle de Rao correspondent en réalité à un test d'analyse de variance multivariée : On parle de **MANOVA**.

Stat	Value	p-value
Wilks' Lambda	0.2053	
Bartlett - B(8)	46.7122	0.000
Rao - F(8, 56)	8.4505	0.000

TABLE 8.23 – MANOVA

Les statistiques de test de Bartlett et de Rao aboutissent à la même conclusion : à 5%, on rejette l'hypothèse selon laquelle les centres de classes sont confondus. L'analyse discriminante est viable dans ce contexte.

8.3.6 Test sur ensemble de facteurs

On cherche à tester la nullité des q derniers rapports de corrélation théoriques. Pour cela, on utilise la statistique suivante :

$$\Lambda_q = \prod_{l=K-q}^H (1 - \eta_l^2) \quad (8.25)$$

L'hypothèse testée est rejetée pour des valeurs de Λ_q trop petites. Bartlett et Rao ont proposé des transformations décroissantes de Λ_q permettant de calculer approximativement le niveau de signification du Λ_q .

1. Transformation de Bartlett

Sous l'hypothèse testée, la statistique :

$$\chi^2 = - \left\{ I - 1 - \frac{J + K}{2} \right\} \log(\Lambda_q) \quad (8.26)$$

suit approximativement une loi du Khi-deux à $q(J - K + q + 1)$ degrés de liberté.

2. Transformation de Rao

Rao a proposé la statistique

$$F = \frac{1 - \Lambda_q^{1/t}}{\Lambda^{1/t}} \times \frac{rt - 2u}{q(J - K + q + 1)} \quad (8.27)$$

où :

- $r = I - 1 - \frac{J + K}{2}$
- $u = \frac{q(I - K + q + 1) - 2}{4}$
- $t = \begin{cases} \sqrt{\frac{q^2(I - K + q + 1)^2 - 4}{(I - K + q + 1)^2 + q^2 - 5}} & \text{si } (I - K + q + 1)^2 + q^2 - 5 > 0 \\ 1 & \text{sinon} \end{cases}$

Sous l'hypothèse testée, F suit approximativement une loi de Fisher-Snedecor à $q(I - K + q + 1)$ et $rt - 2u$ degrés de liberté. La loi est exacte si $\min(I - K + q + 1, q) \leq 2$.

L'approximation de Rao est plus précise que celle de Bartlett.

```
# test du dernier facteur
q = 1
# stat de test
LQ = 1 - eigen3[1]
print('Stat. de test : %.3f' % (LQ.round(3)))

# Transformation de Barlett
BQ = -(I - 1 - ((J+K)/2))*np.log(LQ)
print('Bartlett transformation : %.3f' %(BQ.round(3)))

# Degrés de liberté
ddlbq = q*(J-K+q+1)
print('Degrés de liberté : %i' %(ddlbq))

# pvalue associé
print("P-value du test : %.4f" % (1-sp.stats.chi2.cdf(BQ,ddlbq)))
```

Informations	Valeur
Stat. de test	0.878
Bartlett transformation	3.828
Degrés de liberté	3
P-value du test	0.2806

TABLE 8.24 – Test sur le second facteur

8.3.7 Taux de bon classement et prédition

Les indicateurs calculés précédemment nous ont permis de constater que seule la variable discriminante Z_1 est utilisable. Elle est nettement plus performante que chacun des variables prises isolément pour séparer les groupes. Le calcul des corrélations totales, intraclasses et interclasses a permis de voir que Z_1 est bien corrélée à toutes les variables du problème. Elle oppose les variables Température, Soleil, Chaleur à la variable Pluie.

On peut interpréter Z_1 comme un score prédictif de la qualité du vin construit à partir des variables météo. La règle de décision consistant à affecter une observation au groupe 1 (Bon) si $Z_1 > 1.5$, au groupe 2 (Moyen) si $-1 \leq Z_1 \leq 1.5$ et au groupe 3 (Médiocre) si $Z_1 < -1$.

```

# Classes prédictes
fitted = list()
for i in range(I):
    if rowcoord.iloc[i,0]<-1:
        val = 'Mediocre'
        fitted.append(val)
    elif rowcoord.iloc[i,0]>1.5:
        val = 'Bon'
        fitted.append(val)
    else:
        val = 'Moyen'
        fitted.append(val)
# Concaténation
pred = pd.DataFrame(np.array(fitted),columns=['Prediction'],
                     index = donnee.index)
rowinfos = pd.concat([rowcoord,donnee[['Qualite']],pred],axis=1)
display(rowinfos.sort_values(by='LD1',ascending=False))

```

Nous effectuons un croisement entre les classes initiales et les classes prédictes.

```

# Matrice de croisement
confmat = pd.crosstab(rowinfos['Qualite'],rowinfos['Prediction'])
confmat.columns.name='Prediction'
display(confmat)

```

Qualite	Prédiction			
	Bon	Mediocre	Moyen	Total
Bon	8	0	3	11
Mediocre	0	11	1	12
Moyen	0	0	11	11
Total	8	11	15	34

TABLE 8.25 – Croisement entre classes initiales et classes prédictes

On voit que notre règle de décision conduit à 4 mal classés, indiqués en bleu dans le tableau 8.26.

Annee	LD1	LD2	Qualite	Prediction
1949	4.119175	-1.223049	Bon	Bon
1945	3.535288	0.932601	Bon	Bon
1947	3.182114	-1.945670	Bon	Bon
1943	2.409876	0.463039	Bon	Bon
1929	2.230889	-0.484319	Bon	Bon
1934	2.174732	0.042822	Bon	Bon
1952	2.167128	0.488959	Bon	Bon
1937	1.552112	0.533567	Bon	Bon
1950	1.466797	-0.216641	Moyen	Moyen
1944	1.138016	0.818206	Moyen	Moyen
1933	1.130406	-1.368430	Moyen	Moyen
1955	0.874239	0.049858	Bon	Moyen
1942	0.785837	-0.159365	Moyen	Moyen
1928	0.743596	-1.721167	Bon	Moyen
1953	0.352444	-0.126575	Bon	Moyen
1940	0.343475	1.563278	Moyen	Moyen
1948	0.209681	1.281275	Moyen	Moyen
1939	-0.306058	1.803016	Moyen	Moyen
1935	-0.356657	-1.364226	Mediocre	Moyen
1946	-0.551913	1.095020	Moyen	Moyen
1938	-0.729461	0.789200	Moyen	Moyen
1924	-0.882552	0.871537	Moyen	Moyen
1926	-0.994856	-0.832957	Moyen	Moyen
1956	-1.094421	0.984656	Mediocre	Mediocre
1957	-1.181896	0.335581	Mediocre	Mediocre
1951	-1.676155	-0.225816	Mediocre	Mediocre
1936	-2.021081	0.542615	Mediocre	Mediocre
1954	-2.102251	1.486226	Mediocre	Mediocre
1925	-2.325456	0.094220	Mediocre	Mediocre
1941	-2.454483	-0.801771	Mediocre	Mediocre
1931	-2.533831	-0.236018	Mediocre	Mediocre
1927	-2.726862	-0.247244	Mediocre	Mediocre
1930	-2.746995	-1.108790	Mediocre	Mediocre
1932	-3.730876	-2.113641	Mediocre	Mediocre

TABLE 8.26 – Les variables discriminantes

On mesure également la performance du modèle grâce au taux de bon classement.

```
# Accuracy
accuracy = np.sum(np.diag(confmat))/I
print(f'Taux de bon classement : %.4f'%(accuracy))
```

Taux de bon classement : 0.8824

On obtient un score 88.24%.

Considérons l'année 1958. Les données (hypothétiques) de cette année sont :

Année	Température	Soleil	Chaleur	Pluie
1958	3000	1 100	20	300

TABLE 8.27 – Données de 1958

Avec la règle de décision définie, on prévoit la qualité du vin de 1958. Calculons Z_1 .

Prévision de l'année 1958

```
Xtest = pd.DataFrame(np.array([3000,1100,20,300]).reshape(1,4),
                     columns = donnee.columns[:-1],index=['1958'])
yPred = etaz.T.dot(Xtest.T).apply(lambda x:x+intercept, axis=0).T
display(yPred.round(4))
```

	LD1	LD2
1958	-2.0277	-0.5694

TABLE 8.28 – Prévision de l'année 1958

Nuage des individus avec prévision

```
lda_rowsup_plot(data=rowcoord,data2=yPred,vsqual =vsQual,eigen=eigen3,  
                 figsize=(12,8))
```

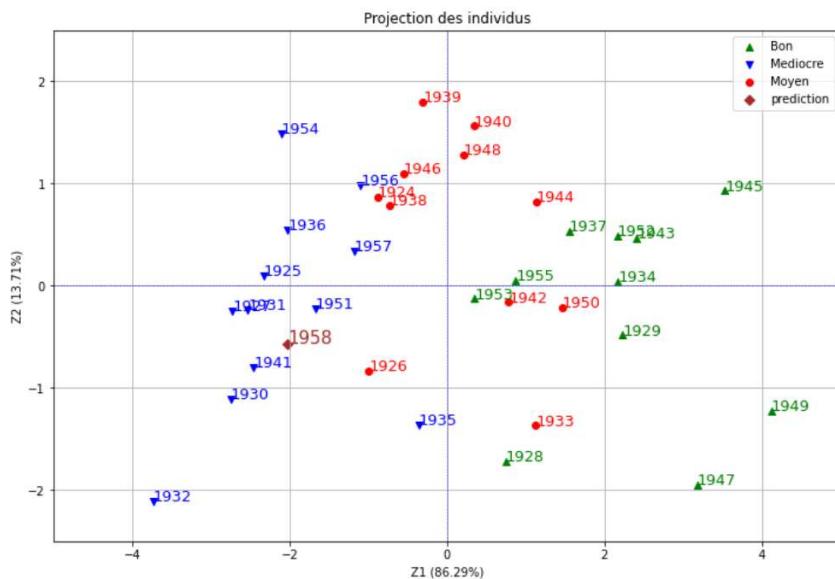


FIGURE 8.5 – Représentation dans le plan factoriel

8.3.8 Évaluation des contributions des variables

Mesurer l'impact des variables est crucial pour l'interprétation du mécanisme d'affectation. Pour l'analyse discriminante, il est possible de produire une mesure d'importance des variables

basées sur leurs contributions à la discrimination. Concrètement, il s'agit d'opposer les lambda de Wilks avec ou sans la variable à évaluer. Sous Python, cela s'effectue à partir des matrices de covariances. L'idée est d'évaluer la contribution de chaque variable dans l'écartement entre les barycentres conditionnels en opposant les lambda de Wilks, avec et sans la variable incriminée. Ces opérations s'effectuent en manipulant judicieusement les matrices de covariance intra-classes et totales sans avoir à réitérer entièrement les calculs de l'analyse discriminante. Ceci grâce à une simple boucle.

```
# degrés de liberté - numérateur
ddlsuppnum = K - 1
# degrés de liberté dénominateur
ddlsuppden = I - K - J + 1
# FTest - préparation
FTest = np.zeros(J)
# p-value pour FTest
pvalueftest = np.zeros(J)
# Tester chaque variable
for j in range(J):
    # Matrices intermédiaires numérateur
    tempnum = W.copy().values
    # Supprimer la référence de la variable à traiter
    tempnum = np.delete(tempnum, j, axis = 0)
    tempnum = np.delete(tempnum, j, axis = 1)
    # Même chose pour le numérateur
    tempden = T.values
    tempden = np.delete(tempden, j, axis = 0)
    tempden = np.delete(tempden, j, axis = 1)
    # Lambda sans la variable
    LWVar = np.linalg.det(tempnum)/np.linalg.det(tempden)
    # FValue
    FValue = ddlsuppden/ddlsuppnum * (LWVar/LW-1)
    # Récupération des valeurs
    FTest[j] = FValue
    pvalueftest[j] = 1 - sp.stats.f.cdf(FValue, ddlsuppnum, ddlsuppden)

# Affichage
temp = np.concatenate((FTest.reshape(J,1), pvalueftest.reshape(J,1)), axis = 1)
print(pd.DataFrame(temp, columns = ["F", "p-value"], index = donnee.columns[:-1]))
```

Régresseurs	F	p-value
Temperature	3.529197	0.042966
Soleil	4.164681	0.026098
Chaleur	0.943617	0.401247
Pluie	2.934485	0.069660

TABLE 8.29 – Statistiques de test

Ces deux colonnes servent à tester la contribution des variables. La statistique suit une loi de Fisher. Les probabilités critiques (p-value) permettent de statuer. Dans notre exemple, seule ACAL ne semble pas contribuer significativement à 5%.

Deuxième partie

Régression linéaire

CHAPITRE 9

LE MODÈLE LINÉAIRE DE RÉGRESSION SIMPLE

Sommaire

9.1 Le modèle et ses hypothèses	221
9.2 Équation d'analyse de la variance	228
9.3 Prévision, tests de significativité des paramètres et intervalles de confiance	230

On parle de modèle linéaire de régression simple, lorsqu'une seule variable est utilisée pour expliquer une autre variable. La variable expliquée est appelée variable endogène et la variable explicative est appelée variable exogène. Un tel modèle peut être spécifié en séries temporelles ou en coupe instantanée. En séries temporelles, les variables sont observées en intervalle de temps régulier alors qu'en coupe instantanée, les données sont observées au même instant et concernent les valeurs prises par la variable pour un groupe d'individu spécifique.

9.1 Le modèle et ses hypothèses

9.1.1 Présentation du modèle

Spécifié en séries temporelles, le modèle linéaire de régression simple s'écrit :

$$y_t = \beta_0 + \beta_1 x_t + \varepsilon_t, \quad \forall t = 1, \dots, n \quad (9.1)$$

Dans ce modèle, y_t est la variable à expliquer au temps t (variable endogène) ; x_t est la variable explicative au temps t (variable exogène) ; β_0 et β_1 sont les coefficients ou paramètres du modèle ; n est le nombre de période (nombre d'observations) et ε_t est le terme d'erreur (l'erreur de spécification), c'est-à-dire la différence entre le modèle vrai et le modèle spécifié.

En effet, le modèle ci-dessus stipule que la variable endogène y est expliquée par la variable exogène x , mais il existe peut être d'autres variables que l'on ignore qui contribue aussi à l'explication de la variable y . L'erreur de spécification (ε) qui est un terme aléatoire à pour rôle de prendre en compte toutes ses autres variables explicatives non expliquées.

L'une des étapes de la résolution du modèle consiste à estimer les paramètres β_0 et β_1 . Certaines hypothèses sont émises sur le terme d'erreur aléatoire (ε) en vue d'obtenir des estimateurs sans biais et convergents des paramètres du modèle.

9.1.2 Les hypothèses du modèle

H1) Les valeurs de la variable explicative sont observées sans erreur

Cette variable est donc non aléatoire (alors que la variable endogène y est aléatoire en raison de la présence du terme aléatoire (ε)).

H2) L'espérance mathématique de l'erreur est nulle : $\mathbb{E}(\varepsilon_t) = 0, \quad \forall t = 1, \dots, n$

Cela signifie qu'en moyenne le modèle est bien spécifié puisque l'erreur moyenne est nulle.

H3) La variance de l'erreur est une constante : $V(\varepsilon_t) = \sigma_\varepsilon^2, \quad \forall t = 1, \dots, n$

C'est l'hypothèse de l'homoscédasticité. Lorsqu'elle n'est pas vérifiée, on dit que le modèle est hétéroscédastique.

H4) Les termes d'erreur sont indépendants les uns des autres dans le temps : $\text{Cov}(\varepsilon_t, \varepsilon_s) = 0, \forall t \neq s$

C'est l'hypothèse de non corrélation ou d'indépendance des erreurs. Une erreur à l'instant t n'a pas d'impact sur les autres erreurs. Lorsque cette hypothèse est violée, on dit que les erreurs sont corrélées ou d'autocorrélation des erreurs.

H5) L'erreur est indépendante de la variable explicative : $\text{Cov}(x_t, \varepsilon_t) = 0 \forall t = 1, \dots, n$

H6) Lorsque le nombre d'observation n devient grand, les premiers moments empiriques de x (espérance mathématique et variance) sont finis

Remarque :

Le modèle linéaire de régression simple spécifié en coupe instantanée s'écrit :

$$y_i = \beta_0 + \beta_1 x_i, \quad \forall i = 1, \dots, n \quad (9.2)$$

y_i est la variable à expliquer de l'individu i et x_i la variable explicative de l'individu i .

9.1.3 Estimation des paramètres par moindres carrés ordinaires (MCO)

La méthode des moindres carrés ordinaires consiste à minimiser la somme des carrés des erreurs. Soit le programme

$$\min_{\beta_0, \beta_1} \sum_{t=1}^n (y_t - \beta_0 - \beta_1 x_t)^2 \quad (9.3)$$

Ce programme admet son minimum lorsque les dérivées partielles premières par rapport à β_0 et β_1 sont nulles. On obtient les estimateurs suivants :

$$\begin{cases} \hat{\beta}_1 = \frac{\text{Cov}(x_t, y_t)}{V(x_t)} \\ \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \end{cases}$$

Remarque :

- Ces estimateurs sont à la fois sans biais et convergents. De plus, ce sont des fonctions linéaires de y_t .

- Les estimateurs $\hat{\beta}_0$ et $\hat{\beta}_1$ sont des variables aléatoires réelles de matrice de covariance :

$$\hat{\sigma}_{\hat{\beta}}^2 = \hat{\sigma}_{\varepsilon}^2 \begin{pmatrix} \frac{1}{n} + \frac{\bar{x}^2}{\sum_{t=1}^n (x_t - \bar{x})^2} & -\frac{\bar{x}}{\sum_{t=1}^n (x_t - \bar{x})^2} \\ -\frac{\bar{x}}{\sum_{t=1}^n (x_t - \bar{x})^2} & -\frac{1}{\sum_{t=1}^n (x_t - \bar{x})^2} \end{pmatrix}$$

où $\hat{\sigma}_{\varepsilon}^2$ est l'estimateur sans biais et convergent de σ_{ε}^2 avec :

$$\hat{\sigma}_{\varepsilon}^2 = \frac{1}{n-2} \sum_{t=1}^n (y_t - \hat{\beta}_0 - \hat{\beta}_1 x_t)^2 \quad (9.4)$$

Application : La concentration en ozone

Nous allons traiter les 50 données journalières de concentration en ozone [voir [\[CHMLR19\]](#)]. La variable à expliquer est la concentration en ozone notée O3 et la variable explicative est la température notée T12.

```
# Chargement des données
import pandas as pd
```

```
donnee = pd.read_csv('ozone.txt', sep=';', index_col=0)
```

On affiche nos données les 10 premières observations de notre jeu de données.

```
# 10 premières observations
print(donnee[['O3', 'T12']].head(10))
```

Date	O3	T12
1996-04-22	63.6	13.4
1996-04-29	89.6	15.0
1996-05-06	79.0	7.9
1996-05-14	81.2	13.1
1996-05-21	88.0	14.1
1996-05-28	68.4	16.7
1996-06-05	139.0	26.8
1996-06-12	78.2	18.4
1996-06-19	113.8	27.2
1996-06-27	41.8	20.6

TABLE 9.1 – 10 données de température à 12 h et teneur en ozone.

Nous allons chercher à expliquer le maximum de O3 de la journée par la température à 12 h. D'un point de vue pratique, le but de cette régression est double :

1. Ajuster un modèle pour expliquer la concentration en O3 en fonction de T12 ;
2. Prédire les valeurs de concentration en O3 pour de nouvelles valeurs de T12.

Avant toute analyse, il est intéressant de représenter les données. Chaque point du graphique 9.1 représente, pour un jour donné, une mesure de la température à T12 h et le pic d'ozone de la journée.

```
# Nuage des points
import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))
plt.scatter(donnee['T12'], donnee['O3'], color="blue")
for i in range(len(donnee)):
    plt.annotate(i+1, (donnee['T12'].values[i], donnee['O3'].values[i]),
                 fontsize=13)
plt.grid()
plt.xlabel('T12')
plt.ylabel('O3')
plt.title("Nuage de points : O3 en fonction de T12")
plt.show()
```

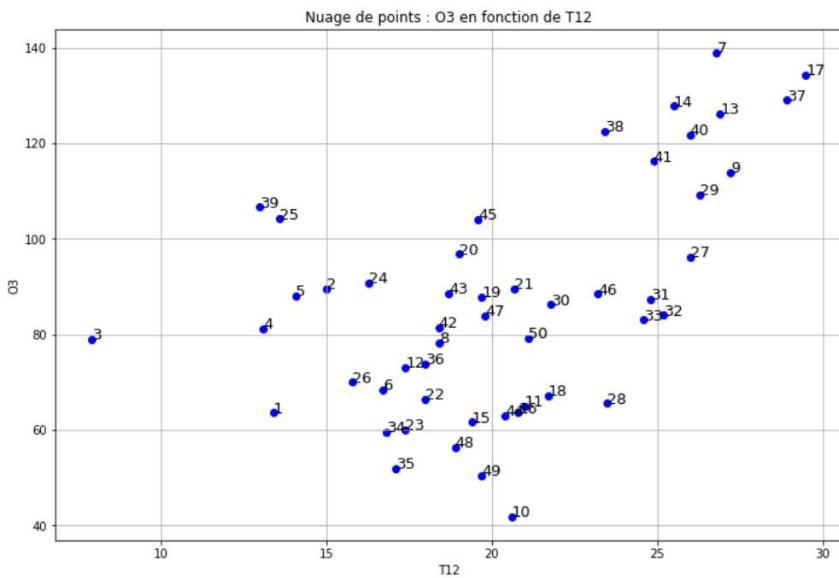


FIGURE 9.1 – 50 données journalières de température et O3

Ce graphique permet de vérifier visuellement si une régression linéaire est pertinente. Autrement dit il suffit de regarder si le nuage de point s'étire le long d'une droite. Bien qu'ici il semble que le nuage s'étire sur une première droite jusqu'à 22 ou 23°C puis selon une autre droite pour

les hautes valeurs de températures, nous pouvons tenter une régression linéaire simple. À l'aide de la librairie `statsmodels`, on ajuste le modèle de régression :

$$O3_t = \beta_0 + \beta_1 T12_t + \varepsilon_t \quad (9.5)$$

```
# Estimation des paramètres
from statsmodels.api import OLS

model = OLS.from_formula(formula = 'O3~T12', data=donnee).fit()
# résultat
print(model.summary2())
```

Ici, nous appelons la fonction `summary2` afin d'avoir l'estimation de la variance de l'erreur ε .

Results: Ordinary least squares						
		====				
Model:	OLS	Adj. R-squared:			0.264	
Dependent Variable:	O3	AIC:			445.9136	
Date:	2022-02-08 20:27	BIC:			449.7376	
No. Observations:	50	Log-Likelihood:			-220.96	
Df Model:	1	F-statistic:			18.58	
Df Residuals:	48	Prob (F-statistic):			8.04e-05	
R-squared:	0.279	Scale:			420.40	
		Coef.	Std.Err.	t	P> t	[0.025 0.975]
		-----	-----	-----	-----	-----
Intercept	31.4150	13.0584	2.4057	0.0200	5.1592	57.6707
T12	2.7010	0.6266	4.3107	0.0001	1.4412	3.9609
		-----	-----	-----	-----	-----
Omnibus:	2.252	Durbin-Watson:			1.461	
Prob(Omnibus):	0.324	Jarque-Bera (JB):			1.348	
Skew:	0.026	Prob(JB):			0.510	
Kurtosis:	2.197	Condition No.:			94	
		-----	-----	-----	-----	-----

On a le modèle ajusté suivant :

$$\hat{O}3 = 31.41 + 2.77 T12 \quad (9.6)$$

L'estimation de la variance $\hat{\sigma}^2$ vaut

```
# variance de l'erreur
scale = model.scale
print("Variance de l'erreur : %.4f"%(scale))

Variance de l'erreur : 420.4041
```

Afin d'examiner la qualité du modèle et des observations, nous traçons la droite ajustée et les observations.

```

# Graphique
beta0 = round(model.params[0],2); beta1 = round(model.params[1],2)
plt.figure(figsize=(12,8))
plt.scatter(donnee['T12'], donnee['O3'], color="blue")
plt.plot(donnee['T12'],model.fittedvalues,color='green',
          label = f'valeur ajustée : y={beta0} + {beta1}x')
for i in range(len(donnee)):
    plt.annotate(i+1,(donnee['T12'].values[i], donnee['O3'].values[i]),
                 fontsize=13)
plt.grid()
plt.legend()
plt.xlabel('T12')
plt.ylabel('O3')
plt.title("Nuage de points : O3 en fonction de T12")
plt.show()

```

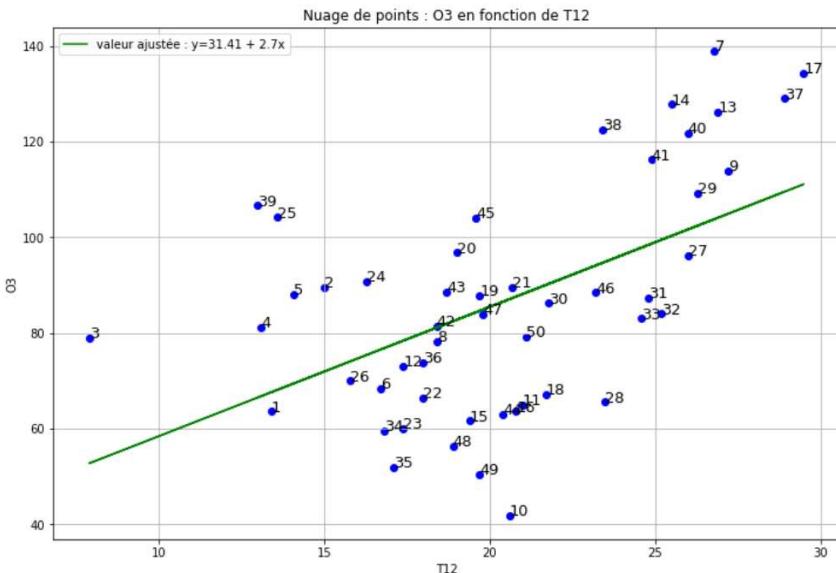


FIGURE 9.2 – 50 données journalières de température et O3 et l'ajustement linéaire

On retourne les variances et covariances des estimateurs :

```

# Matrice des variances-covariances des estimateurs
covparams = model.cov_params()
print(covparams)

```

	Intercept	T12
Intercept	170.522700	-7.978082
T12	-7.978082	0.392622

9.1.4 Estimation par maximum de vraisemblance (MV)

A côté de la méthode des moindres carrés ordinaires (MCO), la méthode du maximum de vraisemblance (*Maximum Likelihood Method*, en anglais) [voir [JJ99]] permet également d'estimer les paramètres d'un modèle de régression, sous l'hypothèse que la vraie loi de distribution desdits paramètres est connue. Si le principe pour les MCO est de trouver le paramètre qui minimise la somme des carrés des erreurs, la méthode du maximum de vraisemblance cherche par contre à trouver le paramètre à même (ayant une forte probabilité) de reproduire les vraies valeurs de l'échantillon.

Nous supposons désormais que les erreurs suivent une loi normale $\varepsilon_t \sim \mathcal{N}(0, \sigma_\varepsilon^2)$. Ce qui implique que $y_t \sim \mathcal{N}(\beta_0 + \beta_1 x_t, \sigma_\varepsilon^2)$. La vraisemblance est la densité de l'échantillon vue comme fonction des paramètres. Grâce à l'indépendance des erreurs, les observations sont indépendantes et la vraisemblance s'écrit :

$$L(y_1, \dots, y_n; \beta_0, \beta_1, \sigma_\varepsilon^2) = \prod_{t=1}^n f(y_t) = \left(\frac{1}{2\pi\sigma_\varepsilon^2} \right)^{n/2} \exp \left(-\frac{1}{2\sigma_\varepsilon^2} \sum_{t=1}^n (y_t - \beta_0 - \beta_1 x_t)^2 \right) \quad (9.7)$$

Lorsqu'on effectue une transformation logarithmique de la fonction de vraisemblance ci-dessus, l'on obtient la fonction dite log-vraisemblance qui servira de base à l'estimation des paramètres $\hat{\beta}_0, \hat{\beta}_1$ et $\hat{\sigma}_\varepsilon^2$. La log-vraisemblance s'écrit :

$$\mathcal{L}(\beta_0, \beta_1, \sigma_\varepsilon^2) = -\frac{n}{2} \ln(\sigma_\varepsilon^2) - \frac{n}{2} \ln(2\pi) - \frac{1}{2\sigma_\varepsilon^2} \sum_{t=1}^n (y_t - \beta_0 - \beta_1 x_t)^2 \quad (9.8)$$

Pour estimer $\hat{\beta}_0, \hat{\beta}_1$ et $\hat{\sigma}_\varepsilon^2$ par maximum de vraisemblance, la démarche consiste à maximiser la fonction log-vraisemblance ci-dessus, ce qui revient à annuler ses dérivées premières par rapport aux paramètres β_0, β_1 et σ_ε^2 . On about à :

$$\begin{cases} \hat{\beta}_1^{\text{EMV}} = \frac{\text{Cov}(x_t, y_t)}{\text{V}(x_t)} \\ \hat{\beta}_0^{\text{EMV}} = \bar{y} - \hat{\beta}_1^{\text{EMV}} \bar{x} \\ \hat{\sigma}_{\varepsilon, \text{EMV}}^2 = \frac{1}{n} \sum_{t=1}^n (y_t - \hat{\beta}_0^{\text{EMV}} - \hat{\beta}_1^{\text{EMV}} x_t)^2 \end{cases}$$

Les deux premières expressions sont identiques aux équations normales fournies par les MCO. Par contre, la troisième équation est différente de celle obtenue avec les MCO. En effet, on a la relation :

$$\hat{\sigma}_{\varepsilon, \text{EMV}}^2 = \frac{n-2}{n} \hat{\sigma}_{\varepsilon, \text{MCO}}^2 \quad (9.9)$$

Puisque l'estimation MCO de la variance des erreurs est sans biais, celui du maximum de vraisemblance est biaisé, mais reste convergent. Cette dernière propriété garantit la minimisation du biais avec l'accroissement de la taille de l'échantillon.

On définit notre fonction log-vraisemblance en prenant l'opposé de cette fonction.

```
# Fonction log-vraisemblance
import numpy as np
```

```

def logvraisemblance(parameters):
    beta0 = parameters[0]
    beta1 = parameters[1]
    sigma = parameters[2]
    for i in np.arange(0, len(x)):
        y_exp = beta0 + beta1*x
        loglik = (len(x)/2 * np.log(2 * np.pi) + len(x)/2 * np.log(sigma ** 2) +
                  1 / (2 * sigma ** 2) * sum((y - y_exp) ** 2))

    return loglik

```

On estime nos paramètres grâce à la fonction `minimize` du package `Scipy`. Cette fonction recherche le minimum d'une fonction objectif.

```
# Estimation des paramètres par maximum de vraisemblance
from scipy.optimize import minimize
```

```

x = donnee.T12.values
y = donnee.O3.values
opt = minimize(logvraisemblance, np.array([1,1,1]), method="L-BFGS-B")
params = pd.DataFrame({'Params.':['Intercept','T12','scale'],
                       'Coef.':[opt.x[0],opt.x[1],opt.x[2]**2]})

print(params)

      Params.      Coef.
0   Intercept  31.415264
1         T12    2.701019
2       scale  403.588368

```

On voit que les valeurs estimées obtenues sont identiques pour les deux premiers paramètres, sauf celle liée à l'estimation de la variance de l'erreur.

9.2 Équation d'analyse de la variance

9.2.1 Équation de décomposition de la variance

Cette équation signifie que la variabilité totale est égale à la variabilité expliquée plus la variabilité résiduelle et permet de juger de la qualité de l'ajustement d'un modèle. Elle s'écrit :

$$\sum_{t=1}^n (y_t - \bar{y})^2 = \sum_{t=1}^n (\hat{y}_t - \bar{y})^2 + \sum_{t=1}^n (y_t - \hat{y}_t)^2 \quad (9.10)$$

où

$$SCT = SCE + SCR \quad (9.11)$$

- SCT représente la variabilité totale ou somme des carrés totaux : $SCT = \sum_{t=1}^n (y_t - \bar{y})^2$
- SCE est la variabilité due à la régression ou somme des carrés expliquées : $SCE = \sum_{t=1}^n (\hat{y}_t - \bar{y})^2$

- SCR est la variabilité due aux erreurs ou somme des carrés résiduels : $SCR = \sum_{t=1}^n (y_t - \hat{y}_t)^2$

9.2.2 Le coefficient de détermination R^2

C'est un indicateur permettant de mesurer la qualité d'ajustement d'un modèle. Il est défini par la relation :

$$R^2 = \frac{SCE}{SCT} = 1 - \frac{SCR}{SCT} \quad (9.12)$$

Pour un modèle linéaire de régression simple, on a la formule suivante :

$$R^2 = \hat{\beta}_1^2 \frac{\sum_{t=1}^n (x_t - \bar{x})^2}{\sum_{t=1}^n (y_t - \bar{y})^2} \quad (9.13)$$

Propriétés :

- On a toujours $0 < R^2 < 1$ et l'ajustement du modèle est d'autant meilleur que R^2 est proche de 1 ;
- R^2 est le carré du coefficient de corrélation linéaire $\rho_{x,y}$ et on a toujours : $-1 < \rho_{x,y} < 1$;

```
# Coefficient de détermination
r2squared = model.rsquared
print('Coefficient de détermination : %.4f'%(r2squared))
```

Coefficient de détermination : 0.2791

On obtient un score de 28% avec notre modèle. Cette valeur du R^2 est faible : peut-être qu'une régression linéaire simple n'est-elle pas adaptée ici.

Quelques limites de la notion de corrélation :

1. C'est une relation linéaire : L'application de la formule du calcul de $\rho_{x,y}$ ne permet de déterminer que des corrélations linéaires entre variables. Un coefficient de corrélation nul ($\rho_{x,y} = 0$) indique que la covariance entre x et y est nulle ($\text{Cov}(x,y)$). Mais, deux variables en totale dépendance peuvent avoir un $\rho_{x,y}$ lorsque la relation qui leur lie n'est pas linéaire.
2. La corrélation n'est pas causalité : Le fait d'avoir un coefficient de corrélation élevé entre deux variables ne signifie pas qu'il existe un lien statistique entre ces variables.

9.2.3 Tableau d'analyse de la variance

Source de variation	Somme des carré (SC)	Degré de liberté (ddl)	Carré moyen (SC/ddl)
Exogène(x)	SCE	1	$SCE/1$
Résiduelle	SCR	$n - 2$	$SCR/n - 2$
Totale	SCT	$n - 1$	

TABLE 9.2 – Tableau d'analyse de la variance

9.3 Prévision, tests de significativité des paramètres et intervalles de confiance

9.3.1 Prévision de la variable endogène

Supposons un horizon θ tel que $\theta > n$. Soit x_θ , la réalisation de la variable exogène à l'instant θ , alors la valeur prévue pour la variable endogène à l'instant θ est :

$$\hat{y}_\theta = \hat{\beta}_0 + \hat{\beta}_1 x_\theta \quad (9.14)$$

Toutefois, la vraie réalisation de la variable endogène à l'instant θ sera donnée par :

$$y_\theta = \beta_0 + \beta_1 x_\theta + \varepsilon_\theta \quad (9.15)$$

La prévision définie ci-dessus comporte ainsi une erreur définie par :

$$e_\theta = y_\theta - \hat{y}_\theta \quad (9.16)$$

avec e_θ l'erreur de prévision.

On démontre aisément que la variance de l'erreur de prévision est définie par :

$$V(e_\theta) = \sigma_\varepsilon^2 \left(1 + \frac{1}{n} + \frac{(x_\theta - \bar{x})^2}{\sum_{t=1}^n (x_t - \bar{x})^2} \right) \quad (9.17)$$

Mais, on ne connaît cette variance que par son estimateur défini par :

$$\hat{V}(e_\theta) = \hat{\sigma}_\varepsilon^2 \left(1 + \frac{1}{n} + \frac{(x_\theta - \bar{x})^2}{\sum_{t=1}^n (x_t - \bar{x})^2} \right) \quad (9.18)$$

On peut alors donner un intervalle de confiance au risque α pour la prévision de la variable endogène. En effet :

$$IC_{1-\alpha}(y_\theta) = \left[\hat{y}_\theta - t_{n-1}^{1-\alpha/2} \sqrt{\hat{V}(e_\theta)}; \hat{y}_\theta + t_{n-1}^{1-\alpha/2} \sqrt{\hat{V}(e_\theta)} \right] \quad (9.19)$$

où $t_{n-1}^{1-\alpha/2}$ est le quantile d'ordre $1 - \alpha/2$ de la loi de Student à $n - 2$ degrés de liberté.

```
# Prédiction
prediction = model.get_prediction(exog=donnée['T12'])
frame= prediction.summary_frame(alpha=0.05)
print(frame.head().round(3))
```

	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
0	67.609	5.216	57.121	78.097	25.070	110.148
1	71.930	4.418	63.047	80.814	29.759	114.102
2	52.753	8.305	36.055	69.451	8.274	97.232
3	66.799	5.374	55.994	77.603	24.181	109.416
4	69.500	4.858	59.732	79.267	27.133	111.866

TABLE 9.3 – Prévision sur les valeurs observées

```
plt.figure(figsize=(14,8))
plt.scatter(donnée['T12'], donnée['O3'], color="blue")
plt.plot(donnée['T12'],frame['mean'],color='green',label = 'valeur ajustée')
plt.plot(donnée['T12'],frame['mean_ci_lower'], color='black',linestyle='--')
plt.plot(donnée['T12'],frame['mean_ci_upper'], color='black',linestyle='--',label="")
plt.plot(donnée['T12'],frame['obs_ci_lower'], color='red',linestyle='--')
plt.plot(donnée['T12'],frame['obs_ci_upper'], color='red',linestyle='--')
for i in range(len(donnée)):
    plt.annotate(i+1,(donnée['T12'].values[i],donnée['O3'].values[i]),
                 fontsize=13)
plt.grid()
plt.legend()
plt.xlabel('T12')
plt.ylabel('O3')
plt.title("Nuage de points : O3 en fonction de T12")
plt.show()
```

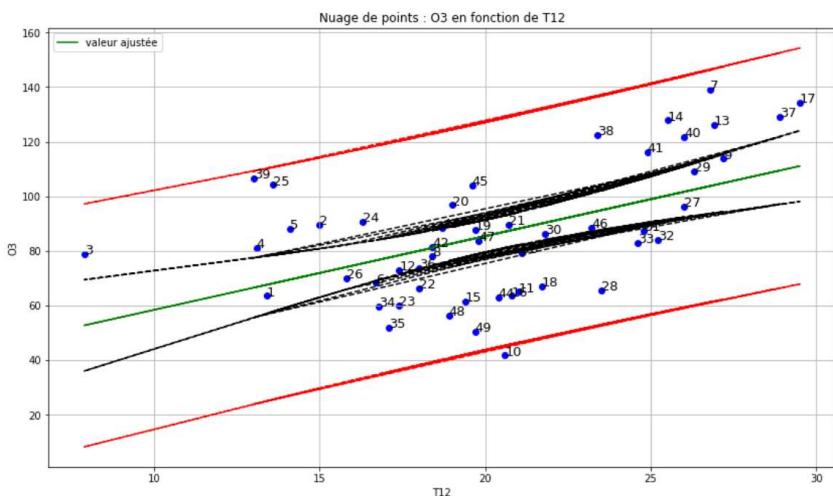


FIGURE 9.3 – Prévision - intervalle de confiance

9.3.2 Tests de significativité

Cette construction est facilitée par l'hypothèse de normalité des erreurs. En effet, on admet que $\varepsilon_t \sim \mathcal{N}(0, \sigma_\varepsilon^2)$. Cette hypothèse n'est indispensable pour obtenir des estimateurs convergents des paramètres du modèle. Mais elle est utilisée pour la construction des tests statistiques et intervalle de confiance. Les trois tests suivantes sont équivalents et portent sur la significativité du modèle linéaire simple :

	Hypothèses	
Test 1	$H_0 : \beta_1 = 0$	$H_1 : \beta_1 \neq 0$
Test 2	$H_0 : \rho_{xy} = 0$	$H_1 : \rho_{xy} \neq 0$
Test 3	$H_0 : SCE = 0$	$H_1 : SCE \neq 0$

i) Test 1

Il porte sur le coefficient β_1 (test de significativité de β_1). Pour effectuer ce test, on admet que :

$$\frac{\hat{\beta}_1 - \bar{a}}{\sigma_{\hat{\beta}_1}} \sim \mathcal{N}(0,1) \quad (9.20)$$

Cela est valable aussi bien pour β_0 que pour β_1 . \bar{a} représente un nombre donné. Dans le test de significativité d'un coefficient β_1 , $\bar{a} = 0$. On a donc que :

$$\frac{\hat{\beta}_1}{\sigma_{\hat{\beta}_1}} \sim \mathcal{N}(0,1) \quad (9.21)$$

$\hat{\sigma}_{\hat{\beta}_1}$ est l'estimateur de $\sigma_{\hat{\beta}_1}$. D'où :

$$t_{cal} = \frac{\hat{\beta}_1}{\hat{\sigma}_{\hat{\beta}_1}} \sim t_{n-2} \quad (9.22)$$

On rejette l'hypothèse nulle (H_0) au risque α si $|t_{n-2}^{1-\alpha/2}| > t_{cal}$. Rejeter H_0 signifie que le coefficient β_1 est significativement différent de zéro et dans le cas d'un modèle linéaire simple, cela signifie que le modèle est significatif.

```
#Test de significativité des paramètres
ttest = pd.concat([model.tvalues,model.pvalues],axis=1)
ttest.columns = ['t','P>|t|']
print(ttest)

          t      P>|t|
Intercept  2.405723  0.020042
T12        4.310650  0.000080
```

$P>|t|$ est appelée probabilité critique ou « p-value ». Elle représente la probabilité, pour la statistique de test sous H_0 , de dépasser la valeur estimée. Ici, nous rejetons l'hypothèse H_0 pour les deux paramètres estimés au niveau $\alpha = 5\%$.

ii) Test 2

C'est le test de significativité du coefficient de corrélation linéaire ρ_{xy} . Pour l'effectuer, on :

- Calcul d'abord le coefficient de corrélation empirique : $\rho_{xy} = \frac{\text{Cov}(x,y)}{\sigma_x \sigma_y}$
- Calcule la statistique du test : $t_{cal} = \frac{\rho_{xy}}{\sqrt{\frac{1 - \rho_{xy}^2}{n - 2}}}$

On rejette l'hypothèse nulle H_0 au risque α si $|t_{n-2}^{1-\alpha/2}| > t_{cal}$.

iii) Test 3

C'est le test de significativité du modèle encore appelé test par analyse de la variance. Pour effectuer ce test, la statistique calculée est celle de Fisher :

$$F_{cal} = \frac{\frac{SCE}{ddl_{SCE}}}{\frac{SCR}{n-2}} = \frac{\frac{SCE}{SCR}}{n-2} = \frac{\frac{R^2}{1-R^2}}{n-2} \quad (9.23)$$

On rejette l'hypothèse nulle H_0 au risque α si F_{cal} est supérieur à la valeur lue sur la table de la loi de Fischer à 1 et $n - 2$ degrés de liberté.

Avec les 3 tests ci-dessus, le rejet de H_0 signifie que le modèle est significatif. On peut aussi montrer que $F_{cal} = (t_{cal})^2$.

```
# Test de Fisher
import numpy as np
fisher = pd.DataFrame(np.transpose([model.df_model, model.df_resid,
                                    model.fvalue, model.f_pvalue])).T
fisher.columns = ['d1', 'd2', 'f', 'pvalue']
fisher.index = ['f-test']
print(fisher)

      d1      d2          f    pvalue
f-test  1.0   48.0  18.581706  0.00008
```

Test portant sur un paramètre

On rappelle que le modèle considéré est défini par l'équation (9.1). Nous voulons tester au risque α l'hypothèse nulle $H_0 : \beta_i = \bar{a}$ (où \bar{a} est une valeur connue et $i = 0,1$) contre une hypothèse alternative. Pour effectuer ce test, la statistique à calculer est :

$$t_{cal} = \frac{\hat{\beta}_i - \bar{a}}{\hat{\sigma}_{\hat{\beta}_i}} \sim t_{n-2} \quad (9.24)$$

Le rejet de décision est indiqué dans le tableau :

Hypothèse		Rejeter H_0 au risque α
$H_0 : \beta_i = \bar{\alpha}$	$H_0 : \beta_i \neq \bar{\alpha}$	$ t_{n-2}^{1-\alpha/2} > t_{cal}$
$H_0 : \beta_i = \bar{\alpha}$	$H_0 : \beta_i > \bar{\alpha}$	$t_{cal} > t_{n-2}^{1-\alpha}$
$H_0 : \beta_i = \bar{\alpha}$	$H_0 : \beta_i < \bar{\alpha}$	$t_{cal} < -t_{n-2}^{1-\alpha}$

où $t_{n-2}^{1-\alpha/2}$ et $t_{n-2}^{1-\alpha}$ sont les quantiles d'ordre respectifs $1 - \alpha/2$ et $1 - \alpha$ de la loi de Student à $n - 2$ degrés de libertés (n est le nombre d'observation).

9.3.3 Intervalle de confiance pour β_i

Du test bilatéral portant sur le paramètre β_i tel que défini ci-dessus, on peut déduire qu'un intervalle de confiance du coefficient β_i au risque α est donné par :

$$IC_{1-\alpha}(\beta_i) = \left[\hat{\beta}_i - \hat{\sigma}_{\hat{\beta}_i} t_{n-2}^{1-\alpha/2}; \hat{\beta}_i + \hat{\sigma}_{\hat{\beta}_i} t_{n-2}^{1-\alpha/2} \right] \quad (9.25)$$

Intervalle de confiance des paramètres

```
confint = model.conf_int(alpha=0.05)
```

```
confint.columns = ['B.inf', 'B.sup']
```

```
print(confint)
```

	B.inf	B.sup
Intercept	5.159232	57.670715
T12	1.441180	3.960890

La première ligne correspond à l'intervalle de confiance de β_0 et la seconde correspond à l'intervalle de confiance de β_1 . L'intervalle de confiance à 95% sur l'ordonnée à l'origine est étendu (52.5). Cela provient des erreurs (l'estimation de σ est de 20.5), mais surtout du fait que les températures sont en moyenne très loin de 0. Cependant ce coefficient en fait pas très souvent l'objet d'interprétation. L'autre IC à 95% est moins étendu (2.5). Nous constatons qu'il semble exister un effet de la température sur les pics d'ozone, bien que l'on pose la question de la validité de l'hypothèse linéaire.

9.3.4 Région de confiance simultanée de β_0 et β_1

Une région de confiance simultanée des deux paramètres inconnus $\beta = (\beta_0, \beta_1)$ est donnée par l'équation suivante :

$$\frac{1}{2\hat{\sigma}^2} \left[n(\hat{\beta}_0 - \beta_0)^2 + 2n\bar{x}(\hat{\beta}_0 - \beta_0)(\hat{\beta}_1 - \beta_1) + \sum_{t=1}^n x_t^2(\hat{\beta}_1 - \beta_1)^2 \right] \leq f_{(2,n-2)}(1 - \alpha) \quad (9.26)$$

où $f_{(2,n-2)}(1 - \alpha)$ représente le fractile de niveau $(1 - \alpha)$ d'une loi de Fisher à $(2, n - 2)$ degrés de liberté.

FIGURE 9.4 – Région de confiance simultanée des deux paramètres

Les axes de l'ellipse ne sont pas parallèles aux axes du graphique, les deux estimateurs sont corrélés. Nous retrouvons que la corrélation entre les deux estimateurs est toujours négative (ou nulle), le plus axe de l'ellipse ayant une pente négative. Nous observons bien sûr une différence entre le rectangle de confiance, juxtaposition des deux intervalles de confiance et l'ellipse.

9.3.5 Intervalle de confiance pour σ^2

Un intervalle de confiance de σ^2 est donné par :

$$IC(\sigma_\varepsilon^2) = \left[\frac{(n-2)}{\chi_1^2} \hat{\sigma}_\varepsilon^2, \frac{(n-2)}{\chi_2^2} \hat{\sigma}_\varepsilon^2 \right] \quad (9.27)$$

avec χ_1^2 ayant la probabilité $\alpha/2$ d'être dépassée et χ_2^2 la probabilité $1 - \alpha/2$ d'être dépassée à $n-p-1$ degrés de liberté.

CHAPITRE 10

MODÈLE LINÉAIRE DE RÉGRESSION MULTIPLE

Sommaire

10.1 Spécification du modèle	236
10.2 Estimation et propriétés des estimateurs	239
10.3 L'analyse de la variance	243
10.4 Quelques tests et intervalles de confiance	245
10.5 Influence et diagnostics	249
10.6 Autres mesures de diagnostiques	259

Le modèle linéaire de régression multiple [voir [B+21]] est une extension du modèle linéaire simple. Ici, plusieurs variables exogènes expliquent le comportement de la variable endogène.

10.1 Spécification du modèle

10.1.1 Présentation du modèle

Spécifié en séries temporelles, on a le modèle suivant :

$$y_t = \beta_0 + \beta_1 x_{1t} + \cdots + \beta_p x_{pt} + \varepsilon_t \quad \forall t = 1, \dots, n \quad (10.1)$$

où y_t est la variable à expliquer au temps t et les x_{jt} ($j = 1, \dots, p$) les p variables explicatives au temps t . β_0, \dots, β_p les $(p+1)$ paramètres du modèle. Comme au chapitre précédent, ε_t représente l'erreur de spécification, c'est-à-dire la différence entre le modèle vrai et le modèle spécifié. n est le nombre d'observation.

Si on était dans un univers déterministe, la loi en question serait parfaitement vérifiée, mais ce n'est pas le cas pour plusieurs raisons :

- Les phénomènes économiques sont caractérisés par l'interdépendance entre nombreux éléments, ce qui entraîne que les variables explicatives susceptibles d'exercer une influence sur la variable expliquée sont nombreuses. Mais l'effet des variables qui ont été omises explique qu'il y ait des écarts entre la réalité observée et le résultat de la fonction ;

- Les phénomènes économiques sont mis en œuvre par des individus qui n'ont eux-mêmes pas un comportement déterminé. C'est donc leur libre arbitre qui fait que parfois ils n'agissent pas comme prévue et on n'obtient pas les résultats escomptés.

Les lois économiques sont donc seulement vraies en moyenne. Elles ont le caractère de lois statistiques. On choisit de traduire les écarts entre la réalité observée et les résultats de l'estimation par une variable aléatoire notée ε_t encore appelée terme d'erreur.

Compte tenu de l'existence de ce terme, la loi qui va gouverner la variable expliquée (y_t) n'est pas une loi mathématique mais une loi statistique et il faudra se donner une distribution pour caractériser ε_t .

L'estimation des coefficients β_0, \dots, β_p fournit des valeurs qui seront elles-mêmes des variables aléatoires ayant une distribution de probabilité que l'on devra spécifier selon la distribution de ε_t .

On dispose d'un ensemble d'observation (données) portant à la fois sur la variable expliquée et sur les variables explicatives. Sachant que la relation a un caractère probabiliste, il n'est pas possible de déterminer de façon unique les vraies valeurs des paramètres.

Les paramètres estimés sont désignés par $\hat{\beta}_0, \dots, \hat{\beta}_p$ et les connaissant on pourra remonter aux valeurs vraies des paramètres.

Pour que cette opération d'inférence statistique soit possible, il faut qu'un certains nombres d'hypothèses soit vérifié, sinon, on aura toujours un résultat numérique sans valeur car il ne permet pas de porter un jugement sur les vraies valeurs des coefficients recherchés.

10.1.2 Le modèle sous forme matricielle

En écrivant le modèle observation par observation, on obtient :

$$\begin{cases} y_1 = \beta_0 + \beta_1 x_{11} + \cdots + \beta_p x_{p1} + \varepsilon_1 \\ \vdots \\ y_n = \beta_0 + \beta_1 x_{1n} + \cdots + \beta_p x_{pn} + \varepsilon_n \end{cases}$$

Soit, sous forme matricielle, ce modèle s'écrit :

$$Y_{(n,1)} = X_{(n,p+1)} \beta_{(p+1,1)} + \varepsilon_{(n,1)} \quad (10.2)$$

avec

$$Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \quad X = \begin{pmatrix} 1 & \cdots & x_{p1} \\ \vdots & & \vdots \\ 1 & \vdots & x_{pn} \end{pmatrix}, \quad \beta = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

10.1.3 Hypothèses de base

Les hypothèses en question portent sur les variables ainsi que sur le terme d'erreur (ε_t).

A) Les hypothèses classiques (hypothèses stochastiques)

Hypothèse 1 : Les variables sont observées sans erreur

Cela suppose que la collecte des informations statistiques a été faite sans erreur. On peut cependant avoir une petite idée sur la fiabilité des chiffres utilisés car le degré de précision de la méthode d'estimation est dans une certaine mesure limité par la qualité des données. Il ne sert à rien d'utiliser des méthodes sophistiquées sur des données douteuses.

Hypothèse 2 : Le terme d'erreur a une espérance mathématique nulle : $E(\varepsilon) = 0$

C'est-à-dire qu'on se trompe tantôt dans un sens et tantôt dans l'autre sens, l'erreur n'est donc pas systématique.

Hypothèse 3 : Le terme d'erreur a une variance qui est constante : $V(\varepsilon) = \sigma_\varepsilon^2 I_n$

C'est ce qu'on appelle l'homoscédasticité c'est-à-dire la marge d'erreur est supposée identique en tout point de l'échantillon. C'est une hypothèse parfois contestable dans la mesure où on a des raisons de penser que la variance peut augmenter ou diminuer sur l'ensemble des observations. Cette hypothèse peut être testée et quand on a repéré l'hétérosécédasticité, on peut la corriger et obtenir néanmoins des bons estimateurs.

Hypothèse 4 : Les termes d'erreur sont indépendants les uns des autres dans le temps : $\text{Cov}(\varepsilon_t, \varepsilon_s) = 0, \quad \forall t \neq s$

Cette hypothèse traduit le fait que l'erreur n'est pas systématique mais aussi le fait que la dispersion est constante sur l'ensemble de l'échantillon (non corrélation des erreurs). Cette hypothèse peut être testée et lorsqu'elle n'est vérifiée, on parle d'auto-corrélation des erreurs. Cette situation est due très souvent à la mauvaise spécification du modèle. En effet, une spécification défectueuse est de nature à entraîner les erreurs systématiques. La mauvaise spécification peut être due à l'omission d'une variable importante dans la modélisation, aux choix erronés du type de fonction qui relie la variable expliquée aux variables explicatives.

Hypothèse 5 : Les variables explicatives x_{1t}, \dots, x_{pt} sont indépendantes des termes de l'erreur : $\text{Cov}(X, \varepsilon) = 0$

Dans le cas contraire, on va dire que les variables explicatives sont entachées des erreurs et on utilise la méthode des variables instrumentales (MVI) pour corriger ce problème.

Hypothèse 6 : Les termes de l'erreur suivent une loi normale

C'est grâce à cette hypothèse que l'inférence statistique peut se réaliser car c'est elle qui va préciser la distribution statistique des estimateurs $\hat{\beta}$. Elle joue donc un rôle important essentiel bien qu'elle soit difficile à vérifier. Il existe des tests de normalité, mais ils ne sont applicables sur les termes d'erreur qui par définition demeure inconnu.

B) Les hypothèses structurelles

Hypothèse 7 : Les variables explicatives x_{1t}, \dots, x_{pt} ne sont pas linéairement dépendantes (absence de colinéarité des variables explicatives)

Ceci implique que la matrice $X'X$ est régulière et que la matrice inverse $(X'X)^{-1}$ existe. Si deux ou plusieurs variables explicatives sont fortement liées, on parle de colinéarité des variables

explicatives.

Hypothèses 8 : $(X'X)/n$ tend vers une matrice finie non singulière

Hypothèse 9 : n (le nombre d'observations) est supérieur à $p + 1$ (le nombre de séries explicatives).

10.2 Estimation et propriétés des estimateurs

10.2.1 Estimation des coefficients de régression par MCO

Le modèle sous forme matricielle à p variables explicatives et n observations s'écrit :

$$Y = X\beta + \varepsilon \quad (10.3)$$

Pour estimer le vecteur β composé des coefficients β_0, \dots, β_p , nous appliquerons la méthode des moindres carrés ordinaires qui consiste à minimiser la somme des carrés des erreurs, soit :

$$\min_{\beta} \varepsilon' \varepsilon = \min_{\beta} (Y - X\beta)' (Y - X\beta) \quad (10.4)$$

Ce programme admet son minimum lorsque la dérivée partielle est nulle. On obtient l'estimateur suivant :

$$\hat{\beta} = (X'X)^{-1} X' Y \quad (10.5)$$

Les équations issues de la relation $X'X\hat{\beta} = X'Y$ sont appelées équations normales.

Remarque :

- $\hat{\beta}$ est un estimation sans biais et convergent. $\hat{\beta}$ est un estimateur BLUE(*Best Linear Unbiased Estimator*).
- La matrice de variance covariance de $\hat{\beta}$ est donnée par : $\hat{\Omega}_{\hat{\beta}} = \hat{\sigma}_{\varepsilon}^2 (X'X)^2$ où $\hat{\sigma}_{\varepsilon}^2$ est l'estimateur sans biais et convergent de σ_{ε}^2 avec :

$$\hat{\sigma}_{\varepsilon}^2 = \frac{1}{n-p-1} \sum_{t=1}^n (y_t - \hat{\beta}_0 - \hat{\beta}_1 x_{1t} - \dots - \hat{\beta}_p x_{pt})^2 \quad (10.6)$$

Application

Nous cherchons à expliquer l'ozone (03) par deux variables explicatives, la température à 12h (T12) et le vent (Vx). Le vent est mesuré en degré (direction) et mètre par seconde (vitesse).

$$03 = f(T12, Vx) \quad (10.7)$$

Nous chargeons nos données.

```
# Chargement des données
import pandas as pd

donnee = pd.read_csv('ozone.txt', sep=';', index_col=0)
print(donnee[['03', 'T12', 'Vx']].head(10))
```

Date	O3	T12	Vx
1996-04-22	63.6	13.4	9.35
1996-04-29	89.6	15.0	5.40
1996-05-06	79.0	7.9	19.30
1996-05-14	81.2	13.1	12.60
1996-05-21	88.0	14.1	-20.30
1996-05-28	68.4	16.7	-3.69
1996-06-05	139.0	26.8	8.27
1996-06-12	78.2	18.4	4.93
1996-06-19	113.8	27.2	-4.93
1996-06-27	41.8	20.6	-3.38

TABLE 10.1 – Caption

Visualisons la structure de relation de nos données prises deux à deux.

```
# Pairsplot
import seaborn as sns
import matplotlib.pyplot as plt

sns.pairplot(donnee[['O3', 'T12', 'Vx']], height=2.5, corner=True)
plt.show()
```

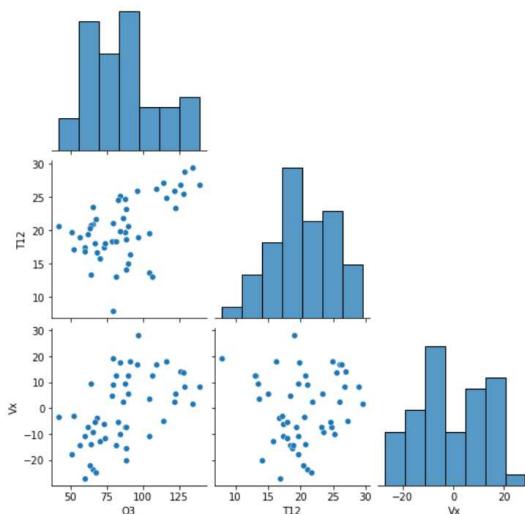


FIGURE 10.1 – Pairs plot

On calculons la corrélation linéaire de Pearson entre nos variables.

```
#Matrice des corrélations
matrice_corr = donnee[['O3', 'T12', 'Vx']].corr().round(2)
fig, axes = plt.subplots(figsize=(8,6))
```

```

sns.heatmap(data=matrice_corr, annot=True, vmin=-1, vmax=1,
            center = 0, cmap = 'RdBu', linewidths=0.5, ax=axes)
plt.title('Heatmap des correlations croisées entre variables')
plt.show()

```

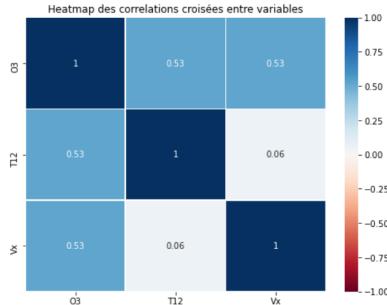


FIGURE 10.2 – Matrice des corrélations

Faisons une représentation en 3D.

```

# Nuage de points en 3D
from mpl_toolkits import mplot3d

fig = plt.figure(figsize = (12,8))
axes = plt.axes(projection = "3d")
axes.scatter3D(donnee['T12'], donnee['Vx'], donnee['O3'], color="blue")
plt.title("Nuage de points : O3 en fonction de T12 et Vx")
axes.set_xlabel('T12')
axes.set_ylabel('Vx')
axes.set_zlabel('O3')
plt.show()

```

NUAGE DE POINTS : O3 EN FONCTION DE T12 ET VX

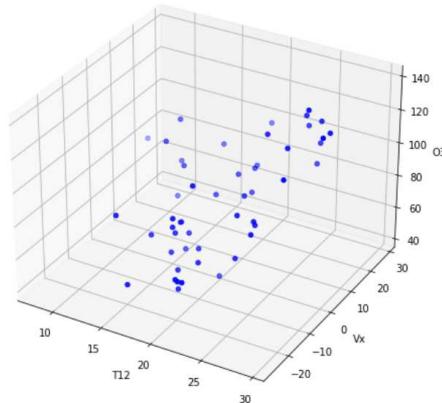


FIGURE 10.3 – Nuage de points de O3 en fonction de T12 et Vx

Il est très difficile de voir si une régression est adaptée, ce qui signifie ici que les points ne doivent pas être très éloignés d'un plan commun. Le modèle de régression est :

$$03 = \beta_0 + \beta_1 T12 + \beta_2 Vx + \varepsilon \quad (10.8)$$

```
# Estimation des paramètres
from statsmodels.api import OLS

model = OLS.from_formula(formula = '03~T12+Vx', data=donnee).fit()
# resultat
print(model.summary2())

Results: Ordinary least squares
=====
Model:                 OLS                  Adj. R-squared:      0.505
Dependent Variable: 03                   AIC:                427.0664
Date:          2022-02-09 09:00 BIC:                432.8025
No. Observations:   50                  Log-Likelihood:    -210.53
Df Model:            2                  F-statistic:        25.96
Df Residuals:       47                  Prob (F-statistic): 2.54e-08
R-squared:           0.525               Scale:              282.97
-----
          Coef.    Std.Err.      t     P>|t|      [0.025  0.975]
-----
Intercept      35.4530   10.7446   3.2996  0.0019  13.8377  57.0683
T12             2.5380    0.5151   4.9270  0.0000   1.5017  3.5744
Vx              0.8736    0.1772   4.9309  0.0000   0.5172  1.2300
-----
Omnibus:          0.280            Durbin-Watson:      1.678
Prob(Omnibus):  0.869            Jarque-Bera (JB):  0.331
Skew:             0.165            Prob(JB):          0.848
Kurtosis:         2.777            Condition No.:    94
=====

# variance de l'erreur
scale = model.scale
print("Variance de l'erreur : %.4f"%(scale))

Variance de l'erreur : 282.9659
```

L'estimation de $\hat{\sigma}_\varepsilon$ vaut ici 16.82 et nous avons $n = 50$ pour $p = 2$ variables, ce qui donne $n - p - 1 = 47$ (degrés de liberté).

10.2.2 Estimation des coefficients de régression par MV

A partir de l'hypothèse 6, nous avons $\varepsilon \sim \mathcal{N}(0, \sigma^2 I_n)$. La vraisemblance s'écrit :

$$L(\beta, \sigma^2) = \left(\frac{1}{2\pi\sigma^2} \right)^{n/2} \exp \left(-\frac{1}{2\sigma^2} \sum_{t=1}^n \left(y_t - \beta_0 - \sum_{j=1}^p \beta_j x_{tj} \right)^2 \right) \quad (10.9)$$

La fonction log-vraisemblance associée s'écrit :

$$\mathcal{L}(\beta, \sigma^2) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \|Y - X\beta\|^2 \quad (10.10)$$

Nous avons :

$$\begin{cases} \hat{\beta}_{MV} = \hat{\beta} \\ \hat{\sigma}_{MV}^2 = \frac{1}{n} \|Y - X\hat{\beta}_{MV}\|^2 \end{cases}$$

et donc que $\hat{\sigma}_{MV}^2 = \frac{n-p-1}{n} \hat{\sigma}^2$.

```
# fonction log-vraisemblance
import numpy as np

def logvraisemblance(parameters):
    beta0 = parameters[0]
    betap = parameters[1:-1]
    sigma = parameters[-1]
    for i in np.arange(0, len(x)):
        y_exp = beta0 + np.dot(x, betap)
        loglik = (len(x)/2 * np.log(2 * np.pi) + len(x)/2 * np.log(sigma ** 2) +
                   1 / (2 * sigma ** 2) * np.linalg.norm(y - y_exp)**2)
    return loglik

# Estimation des paramètres par maximum de vraisemblance
from scipy.optimize import minimize

x = donnee[['T12', 'Vx']].values
y = donnee.O3.values
opt = minimize(logvraisemblance, np.array([1, 1, 1, 1]), method="L-BFGS-B")
params = pd.DataFrame({'Params.': ['Intercept', 'T12', 'Vx', 'scale'],
                       'Coef.': [opt.x[0], opt.x[1], opt.x[2], opt.x[3]**2]})
print(params.round(4))

      Params.      Coef.
0  Intercept  35.4530
1        T12   2.5380
2        Vx    0.8736
3     scale  265.9882
```

Les valeurs estimées des paramètres sont identiques à celles obtenues les MCO, sauf celle de la variance de l'erreur. On s'attendait à ce résultat.

10.3 L'analyse de la variance

L'analyse de la variance permet d'appréhender l'influence des variables exogènes sur la variable endogène.

10.3.1 Équation d'analyse de la variance

Cette équation de décomposition de la variance qui permet de juger de la qualité de l'ajustement d'un modèle stipule que la variabilité totale est égale à la variabilité expliquée plus la variabilité des résidus. Elle s'écrit :

$$SCT = SCE + SCR \quad (10.11)$$

avec $SCT = y' y - n\bar{y}^2$; $SCE = \hat{y}' \hat{y} - n\bar{y}^2$ et $SCR = (y - \hat{y})'(y - \hat{y})$

Cette équation permet d'apprecier la qualité de l'ajustement du modèle considéré; en effet, plus la variance expliquée (SCE) est « proche » de la variance totale (SCT) ($SCE \rightarrow SCT$), meilleur est l'ajustement global du modèle.

Mais puisque ces valeurs dépendent des unités de mesure, on préfère utiliser un nombre sans dimension pour juger de la qualité d'un ajustement : le coefficient de détermination multiple.

10.3.2 Le coefficient de détermination multiple : R^2

Il est défini par :

$$R^2 = \frac{SCE}{SCT} \quad (10.12)$$

R^2 mesure la proportion de la variance de la variable endogène expliquée par la régression de cette dernière sur la variable exogène. Plus précisément, R^2 montre le rôle joué par l'ensemble des variables exogènes sur l'évolution de la variable endogène. Il d'autant meilleur qu'il est proche de 1.

```
# Coefficient de détermination
rsquared = model.rsquared
print('Coefficient de détermination : %.4f'%(r2squared))
```

Coefficient de détermination : 0.5249

10.3.3 Le coefficient de détermination ajustée ou corrigé : \bar{R}^2

Toutefois, le coefficient de détermination multiple seul est insuffisant pour juger de la qualité d'un modèle. Il masque l'influence des variables exogènes considérées isolément sur le comportement de l'endogène. Il ne peut donc pas se substituer au test des paramètres du modèle. En outre, ce coefficient ne tient compte ni du nombre d'observations, ni du nombre de variables explicatives dans le modèle.

C'est compte tenu de ces insuffisances que l'on a convenu de proposer un coefficient de détermination noté \bar{R}^2 , appelé « R^2 corrigé ou ajusté », défini par :

$$\bar{R}^2 = 1 - \frac{n-1}{n-p-1}(1-R^2) \quad (10.13)$$

```
# Coefficient de détermination ajusté
adjr2squared = model.rsquared_adj
print('Coefficient de détermination adj. : %.4f'%(adjr2squared))
```

Coefficient de détermination adj. : 0.5047

10.3.4 Tableau d'analyse de la variance

Source de variation	Somme des carré (SC)	Degré de liberté (ddl)	Carré moyen (SC/ddl)
Exogène	SCE	p	SCE/k
Résiduelle	SCR	$n - p - 1$	$SCR/n - p - 1$
Total	SCT	$n - 1$	

TABLE 10.2 – Tableau d'analyse de la variance

10.4 Quelques tests et intervalles de confiance

Les tests et intervalles classiques de confiance relatifs aux paramètres du modèle s'effectuent suivant la même logique qu'avec le modèle linéaire simple ; le degré de liberté, qui était de $(n - 2)$, est maintenant à $(n - p - 1)$.

10.4.1 Intervalle de confiance de la variance de l'erreur

L'intervalle de confiance de la variance de l'erreur permet de déterminer une fourchette de variation de l'amplitude de l'erreur. Pour un intervalle à $(1 - \alpha)\%$, il est donné par :

$$IC(\sigma_{\varepsilon}^2) = \left[\frac{(n - p - 1)}{\chi_1^2} \hat{\sigma}_{\varepsilon}^2; \frac{(n - p - 1)}{\chi_2^2} \hat{\sigma}_{\varepsilon}^2 \right] \quad (10.14)$$

avec χ_1^2 ayant la probabilité $\alpha/2$ d'être dépassée et χ_2^2 la probabilité $1 - \alpha/2$ d'être dépassée à $n - p - 1$ degrés de liberté. On rappelle que :

$$\frac{(n - p - 1) \hat{\sigma}_{\varepsilon}^2}{\sigma_{\varepsilon}^2} \sim \chi_{(n-p-1)}^2 \quad (10.15)$$

10.4.2 Test de comparaison d'un ensemble de paramètres à un ensemble de valeur

Il s'agit de tester simultanément l'égalité d'un sous ensemble de coefficients de régression à des valeurs fixées. Ce sont des tests du genre :

$$\begin{cases} H_0 : \beta_q = \bar{\beta}_q \\ H_1 : \beta_q \neq \bar{\beta}_q \end{cases}$$

q étant le nombre de coefficients retenus, c'est-à-dire la dimension de chacun des vecteurs β_q . Pour effectuer ce test, on calcule la statistique de Fischer suivante :

$$F_0 = \frac{1}{q} \left(\hat{\beta}_q - \bar{\beta}_q \right)' \hat{\Omega}_{\hat{\beta}_q}^{-1} \left(\hat{\beta}_q - \bar{\beta}_q \right) \quad (10.16)$$

On rejette l'hypothèse nulle au risque α si $F_0 > F_{\alpha}(q, n - p - 1)$.

On souhaite vérifier la nullité de tous les coefficients à l'exception de la constante. L'hypothèse nulle peut s'écrire sous une forme matricielle :

$$H_0 : R\beta = 0 \quad (10.17)$$

où β est le vecteur des coefficients et R la matrice des contraintes définie par :

$$R = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

En effet, en développant, nous avons :

$$H_0 : \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \iff H_0 : \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

```
# Test de nullité simultanée des paramètres
import numpy as np
```

```
# Matrice des contraintes
R = np.array([[0,1,0],[0,0,1]])
# Test statistique
print(model.f_test(R))
```

```
<F test: F=array([[25.96040746]]), p=2.541201206069178e-08, df_denom=47, df_num=2>
```

10.4.3 Test de significativité globale du modèle

Ce test peut se faire de deux manières différentes :

A) Soit par analyse de la variance

Dans ce cas, les hypothèses sont :

$$\begin{cases} H_0 : SCE = 0 \\ H_1 : SCE \neq 0 \end{cases}$$

La statistique à utiliser est celle de Fischer et se calcule comme suit :

$$F_{cal} = \frac{\frac{SCE}{ddl_{SCE}}}{\frac{SCR}{ddl_{SCR}}} = \frac{\frac{SCE}{p}}{\frac{SCR}{n-p-1}} = \frac{\frac{R^2}{p}}{\frac{1-R^2}{n-p-1}} \quad (10.18)$$

On rejette l'hypothèse nulle au risque α si $F_{cal} > F_\alpha(p, n - p - 1)$.

```
# Test de significativité globale
fstatistic = model.mse_model/model.mse_resid
print('f-statistic : %.2f' % (fstatistic))

f-statistic : 25.96

# Test de Fisher
import numpy as np
fisher = pd.DataFrame(np.transpose([model.df_model, model.df_resid,
                                    model.fvalue, model.f_pvalue])).T
fisher.columns = ['d1', 'd2', 'f', 'pvalue']
fisher.index = ['f-test']
print(fisher)

      d1      d2          f      pvalue
f-test  2.0   47.0  25.960407  2.541201e-08
```

B) Soit comme un test de comparaison d'un ensemble de paramètre à un ensemble de valeurs fixées

Nous allons effectuer un test de significativité des paramètres.

```
#Test de significativité des paramètres
ttest = pd.concat([model.tvalues,model.pvalues],axis=1)
ttest.columns = ['t','P>|t|']
print(ttest)

          t      P>|t|
Intercept  3.299615  0.001852
T12        4.927001  0.000011
Vx         4.930910  0.000011
```

Toutes les variables sont pertinentes. On construit les intervalles de confiance pour nos paramètres.

```
# Intervalle de confiance des paramètres
confint = model.conf_int(alpha=0.05)
confint.columns = ['B.inf','B.sup']
print(confint)

          B.inf      B.sup
Intercept  13.837681  57.068331
T12        1.501738   3.574355
Vx         0.517166   1.229975
```

10.4.4 Prévision de la variable endogène

Le modèle spécifié s'écrit :

$$Y = X\beta + \varepsilon \quad (10.19)$$

Le modèle estimé à l'aide d'un échantillon de taille n observations s'écrit :

$$Y = X\hat{\beta} + e \quad (10.20)$$

La prédiction pour la période $(t+h)$ est donnée par l'équation :

$$\hat{y}_t = \hat{\beta}_0 + \sum_{j=1}^p \hat{\beta}_j x_{pt+h} \quad (10.21)$$

L'erreur de prévision est donnée par :

$$e_{t+h} = y_{t+h} - \hat{y}_{t+h} \quad (10.22)$$

La variance estimée de l'erreur de prévision est donnée par :

$$\hat{\sigma}_{e_{t+h}}^2 = \hat{\sigma}_\varepsilon^2 \left(1 + X'_{t+h} (X' X)^{-1} X_{t+h} \right) \quad (10.23)$$

où $X_{t+h} = \begin{pmatrix} 1 \\ x_{1t+h} \\ \dots \\ x_{pt+h} \end{pmatrix}$ est le vecteur des réalisations de la variable exogène à la prévision $t + h$.

Il est ensuite possible de déterminer un intervalle de confiance de la prévision considérée, soit :

$$\text{IC}(y_{t+h}) = \left[\hat{y}_{t+h} \pm t_{(n-k-1)}^{1-\alpha/2} \hat{\sigma}_\varepsilon \sqrt{1 + X'_{t+h}(X'X)^{-1}X_{t+h}} \right] \quad (10.24)$$

On calcule la valeur prédictive pour les valeurs $\text{T12} = 5$ et $\text{Vx}=0$.

```
# Prévision
prediction = model.get_prediction(exog=dict(T12=5,Vx=0))
frame = prediction.summary_frame(alpha=0.05)
print(frame.round(3))

mean mean_se mean_ci_lower mean_ci_upper obs_ci_lower obs_ci_upper
0 48.143    8.253      31.541      64.746      10.449      85.837
```

La colonne `mean` indique la valeur prédictive définie par 10.21. La colonne `mean_se` donne l'écart-type de l'erreur due à l'estimation des paramètres du modèle :

$$\hat{\sigma} \sqrt{X'_{t+h}(X'X)^{-1}X_{t+h}} \quad (10.25)$$

Les colonnes `mean_ci_lower` et `mean_ci_upper` sont les bornes inférieures et supérieures de l'intervalle de confiance pour l'estimation de la moyenne (équation 10.24).

Enfin, les colonnes `obs_ci_lower` et `obs_ci_upper` sont les bornes inférieures et supérieures de l'intervalle de confiance de prédiction.

```
# Valeur prédictive
valpred = prediction.predicted_mean
print('Valeur prédictive : %.4f%(valpred)

# Erreur de prévision
error_var_pred = prediction.var_pred_mean
print('Ecart-type : %.4f'%(np.sqrt(error_var_pred)))

# Intervalle de confiance de l'estimation de la moyenne
ic_mean_pred = prediction.conf_int(alpha=0.05)
print('Intervalle de confiance : ', ic_mean_pred)

# Ecart-type de l'estimation de la moyenne
se_mean = prediction.se_mean
print("Ecart-type de l'estimation de la moyenne : %.4f"%(se_mean))

# Ecart-type de la prédiction
se_obs = prediction.se_obs
print("Ecart-type de la prédiction : %.4f"%(se_obs))
```

```
Valeur prédictive : 48.1432
Ecart-type : 8.2529
Intervalle de confiance : [[31.54052304 64.74595471]]
Ecart-type de l'estimation de la moyenne : 8.2529
Ecart-type de la prédiction : 18.7370
```

En particulier, la dernière valeur correspond au calcul de l'équation

$$\hat{\sigma} \sqrt{1 + X'_{t+h}(X'X)^{-1}X_{t+h}} \quad (10.26)$$

On vérifie :

```
from scipy import stats

# qpred contient la borne inférieure et supérieure
qpred = stats.t.interval(alpha=0.95, df=model.df_resid)
ic_pred = prediction.predicted_mean + qpred*prediction.se_obs
print('Intervalle de confiance :', ic_pred)

Intervalle de confiance : [10.4491871 85.83729064]
```

10.5 Influence et diagnostics

Le critère des moindres carrés, comme celui de la vraisemblance, appliqué à une distribution gaussienne douteuse, est très sensible à des observations atypiques, hors « norme » (outliers), c'est-à-dire qui présentent des valeurs trop singulières. Un diagnostic doit être établi dans le cadre spécifique du modèle recherché afin d'identifier les observations *influentes*, c'est-à-dire celles dont une faible variation du $p+1$ -uplet $(x_{1i}, \dots, x_{pi}, y_i)$ introduisent une modification importante des caractéristiques du modèle.

Ces observations repérées, il n'y a pas de remède universel : supprimer une valeur aberrante, corriger une erreur de mesure, construire une estimation robuste (en norme \mathcal{L}_1), ne rien faire..., cela dépend du contexte et doit être négocié avec le commanditaire de l'étude.

```
# Chargement des données
import pandas as pd

donnee = pd.read_csv('ozone_long.txt', sep=';')
print(donnee.head(10))
```

	O3	T6	T12	Ne12	Ne15	Vx	O3v
0	47.6	10.1	13.3	8	8	-3.4641	62.2
1	56.2	9.5	13.8	7	0	0.0000	47.6
2	61.8	3.6	16.8	2	2	-0.3420	56.2
3	50.8	9.5	11.4	7	7	-0.5209	61.8
4	59.8	9.8	13.8	8	8	-0.9848	50.8
5	53.4	8.9	12.6	8	8	-1.8794	59.8
6	49.4	11.2	13.3	8	7	-0.1736	53.4
7	82.8	10.0	15.6	1	3	6.0000	49.4
8	76.8	3.7	12.3	3	5	4.0000	82.8
9	92.6	3.1	14.1	4	4	1.7321	76.8

TABLE 10.3 – 50 données de température à 12 h et teneur en ozone complétées

On ajuste le modèle suivant :

$$O3 = \beta_0 \beta_1 T6 + \beta_2 T12 + \beta_3 Ne12 + \beta_4 Ne15 + \beta_5 Vx + \varepsilon \quad (10.27)$$

```
# Estimation des paramètres
from statsmodels.api import OLS
```

```
model = OLS.from_formula(formula = 'O3~T6+T12+Ne12+Ne15+Vx+O3v',
                         data=donnee).fit()
```

```
print(model.summary2())
```

Results: Ordinary least squares

```
=====
Model:                 OLS                   Adj. R-squared:   0.667
Dependent Variable:  O3                    AIC:           8158.0123
Date:                  2022-02-09 20:48 BIC:           8192.4639
No. Observations:    1014                 Log-Likelihood: -4072.0
Df Model:                6                  F-statistic:     339.4
Df Residuals:          1007                Prob (F-statistic): 7.82e-238
R-squared:               0.669                Scale:           181.38
=====
```

	Coef.	Std. Err.	t	P> t	[0.025	0.975]
Intercept	34.0895	3.1749	10.7371	0.0000	27.8593	40.3198
T6	-1.5935	0.1842	-8.6491	0.0000	-1.9550	-1.2319
T12	2.0775	0.1717	12.1011	0.0000	1.7406	2.4144
Ne12	-0.9538	0.3234	-2.9497	0.0033	-1.5884	-0.3193
Ne15	-0.7738	0.2780	-2.7837	0.0055	-1.3193	-0.2283
Vx	0.7108	0.1398	5.0842	0.0000	0.4364	0.9851
O3v	0.4548	0.0205	22.2005	0.0000	0.4146	0.4949
<hr/>						
Omnibus:	3.360		Durbin-Watson:		1.825	
Prob(Omnibus):	0.186		Jarque-Bera (JB):		3.526	
Skew:	-0.067		Prob(JB):		0.172	
Kurtosis:	3.256		Condition No.:		676	
<hr/>						

```
# Influence du model
from statsmodels.stats.outliers_influence import OLSInfluence
influ = OLSInfluence(model)
```

10.5.1 Résidus et PRESS

Différents types de résidus sont définis afin d'affiner leurs propriétés.

i) Les résidus

Ils correspondent à la différence entre les valeur observé et la va leur ajustée de la cible.

$$\hat{\varepsilon}_i = y_i - \hat{y}_i \quad (10.28)$$

```
# Residus
import numpy as np
import matplotlib.pyplot as plt

n = len(donnee)
resid = model.resid

plt.figure(figsize=(14,8))
plt.scatter(np.arange(n),resid)
plt.xlabel('individu')
plt.ylabel('résidus')
plt.title('Représentation des résidus')
plt.show()
```

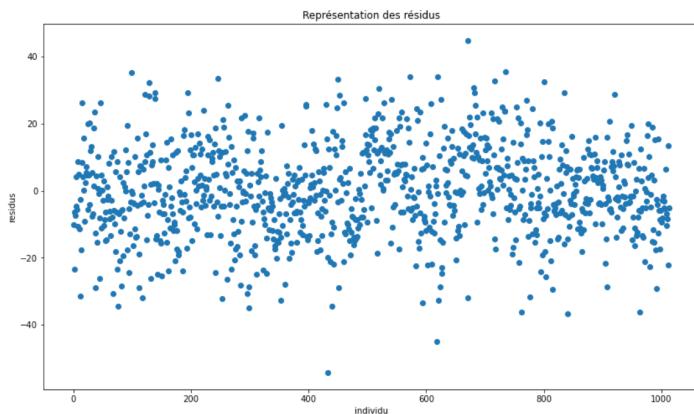


FIGURE 10.4 – Nuage de points des résidus

ii) Les résidus_(i)

on a :

$$\hat{\varepsilon}_{(i)i} = y_i - \hat{y}_{(i)i} = \frac{\hat{\varepsilon}}{1 - h_{ii}} \quad (10.29)$$

où $\hat{y}_{(i)i}$ est la prévision de y_i , calculée sans la i -ème observation.

```
# résidus press
resid_press = influ.resid_press

plt.figure(figsize=(14,8))
plt.scatter(np.arange(n),resid_press)
plt.xlabel('individu')
plt.ylabel('résidus')
plt.title('Représentation des résidus PRESS')
plt.show()
```

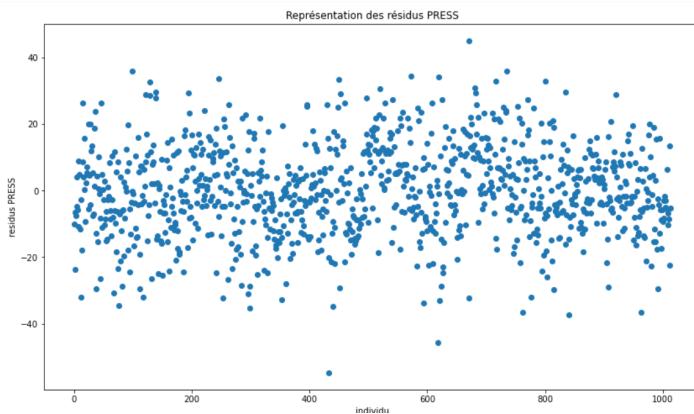


FIGURE 10.5 – Résidus PRESS

Ce type de résidu conduit à la définition du **PRESS**(predicted residual sum of squares) dit de Allen :

$$\text{PRESS} = \frac{1}{n} \sum_{i=1}^n \hat{\varepsilon}_{(i)i}^2 = \frac{1}{n} \sum_{i=1}^n \left(\frac{\hat{\varepsilon}}{1 - h_{ii}} \right)^2 \quad (10.30)$$

C'est une estimation sans biais de la qualité de prévision d'un modèle car une même observation n'est pas utilisée, à la fois, pour estimer le modèle et l'erreur de prévision. Le PRESS est très utile pour comparer les qualités prédictives de plusieurs modèles.

```
# Valeur du PRESS
press = influ.ess_press/n
print('PRESS : %.4f'%(press))
```

PRESS : 182.8723

iii) Les résidus standardisées

Même si l'hypothèse d'homoscédasticité est vérifiée, ceux-ci n'ont pas la même variance : $\mathbb{E}(e_i) = 0$ et $V(e_i) = \sigma_\varepsilon^2(1 - h_{ii})$. Il est donc d'usage d'en calculer des variances standardisées afin de les rendre comparables :

$$r_i = \frac{\hat{\varepsilon}_i}{s\sqrt{1 - h_{ii}}} \quad (10.31)$$

iv) Résidus studentisés

La standardisation « interne » dépend de $\hat{\varepsilon}_i$ dans le calcul de $\hat{\sigma}_\varepsilon$ estimation de $V(\hat{\varepsilon}_i)$. Une estimation non biaisé de cette variance est basée sur :

$$\hat{\sigma}_{\varepsilon(i)}^2 = \frac{1}{n-3} \left[(n-p-1)\hat{\sigma}_\varepsilon^2 - \frac{\hat{\varepsilon}^2}{1-h_{ii}} \right] \quad (10.32)$$

qui ne tient compte de la i -ème observation. On défni alors les résidus studentisés par :

$$t_i = \frac{\hat{\varepsilon}_i}{\hat{\sigma}_{\varepsilon(i)}\sqrt{1 - h_{ii}}} \quad (10.33)$$

Sous hypothèse de normalité, on montre que ces résidus suivent une loi de Student à $(n-3)$ degrés de liberté.

```
# Résidus studentisés
from statsmodels.nonparametric.smoothers_lowess import lowess

x = np.linspace(1,n,n)
y = np.repeat(2,n)
fig, axes = plt.subplots(figsize=(14,8))
axes.scatter(x,influ.resid_studentized_external)
axes.plot(x,y,linestyle='dashed',color='green')
axes.plot(x,-y,linestyle='dashed',color='green')
filtered = lowess(influ.resid_studentized_external,x)
axes.plot(filtered[:,0],filtered[:,1],color='black')
axes.set_xlabel("Individu")
axes.set_ylabel("Résidu studentisé par VC")
axes.set_title("Représentation des résidus en fonction des individus")
plt.show()
```

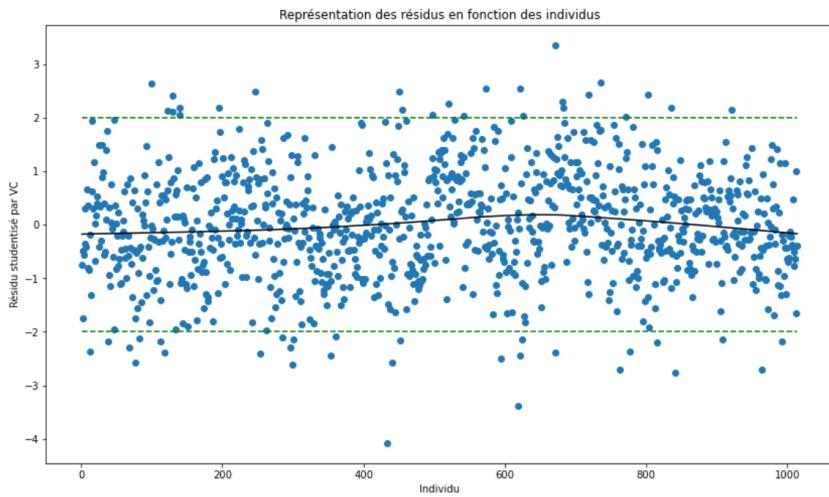


FIGURE 10.6 – Résidus studentisés

Il est normal que certaines valeurs dépassent ce seuil ponctuellement, en fait, en moyenne 5% des individus sont à l'extérieur de la bande matérialisée par les lignes en pointillés vertes. La courbe en noir est un lissage des valeurs des résidus studentisés. On observe une légère courbure autour du jour 600 mais rien de dramatique.

```
import statsmodels.api as sm

fig, ax = plt.subplots(figsize=(14, 8))
fig = sm.graphics.influence_plot(model, size=7, ax=ax)
```

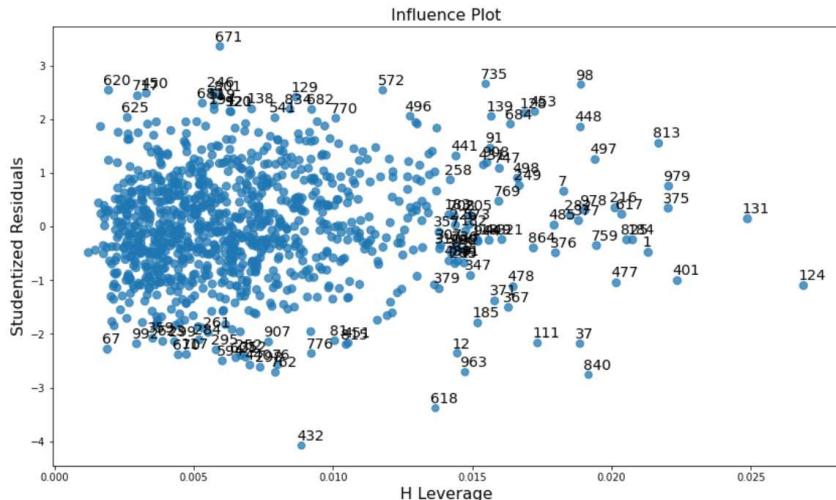


FIGURE 10.7 – Influence plot

Les observations suspectes correspondent au jour 432, 618 et 671 et il faut regarder plus précisément ces observations pour comprendre pourquoi ces points sont aberrants. Une valeur aberrante n'est pas nécessairement une observation que l'on doit écarter de l'échantillon. C'est surtout le cas s'il s'agit d'un point levier.

Valeurs aberrantes

```
print(donnee.iloc[np.where(np.abs(influ.resid_studentized_external) > 3)])
```

	03	T6	T12	Ne12	Ne15	Vx	03v
432	40.2	15.1	21.1	8	7	-4.3301	124.8
618	76.2	16.1	26.7	6	3	-0.6840	144.8
671	88.8	9.9	10.9	8	8	-3.0000	41.6

10.5.2 Analyse de la normalité

L'hypothèse de normalité est examiné d'abord graphiquement en affichant le graphique quantile-quantile aussi appelé **QQ-plot**. Le **QQ-plot** est le nuage de points $(p_i, \varepsilon_{(i)})$ où :

- $\varepsilon_{(i)}$ le i^{e} plus petit élément de n réalisations $\varepsilon_1, \dots, \varepsilon_n$ d'une variable aléatoire ε et
- p_i vaut $\Phi^{-1} \left(\frac{i}{n+1} \right)$ où Φ est la fonction de répartition d'une $\mathcal{N}(0,1)$.

Si $\varepsilon \sim \mathcal{N}(0,1)$, le graphe obtenu est proche de la droite d'équation $y = x$. Cette analyse graphique peut être formalisée à l'aide du test de Shapiro-Wilk.

Ci-dessous est représenté le graphe quantile-quantile des quantiles empiriques en ordonné par rapport aux quantiles théorique d'une loi $\mathcal{N}(0,1)$. Le choix de la version studentisé des résidus est nécessaire pour que chaque valeur soit normalisée et que la comparaison avec les quantiles empiriques fassent sens.

```
## QQ-plot
fig, axes = plt.subplots(figsize=(14, 8))
fig = sm.qqplot(influ.resid_studentized_external, line='45', ax=axes)
plt.title('QQ-plot')
plt.show()
```

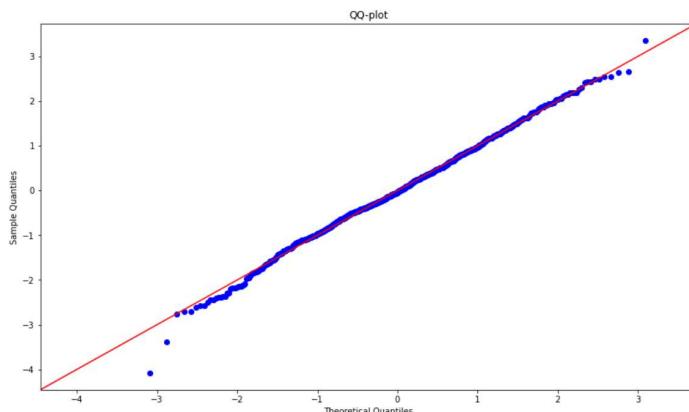


FIGURE 10.8 – QQ-plot des résidus studentisés

On retrouve ici les 3 observations aberrantes de la section précédente : ce sont les 3 points les plus éloignés de la première bissectrice. Cependant, l'hypothèse de normalité reste très raisonnable.

```
# Test de Shapiro
from scipy import stats
shapiro_test = stats.shapiro(influ.resid_studentized_external)
shapiro_test

ShapiroResult(statistic=0.9976094365119934, pvalue=0.1471538096666336)
```

10.5.3 Analyse de l'homoscédasticité

Dans le modèle linéaire (possiblement en absence de normalité), nous avons $\text{Cov}(\hat{Y}, \hat{\epsilon}) = 0$ (qui équivaut à l'indépendance dans le modèle gaussien) de sorte que le nuage de points (\hat{y}_i, t_i^*) ne devrait pas exhiber de structure particulière. Un nuage de points en forme de trompette révèle souvent une hétéroscédasticité alors qu'une forme courbe de type banane indique plutôt l'existence d'une non-linéarité.

À gauche sont représentés les résidus studentisés contre les valeurs ajustées de la variable réponse. On retrouve ces 3 observations un peu en dehors de la norme. Cependant, on observe pas de structure particulière. On observe un phénomène très similaire sur le graphique de droite représentant la valeur absolue des résidus studentisés en fonction de la valeur ajustée de la variable réponse. Le lissage permet de mieux objectiver le fait qu'il n'y ait pas de structure.

```
# Analyse de l'hétéroscédasticité
fig, (axes1, axes2) = plt.subplots(1, 2, figsize=(14, 8))
fig.subplots_adjust(wspace=0.5)
filtered = lowess(abs(influ.resid_studentized_external), model.fittedvalues)
axes1.scatter(model.fittedvalues, influ.resid_studentized_external)
axes2.scatter(model.fittedvalues, np.abs(influ.resid_studentized_external))
axes2.plot(filtered[:, 0], filtered[:, 1], color = 'black')

axes1.set_xlabel("Valeur ajustée Y")
axes1.set_ylabel("Résidu studentisé par VC")
axes1.set_title("Résidus studentisés \n versus valeurs ajustées")

axes2.set_xlabel("Valeur ajustée Y")
axes2.set_ylabel("Résidus studentisés en valeur abs.")
axes2.set_title("Résidus studentisés versus \n valeurs ajustées en valeur abs.")
plt.show()
```

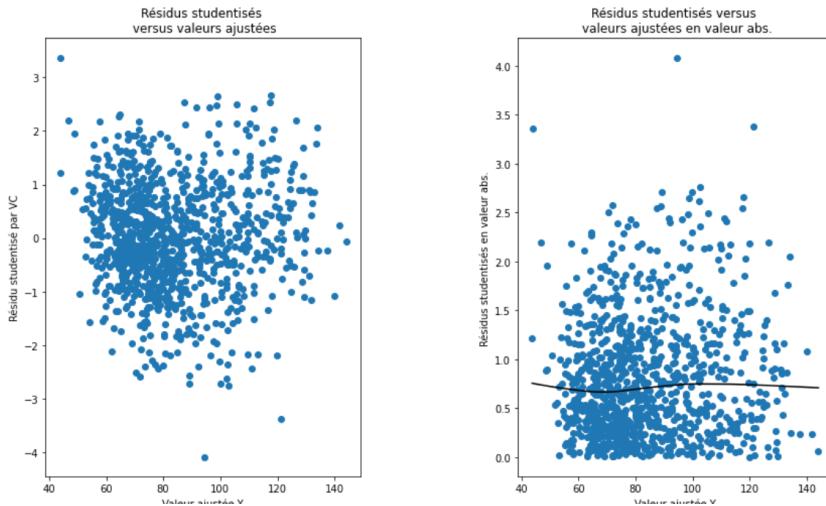


FIGURE 10.9 – Analyse de l'hétéroscédasticité

10.5.4 Analyse de la structure des résidus

Il a déjà été procédé à une première analyse de la structure des résidus dans la section précédente. On peut affiner cette analyse en représentant le graphique des résidus studentisés en fonction de chaque variables explicatives.

```
# analyse de la structure des résidus (résidus versus variables explicatives)
fig, ((ax1,ax2),(ax3,ax4),(ax5,ax6)) = plt.subplots(3,2,figsize=(14,8))
fig.subplots_adjust(wspace=1,hspace=1)
# résidus vs T6
filtered_T6 = lowess(influ.resid_studentized_external,donnee['T6'])
ax1.scatter(donnee['T6'],influ.resid_studentized_external)
ax1.plot(filtered_T6[:,0],filtered_T6[:,1],color = 'black')
ax1.set_xlabel("T6")
ax1.set_ylabel("Résidus stud.")
ax1.set_title("Résidus versus T6")
# résidus vs T12
filtered_T12 = lowess(influ.resid_studentized_external,donnee['T12'])
ax2.scatter(donnee['T12'],influ.resid_studentized_external)
ax2.plot(filtered_T12[:,0],filtered_T12[:,1],color = 'black')
ax2.set_xlabel("T12")
ax2.set_ylabel("Résidus stud.")
ax2.set_title("Résidus versus T12")
# résidus versus Ne12
filtered_Ne12 = lowess(influ.resid_studentized_external,donnee['Ne12'])
ax3.scatter(donnee['Ne12'],influ.resid_studentized_external)
ax3.plot(filtered_Ne12[:,0],filtered_Ne12[:,1],color = 'black')
ax3.set_xlabel("Ne12")
ax3.set_ylabel("Résidus stud.")
ax3.set_title("Résidus versus Ne12")
```

```

# residus versus Ne15
filtered_Ne15 = lowess(influ.resid_studentized_external,donnee['Ne15'])
ax4.scatter(donnee['Ne15'],influ.resid_studentized_external)
ax4.plot(filtered_Ne15[:,0],filtered_Ne15[:,1],color = 'black')
ax4.set_xlabel("Ne15")
ax4.set_ylabel("Résidus stud.")
ax4.set_title("Résidus versus Ne15")
# residus versus Vx
filtered_Vx = lowess(influ.resid_studentized_external,donnee['Vx'])
ax5.scatter(donnee['Vx'],influ.resid_studentized_external)
ax5.plot(filtered_Vx[:,0],filtered_Vx[:,1],color = 'black')
ax5.set_xlabel("Vx")
ax5.set_ylabel("Résidus stud.")
ax5.set_title("Résidus versus Vx")
# residus versus O3v
filtered_O3v = lowess(influ.resid_studentized_external,donnee['O3v'])
ax6.scatter(donnee['O3v'],influ.resid_studentized_external)
ax6.plot(filtered_O3v[:,0],filtered_O3v[:,1],color = 'black')
ax6.set_xlabel("O3v")
ax6.set_ylabel("Résidus stud.")
ax6.set_title("Résidus versus O3v")
plt.show()

```

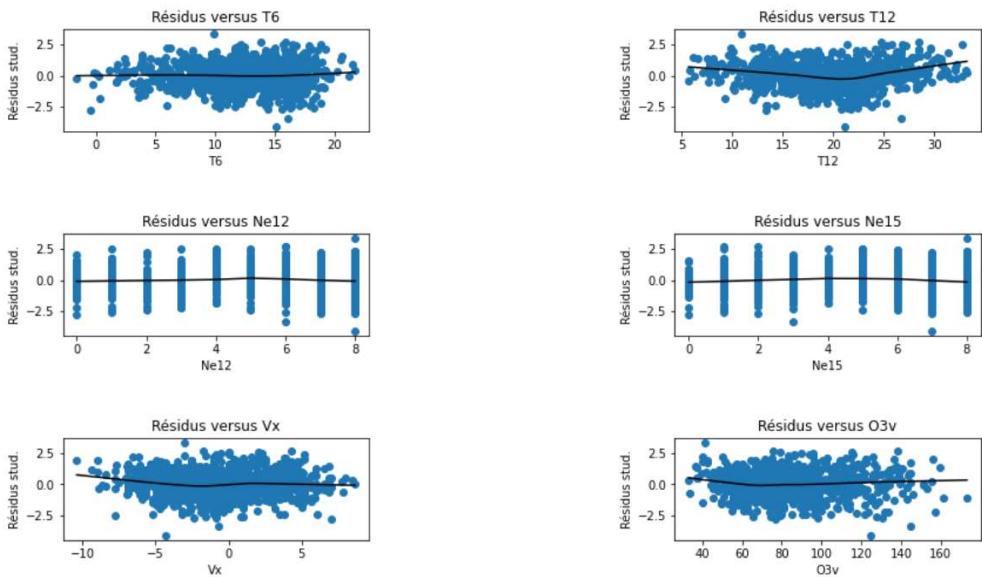


FIGURE 10.10 – Analyse de la structure des résidus

Les graphiques pour les variables T6, Ne12, Ne15 n'exhibe pas de structure particulière si bien que l'on peut supposer que l'hypothèse linéaire en ces variables est raisonnable ainsi que l'hypothèse d'homoscédasticité.

En revanche pour les variables Vx, O3v et plus particulièrement T12 on observe un changement de régime matérialisé par une brisure de la courbe de lissage. Cela peut être dû à une hétérosé-dasticité inhérente au phénomène physique ou à une erreur de modèle. Typiquement, on pourrait tenter de décomposer T12 en deux variables qui modéliseraient le lien avec la température à midi dans chacun des régime inférieure ou supérieure à 20°C. Ceci peut-être fait à l'aide d'indicatrice.

10.6 Autres mesures de diagnostiques

10.6.1 Effet levier et observations influentes

Une autre indication est donnée par l'éloignement des x_{ji} par rapport à leur moyenne \bar{x}_j . En effet, écrivons le vecteur prédicteur \hat{Y} comme combinaison linéaires des observations :

$$\hat{y} = X\hat{\beta} = X(X'X)^{-1}X'Y = Hy, \quad \text{avec } H = X(X'X)^{-1}X' \quad (10.34)$$

H est une matrice de format $n \times n$ et ses éléments diagonaux $h_{ii}(i = 1, \dots, n)$ mesurent ainsi l'impact ou l'importance du rôle que joue y_i dans l'estimation de \hat{y}_i .

```
# Élément diagonaux de H
hii = influ.hat_diag_factor
print(hii)
```

```
[0.00327464 0.02177701 0.0117276 ... 0.00500519 0.00488349 0.00484419]
```

Nous pouvons écrire :

$$\hat{y}_i = \sum_{j=1}^n h_{ij}y_j = h_{ii}y_i + \sum_{j \neq i} h_{ij}y_j \quad (10.35)$$

Ainsi, h_{ii} représente le poids de l'observation i sur sa propre estimation. La matrice H et ses coefficients $(h_{ij})_{i,j=1,\dots,n}$ vérifient :

1. $\text{Trace}(H) = \sum_{i=1}^n h_{ii} = p$, la moyenne des h_{ii} vaut donc p/n . Ainsi, si h_{ii} est « grand » y_i influe fortement sur \hat{y}_i .
2. $\text{Trace}(H) = \text{Trace}(HH) = \sum_{i,j=1}^n h_{ij}^2 = p$
3. $\forall i = 1, \dots, n \ h_{ii} \in [0,1]$
4. $\forall i, j = 1, \dots, n \ h_{ij} \in \left[\frac{1}{2}, \frac{1}{2}\right]$
5. Si $h_{ii} = 1$ alors $h_{ij} = 0 \ \forall j \neq i$
6. Si $h_{ii} = 0$ alors $h_{ij} = 0 \ \forall j \neq i$

Nous avons les cas extrêmes suivants :

- Si $h_{ii} = 1$, \hat{y}_i est entièrement déterminée par y_i car $h_{ij} = 0 \ \forall j \neq i$;

- Si $h_{ii} = 0$, \hat{y}_i n'a pas d'influence sur \hat{y}_i (qui vaut alors 0).

Un point est un point levier si les valeurs h_{ii} de H dépassent les valeurs suivantes :

- $h_{ii} > 2p/n$ selon Hoaglin et Welsh (1978)
- $h_{ii} > 3p/n$ pour $p > 6$ et $n - p - 1 > 12$ selon Velleman et Welsh (1981)
- $h_{ii} > 1/2$ selon Huber (1981)

Cette notion de levier h_{ii} correspond à l'éloignement du centre de gravité de la i^e ligne X : plus le point est éloigné, plus la valeur des h_{ii} augmente. Un point levier n'est pas un point aberrant car il se situe dans le prolongement de la droite de régression et donc son résidu est faible.

```
# Points leviers
yhw = np.repeat(2*7/n,n) # Hoaglin-Welsh
yvw = np.repeat(3*7/n,n) # Velleman-Welsh
fig, axes = plt.subplots(figsize=(14,8))
axes.scatter(x,hii)
axes.plot(x,yhw,linestyle='dashed',color='green',label='Hoaglin-Welsh')
axes.plot(x,yvw,linestyle='dashed',color='blue',label='Velleman-Welsh')
axes.set_xlabel("Individu")
axes.set_ylabel("hii")
axes.set_title("Points leviers")
axes.legend()
plt.show()
```

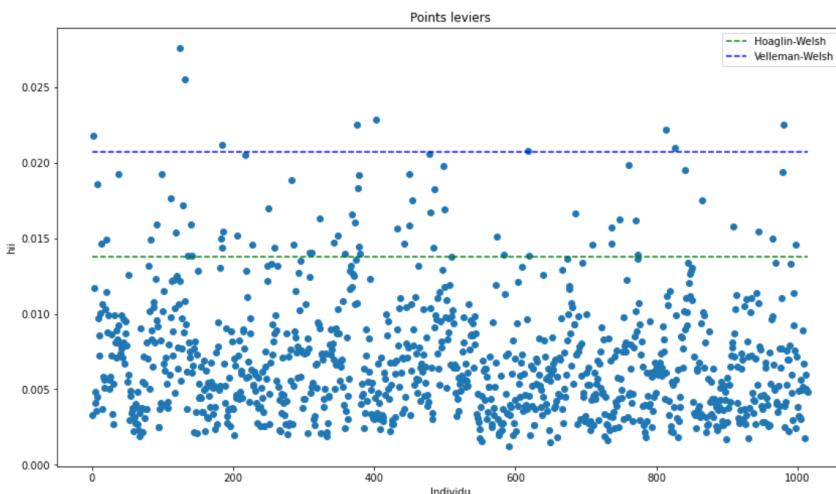


FIGURE 10.11 – Points leviers

10.6.2 Distance de Cook

La distance de Cook mesure l'influence de l'observation i sur l'estimation du paramètre β . La distance de Cook associée à l'observation i est définie par :

$$C_i = \frac{1}{(p+1)\hat{\sigma}^2} \left(\hat{\beta}_{(i)} - \hat{\beta}' \right) \left(X'X \right) \left(\hat{\beta}_{(i)} - \hat{\beta}' \right) = \frac{\left(\hat{y}_i - x_i' \hat{\beta}_{(i)} \right)^2}{(p+1)\hat{\sigma}^2} \quad (10.36)$$

Il est cependant possible de la réexprimer de manière plus concise et plus simple à calculer comme :

$$C_i = \frac{1}{p+1} \frac{h_{ii}}{1-h_{ii}} = \frac{h_{ii}}{(p+1)(1-h_{ii})^2} \frac{\hat{\varepsilon}_i^2}{\hat{\sigma}^2} \quad (10.37)$$

Il est important de noter que la loi de C_i n'est pas une loi de Fisher car $\hat{\beta}_{(i)}$ n'est pas indépendant de $\hat{\beta}$ et $\hat{\sigma}^2$. Pour autant, si $\hat{\beta}_{(i)}$ est remplacée par le vrai paramètre (inconnu) $\beta_0 \in \mathbb{R}^p$, la variable aléatoire

$$\frac{1}{(p+1)\hat{\sigma}^2} \left(\hat{\beta}_0 - \hat{\beta}' \right) \left(X'X \right) \left(\hat{\beta}_0 - \hat{\beta}' \right) \quad (10.38)$$

suit une loi de Fisher $f_{p+1,n-p-1}$. Comme précédemment, une observation i dépassant le seuil fixé doit amener à se poser des questions de conserver ou non cet individu dans l'échantillon. Les mêmes précautions doivent cependant être observées que pour l'écart des observations aberrantes.

```
#Distance de Cook
yok = np.repeat(0.1,n)
cd, pval = influ.cooks_distance
fig, axes = plt.subplots(figsize=(14,8))
axes.scatter(x,cd)
axes.plot(x,yok,linestyle='dashed',color='green')
axes.set_xlabel("Individu")
axes.set_ylabel("distance de Cooks")
axes.set_title("Distance de Cooks")
plt.show()
```

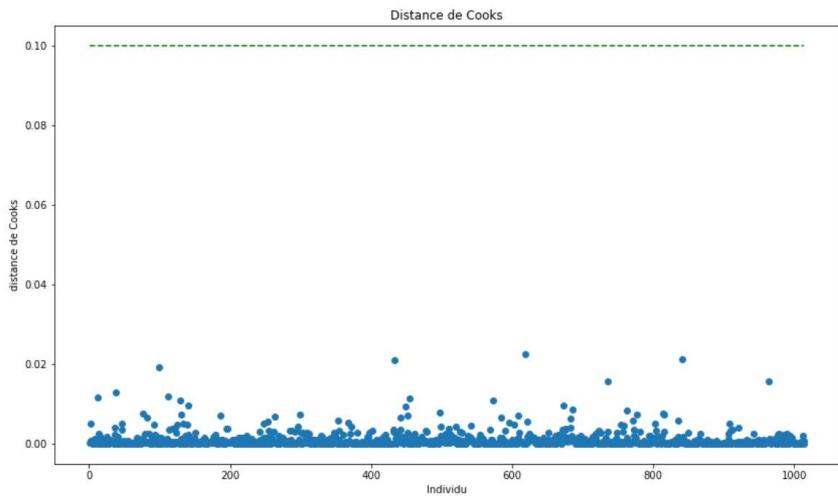


FIGURE 10.12 – Distance de Cook

Une autre mesure d'influence est fréquemment utilisée, il s'agit de l'écart de Welsh-Kuh. Cette mesure est souvent appelée DFFITS dans les logiciels statistiques. L'écart de Welsh-Kuh associé à l'observation i est défini par :

$$WK_i = |t_i^*| \sqrt{\frac{h_{ii}}{1 - h_{ii}}} \quad (10.39)$$

où t_i^* est le résidu studentisé obtenu par validation croisée. L'écart de Welsh-Kuh, au facteur $1/(p+1)$ près, est la distance de Cook dans laquelle l'estimation de $\hat{\sigma}^2$ se fait à l'aide $\hat{\sigma}_{(i)}$. Le seuil recommandé pour WK_i est $2\sqrt{\frac{p+1}{n-p-1}}$.

```
# Dffits
obs, dffits = influ.dffits
print(dffits)

0.16617283842071437

# DFFITS
yok = np.repeat(dffits,n)
yok_r = np.repeat(1.5*dffits,n)
fig, axes = plt.subplots(figsize=(14,8))
axes.scatter(x,abs(obs))
axes.plot(x,yok,linestyle='dashed',color='green')
axes.plot(x,yok_r,linestyle='dashed',color='green')
axes.set_xlabel("Individu")
axes.set_ylabel("Écart de Welsh-Kuh")
axes.set_title("Mesure DFFITS")
plt.show()
```



FIGURE 10.13 – Mesures DFFITS

10.6.3 Sélection des variables

La sélection de modèle peut être vue comme la recherche du modèle optimal au sens d'un critère choisi, parmi tous les modèles possibles. Cela peut être vu comme une optimisation d'une fonction objectif. Nous présentons ici la recherche pas à pas.

1) Méthode ascendante

Appelée *forward selection*, son principe est le suivant : A chaque pas, une variable est ajoutée au modèle.

- Si la méthode ascendante utilise un test F, nous rajoutons la variable X_i dont la probabilité critique (*fp-value*) associée à la statistique partielle de test de Fisher qui compare les 2 modèles est minimale. Nous nous arrêtons lorsque toutes les variables sont intégrées ou lorsque la probabilité critique est plus grande qu'une valeur seuil.
- Si la méthode ascendante utilise un critère de choix, nous ajoutons la variable X_i dont l'ajout au modèle conduit à l'optimisation de la plus grande du critère de choix. Nous nous arrêtons lorsque toutes les variables sont intégrées ou lorsqu'aucune variable ne permet l'optimisation du critère de choix.

Contrairement à R, la librairie **statsmodels** n'a pas de procédure toute faite pour la sélection de variables. On définit donc une fonction pour l'implémenter. Voici un code pour la procédure **forward** basée sur le R^2 ajusté (équivalent à la minimisation de la variance). Le lecteur peut écrire une fonction équivalente pour les C_p de Mallow, l'AIC et le BIC. Nous présentons ici un code modifié de celui téléchargé [ici](#).

```
# Forward selection
from statsmodels.api import OLS
```

```

def forward_selection(data, response, verbose = True):
    remaining = set(data.columns)
    remaining.remove(response)
    selected = []
    current_score, best_new_score = 0.0, 0.0
    while remaining and current_score == best_new_score:
        scores_with_candidates = []
        for candidate in remaining:
            formula = "{} ~ {} + 1".format(response,
                                             ' + '.join(selected + [candidate]))
            score = OLS.from_formula(formula, data).fit().rsquared_adj
            scores_with_candidates.append((score, candidate))
        scores_with_candidates.sort()
        best_new_score, best_candidate = scores_with_candidates.pop()
        if current_score < best_new_score:
            remaining.remove(best_candidate)
            selected.append(best_candidate)
            current_score = best_new_score
        if verbose:
            print('Add {} with Adj. R-squared {:.6}'.format(best_candidate,
                                                               best_new_score))

    formula = "{} ~ {} + 1".format(response,
                                    ' + '.join(selected))
    model = OLS.from_formula(formula, data).fit()
    return model

# Application
foward_model = forward_selection(donnee, 'O3')
print(foward_model.model.formula)

Add O3v      with Adj. R-squared 0.472306
Add Ne12     with Adj. R-squared 0.598518
Add T12      with Adj. R-squared 0.62222
Add T6       with Adj. R-squared 0.655414
Add Vx       with Adj. R-squared 0.664948
Add Ne15     with Adj. R-squared 0.667177
O3 ~ O3v + Ne12 + T12 + T6 + Vx + Ne15 + 1

```

2) Méthode descendante

Appelée *backward selection*, à la première étape toutes les variables sont intégrées au modèle :

- Si la méthode descendante utilise un test F , nous éliminons ensuite la variable X_i dont la valeur p , associée à la statistique partielle de test de Fisher, est la plus grande. Nous nous arrêtons lorsque toutes les variables sont retirées du modèle ou lorsque la valeur p est plus petite qu'une valeur seuil.
- Si la méthode descendante utilise un critère de choix, nous retirons la variable X_i dont le retrait du modèle conduit à l'augmentation la plus grande du critère de choix. Nous nous arrêtons lorsque toutes les variables sont retirées ou lorsque qu'aucune variable ne permet l'augmentation du critère de choix.

3) Méthode progressive

Appelée *stepwise selection*, le principe est le même que pour la méthode ascendante, sauf que l'on peut éliminer des variables déjà introduites. En effet, il peut arriver que des variables introduites en début ne soient plus significatives après introduction de nouvelles variables. Remarquons qu'en général la variable « constante », constituée de 1 et associée au coefficient « moyenne générale », est en général traitée à part et elle est toujours présente dans le modèle.

On retrouve [ici](#) des codes pour les méthodes forward et backward basées sur la p-value.

CHAPITRE 11

LE MODÈLE LINÉAIRE GÉNÉRAL

Sommaire

11.1 Introduction	266
11.2 Modélisation du problème	268
11.3 Analyse de la covariance	273

11.1 Introduction

Le modèle linéaire général ou modèle de régression linéaire sur variables catégorielles est une extension de la régression multiple au cas où il y a des variables explicatives qualitatives. Dans ce cas, pouvons-nous appliquer la méthode des moindres carrés vue dans les chapitres précédents ?

Nous reprenons l'exemple d'ozone où nous souhaitons expliquer la concentration en ozone O_3 en fonction de la température T12 et de la direction du vent `vent`, variable qualitative prenant 4 modalités : NORD, SUD, EST et OUEST.

```
# Chargement des données
import pandas as pd

donnee = pd.read_csv('ozone.txt',sep=';',index_col=0)
print(donnee[['O3','T12','vent']].head(10))
```

Date	O3	T12	vent
1996-04-22	63.6	13.4	EST
1996-04-29	89.6	15.0	NORD
1996-05-06	79.0	7.9	EST
1996-05-14	81.2	13.1	NORD
1996-05-21	88.0	14.1	OUEST
1996-05-28	68.4	16.7	SUD
1996-06-05	139.0	26.8	EST
1996-06-12	78.2	18.4	NORD
1996-06-19	113.8	27.2	SUD
1996-06-27	41.8	20.6	OUEST

TABLE 11.1 – Tableau de données brutes

Nous avons 2 variables explicatives : température à 12 h T12 et direction du vent **vent**. Pouvons-nous effectuer une régression multiple ? Comment utiliser la variable **vent** ? Dans cet exemple simple, nous pouvons représenter les données avec en abscisse la température, en ordonnée la concentration en ozone et couleur la direction du vent.

```
# Graphique
import numpy as np
import matplotlib.pyplot as plt

data = donnee[['T12','O3']]
data.index = [(x+1) for x in range(len(donnee))]
fig, axes = plt.subplots(figsize = (12,8));
axes.grid()
axes.set_title("Nuage de points : O3 en fonction de T12 et vent")
axes.set_xlabel('T12')
axes.set_ylabel('O3')
cdict = {'NORD': 'green', 'SUD': 'blue', 'EST':'red', 'OUEST':'black'}
cmarker = {'NORD': '^', 'SUD': 'v', 'EST':">>','OUEST':<"}
for group in np.unique(donnee['vent']):
    idx = np.where(donnee['vent']==group)
    axes.scatter(data.iloc[:,0].values[idx[0]],
                 data.iloc[:,1].values[idx[0]],
                 label = group, c = cdict[group],
                 marker = cmarker[group], s=100)
    for i in idx[0]:
        axes.text(data.iloc[:,0].values[i],data.iloc[:,1].values[i],
                  s=data.index[i],c = cdict[group], fontsize = 13)
axes.legend()
plt.show()
```

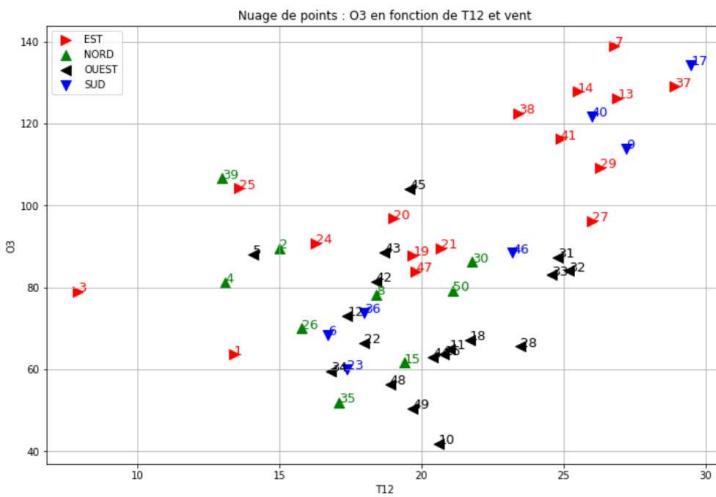


FIGURE 11.1 – Nuage de points

La direction du vent pourrait avoir un effet sur la concentration en ozone, mais cela est difficile à observer.

11.2 Modélisation du problème

11.2.1 Modélisation

En présence des variables qualitatives dans le modèle, il faut procéder à deux opérations.

- Chaque variable qualitative est remplacée par les variables indicatrices de ses modalités. Par exemple, la variable `vent` est remplacée par les variables `EST`, `OUEST`, `NORD`, `SUD`.
- On peut remarquer que la somme des variables indicatrices associées à une variable qualitative est égale à 1. Ceci provoque une redondance entre les variables explicatives qu'il faut prendre en compte. Pour résoudre ce problème, on doit supprimer une variable indicatrice de modalité par variable qualitative. Le choix de la modalité à supprimée est arbitraire et n'a aucune influence sur le calcul des valeurs prédictes. En général, on supprime la variable indicatrice de la dernière modalité de chaque variable qualitative, ou bien la première. Les modèles supprimés servent de référence.

Ici, nous supprimons la modalité `EST`. Le modèle étudié s'écrit :

$$O3 = \beta_0 + \beta_1 T12 + \beta_2 I_{\{vent=NORD\}} + \beta_3 I_{\{vent=OUEST\}} + \beta_4 I_{\{vent=SUD\}} + \varepsilon \quad (11.1)$$

```
# Estimation des paramètres
from statsmodels.api import OLS
```

```
modelcomplet = OLS.from_formula(formula='O3~T12+vent', data=donnee).fit()
print(modelcomplet.summary2())
```

```

Results: Ordinary least squares
=====
Model: OLS Adj. R-squared: 0.509
Dependent Variable: O3 AIC: 428.4127
Date: 2022-02-13 15:57 BIC: 437.9728
No. Observations: 50 Log-Likelihood: -209.21
Df Model: 4 F-statistic: 13.72
Df Residuals: 45 Prob (F-statistic): 2.17e-07
R-squared: 0.549 Scale: 280.27
-----
          Coef. Std.Err. t P>|t| [0.025 0.975]
-----
Intercept      52.3484 12.3370 4.2432 0.0001 27.5005 77.1964
vent[T.NORD]   -15.8292 7.3121 -2.1648 0.0357 -30.5564 -1.1019
vent[T.OUEST]  -29.9384 5.7761 -5.1831 0.0000 -41.5721 -18.3047
vent[T.SUD]    -12.8550 7.6239 -1.6861 0.0987 -28.2102 2.5003
T12            2.4300 0.5476 4.4377 0.0001 1.3271 3.5329
-----
Omnibus:      1.592 Durbin-Watson: 1.687
Prob(Omnibus): 0.451 Jarque-Bera (JB): 1.573
Skew:          0.381 Prob(JB): 0.455
Kurtosis:     2.583 Condition No.: 117
=====
```

Le modèle estimée s'écrit :

$$\hat{O3} = 52.35 + 2.43 \times T12 - 15.83 \times 1_{\{\text{vent}=NORD\}} - 29.94 \times 1_{\{\text{vent}=OUEST\}} - 12.86 \times 1_{\{\text{vent}=SUD\}} \quad (11.2)$$

En choisissant un risque $\alpha = 5\%$, les niveaux de signification donnés montrent que la variable T12 est significative, ainsi que les variables NORD et OUEST. La variable SUD n'est pas significative.

Prédiction

```

prediction = modelcomplet.get_prediction(exog=donnee[['T12','vent']])
frame = prediction.summary_frame(alpha=0.05)
print(frame.head().round(3))
```

	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
0	84.911	5.977	72.872	96.950	49.108	120.714
1	72.970	5.708	61.474	84.466	37.346	108.594
2	71.546	8.397	54.634	88.458	33.824	109.268
3	68.353	6.013	56.242	80.463	32.526	104.180
4	56.674	5.182	46.237	67.110	21.377	91.970

Le graphique de la concentration en ozone observée *versus* la concentration en ozone calculée est donné dans la figure 11.2.

Graphique

```

plt.figure(figsize=(12,8))
plt.title('Concentration ozone observé\nversus\nConcentration ozone calculé')
plt.scatter(frame['mean'], donnee['O3'])
```

```

plt.xlabel('Concentration ozone calculée')
plt.ylabel('Concentration ozone observée')
plt.show()

```

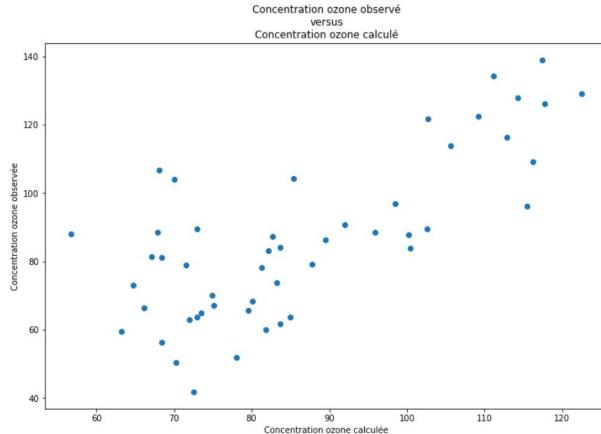


FIGURE 11.2 – Concentration en ozone observée versus calculée

11.2.2 Tests sur les variables explicatives

Nous allons tester la significativité de nos variables explicatives.

1) Tests sur les variables quantitatives

Il s'agit d'étudier sur le modèle (11.1) le test d'hypothèse :

$$\begin{cases} H_0 : \beta_1 = 0 \\ H_1 : \beta_1 \neq 0 \end{cases}$$

On utilise la statistique :

$$F = \frac{\text{SCE}(\text{modèle complet}) - \text{SCE}(\text{modèle complet sauf T12})}{\text{SCR}(\text{modèle complet})/(n - k - 1)} \quad (11.3)$$

On régresse O3 sur vent, c'est-à-dire le modèle suivant :

$$O3 = \alpha_0 + \alpha_1 1_{\{\text{vent=NORD}\}} + \alpha_2 1_{\{\text{vent=OUEST}\}} + \alpha_3 1_{\{\text{vent=SUD}\}} + \eta \quad (11.4)$$

```

# Modèle sans température T12
modelsanst12 = OLS.from_formula(formula='O3~vent', data=donnee).fit()
print(modelsanst12.summary2())

```

Results: Ordinary least squares

```

=====
Model:          OLS           Adj. R-squared:   0.310
Dependent Variable: O3            AIC:           444.5627
Date: 2022-02-13 16:07  BIC:           452.2108

```

No. Observations:	50	Log-Likelihood:	-218.28			
Df Model:	3	F-statistic:	8.338			
Df Residuals:	46	Prob (F-statistic):	0.000156			
R-squared:	0.352	Scale:	394.16			
<hr/>						
	Coef.	Std.Err.	t	P> t	[0.025	0.975]
Intercept	103.8500	4.9634	20.9233	0.0000	93.8593	113.8407
vent[T.NORD]	-25.5611	8.2723	-3.0900	0.0034	-42.2123	-8.9099
vent[T.OUEST]	-32.2722	6.8215	-4.7310	0.0000	-46.0032	-18.5413
vent[T.SUD]	-9.5071	8.9969	-1.0567	0.2962	-27.6169	8.6026
<hr/>						
Omnibus:	1.682	Durbin-Watson:	1.737			
Prob(Omnibus):	0.431	Jarque-Bera (JB):	1.184			
Skew:	0.083	Prob(JB):	0.553			
Kurtosis:	2.264	Condition No.:	5			
<hr/>						

Le modèle estimé s'écrit :

$$\hat{Y} = 103.85 - 25.56 \times 1_{\{\text{vent=NORD}\}} - 32.27 \times 1_{\{\text{vent=OUEST}\}} - 9.51 \times 1_{\{\text{vent=SUD}\}} \quad (11.5)$$

On calcule la statistique F :

```
# Test sur le coefficient de T12
import scipy.stats as stats

u1 = modelcomplet.mse_model - modelsanst12.mse_model
v1 = modelcomplet.mse_resid/modelcomplet.df_resid
fstats1 = u1/v1
print('f-statistic : %.4f'%(fstats1))

f-statistic : 89.6262
```

Cette statistique F est le carré de la statistique de t de Student à $n-k-1$ degrés de liberté. Ainsi, le niveau de significativité du F est exactement celui du t.

```
# Statistique t de Student
tstats = np.sqrt(fstats1)
pvalue1 = 2*(1-stats.t.cdf(np.abs(tstats), df =modelcomplet.df_resid))
print("""t-statistic : %.4f \npvalue : %.4f"""%(tstats,pvalue1))

t-statistic : 9.4671
pvalue : 0.0000
```

On ne peut rejeter l'hypothèse H_0 . La température à 12 h est significative.

2) Tests sur les variables qualitatives

On étudie sur le modèle (11.1) le test d'hypothèse :

$$\begin{cases} H_0 : \beta_2 = \beta_3 = \beta_4 = 0 \\ H_1 : \text{Au moins un de ces } \beta_j \neq 0 \end{cases}$$

On utilise la statistique :

$$F = \frac{1}{3} \frac{(\text{SCE(modèle complet)} - \text{SCE(modèle complet sauf vent)})}{\text{SCR(modèle complet)} / (n - k - 1)} \quad (11.6)$$

Le chiffre 3 au dénominateur correspond au nombre de paramètres à tester.

```
# Modèle sans vent
modelsansvent = OLS.from_formula(formula='O3~T12', data=donnee).fit()
print(modelsansvent.summary2())

Results: Ordinary least squares
=====
Model:                 OLS                   Adj. R-squared:      0.264
Dependent Variable:  O3                      AIC:             445.9136
Date:                2022-02-13 12:16   BIC:             449.7376
No. Observations:    50                    Log-Likelihood:   -220.96
Df Model:              1                     F-statistic:       18.58
Df Residuals:         48                     Prob (F-statistic): 8.04e-05
R-squared:            0.279                  Scale:             420.40
-----
          Coef.    Std.Err.      t     P>|t|    [0.025    0.975]
-----
Intercept    31.4150   13.0584   2.4057   0.0200   5.1592   57.6707
T12          2.7010    0.6266   4.3107   0.0001   1.4412   3.9609
-----
Omnibus:           2.252      Durbin-Watson:        1.461
Prob(Omnibus):    0.324      Jarque-Bera (JB):    1.348
Skew:               0.026      Prob(JB):           0.510
Kurtosis:            2.197      Condition No.:      94
=====
```

Le modèle estimé s'écrit :

$$\hat{O3} = 31.42 + 2.7 \times T12 \quad (11.7)$$

On calcule la statistique F :

```
# Test sur vent
u2 = modelcomplet.mse_model - modelsansvent.mse_model
v2 = modelcomplet.mse_resid/modelcomplet.df_resid
fstats2 = u2/v2
pvalue2 = stats.f.cdf(fstats2,3,modelcomplet.df_resid)
print('f-statistic : %.4f \npvalue : %.4f'%(fstats2,pvalue2))

f-statistic : -636.9506
pvalue : 0.0000
```

La direction du vent est donc globalement significative.

11.3 Analyse de la covariance

11.3.1 Position du problème

Dans l'approche par analyse de la covariance, la démarche la plus naturelle consiste à effectuer K^1 régressions différentes, une pour chaque catégorie. Cela donne en termes de modélisation :

$$03_i^j = \beta_0^j + \beta_1^j T12^j + \varepsilon_i^j, \quad i = 1, \dots, n_j \quad (11.8)$$

où n_j est le nombre total d'individus appartenant au groupe j ($j = \text{EST}, \text{NORD}, \text{OUEST}, \text{SUD}$). Trois cas de figures sont envisageables :

Cas 1 : La pente est identique d'une droite à l'autre

Dans ce cas, la température à 12 h intervient de la même façon d'une direction du vent à l'autre. Ce qui donne en terme de modélisation :

$$03_i^j = \beta_0^j + \beta_1 T12^j + \varepsilon_i^j, \quad i = 1, \dots, n_j \quad (11.9)$$

Graphiquement, on a :

```
# Graphique
plt.figure(figsize=(12,8))
plt.plot([0,10],[0.1,.5],linestyle='--',color = 'red')
plt.plot([0,10],[0.25,0.65],color = 'blue')
plt.plot([0,10],[0.4,0.8],linestyle='--',color = 'green')
plt.annotate('b0=0.1', xy =(10.1,.5),xytext =(10.1,.5))
plt.annotate('b0=0.25', xy =(10.1,.65),xytext =(10.1,.65))
plt.annotate('b0=0.4', xy =(10.1,.8),xytext =(10.1,.8))
plt.title('Pentes parallèles')
plt.ylabel('y')
plt.xlabel('x')
plt.ylim([0,1])
plt.xlim([0,11])
plt.show()
```

1. K est le nombre de modalités de la variable qualitative

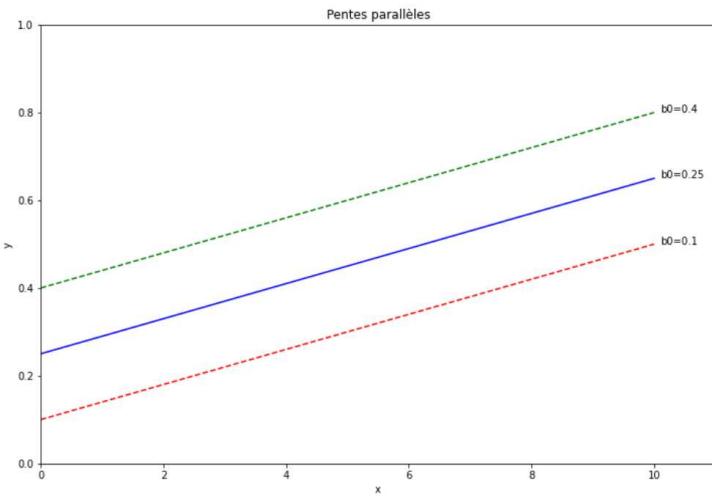


FIGURE 11.3 – Trois droites de régression fictives parallèles

Cas 2 : L'ordonnée à l'origine est la même d'une droite à l'autre

Dans ce cas, nous avons en termes de modélisation :

$$\text{O}3_i^j = \beta_0 + \beta_1^j T12^j + \varepsilon_i^j, \quad i = 1, \dots, n_j \quad (11.10)$$

et graphiquement :

```
# Graphique
plt.figure(figsize=(12,8))
plt.plot([0,10],[0.1,.5],linestyle='--',color = 'red')
plt.plot([0,10],[0.1,0.65],color = 'blue')
plt.plot([0,10],[0.1,0.8],linestyle='--',color = 'green')
plt.annotate('b1=1', xy =(10.1,.5),xytext =(10.1,.5))
plt.annotate('b1=2', xy =(10.1,.65),xytext =(10.1,.65))
plt.annotate('b1=3', xy =(10.1,.8),xytext =(10.1,.8))
plt.annotate('b0', xy =(0,.1),xytext =(-.3,.1))
plt.title('Ordonnée identique')
plt.ylabel('y')
plt.xlabel('x')
plt.ylim([0,1])
plt.xlim([0,11])
plt.show()
```

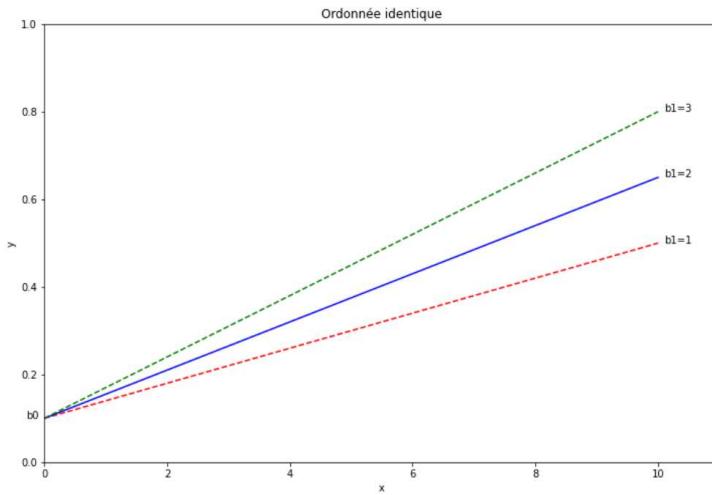


FIGURE 11.4 – Trois droites de régression fictives ayant la même ordonnée à l'origine

Dans le modèle (11.9), le coefficient β_1 est le même dans toutes les directions du vent. Cependant, si nous effectuons 4 régressions distinctes, comment trouverons-nous la même estimation de β_1 ? De même, dans le modèle (11.10), comment allons-nous procéder pour obtenir le même estimateur de β_0 dans chaque direction en effectuant 4 régressions distinctes? Par conséquent, il semble raisonnable de n'effectuer qu'une seule régression.

```
# Modèle par direction du vent
modelEST = OLS.from_formula(formula='03~T12',
                             data=donnee[donnee.vent=='EST']).fit()
modelNORD = OLS.from_formula(formula='03~T12',
                             data=donnee[donnee.vent=='NORD']).fit()
modelOUEST = OLS.from_formula(formula='03~T12',
                             data=donnee[donnee.vent=='OUEST']).fit()
modelsUD = OLS.from_formula(formula='03~T12',
                             data=donnee[donnee.vent=='SUD']).fit()

# Regroupement
EST = pd.concat([modelEST.params, modelEST.pvalues,
                 modelEST.conf_int(alpha=0.05)], axis=1)
NORD = pd.concat([modelNORD.params, modelNORD.pvalues,
                 modelNORD.conf_int(alpha=0.05)], axis=1)
OUEST = pd.concat([modelOUEST.params, modelOUEST.pvalues,
                 modelOUEST.conf_int(alpha=0.05)], axis=1)
SUD = pd.concat([modelsUD.params, modelsUD.pvalues,
                 modelsUD.conf_int(alpha=0.05)], axis=1)
params = pd.concat([EST, NORD, OUEST, SUD], axis=0)
params.columns = ['Coef.', 'P>|t|', '[0.025', '0.975]']
print(params.round(3))
```

vent		Coef.	P> t	[0.025	0.975]
EST	Intercept	45.609	0.004	16.966	74.252
	T12	2.748	0.000	1.444	4.052
NORD	Intercept	106.634	0.010	34.137	179.132
	T12	-1.649	0.379	-5.802	2.504
OUEST	Intercept	64.684	0.030	7.207	122.161
	T12	0.341	0.801	-2.472	3.153
SUD	Intercept	-27.060	0.108	-62.700	8.580
	T12	5.379	0.000	3.835	6.923

TABLE 11.2 – Estimation des paramètres sur les 4 régressions

On voit que pour les directions du vent NORD et OUEST, le coefficient β_1 n'est pas significatif.

11.3.2 Hypothèse gaussienne

Sous l'hypothèse de normalité des résidus, nous pouvons tester toutes les hypothèses linéaires possibles. Un des principaux objectifs de l'analyse de la covariance est de savoir si les variables explicatives influent sur la variable à expliquer. Les deux tests à effectuer sont :

1. Le test d'égalité des pentes

$$\begin{cases} H_0 : \beta_1^{\text{EST}} = \beta_1^{\text{NORD}} = \beta_1^{\text{OUEST}} = \beta_1^{\text{SUD}} = \beta_1 \\ H_1 : \exists(i,j) : \beta_1^i \neq \beta_1^j \end{cases}$$

2. Le test d'égalité des ordonnées à l'origine

$$\begin{cases} H_0 : \beta_0^{\text{EST}} = \beta_0^{\text{NORD}} = \beta_0^{\text{OUEST}} = \beta_0^{\text{SUD}} = \beta_0 \\ H_1 : \exists(i,j) : \beta_0^i \neq \beta_0^j \end{cases}$$

La statistique de test vaut donc :

$$F = \frac{||\hat{Y} - \hat{Y}_0||^2 / (K - 1)}{||Y - \hat{Y}||^2 / (n - K)} \quad (11.11)$$

L'hypothèse H_0 est rejetée en faveur de H_1 si l'observation de la statistique F est supérieure à $F_{K-1,n-K}(1-\alpha)$, la valeur α étant la probabilité de rejeter à tort H_0 ou erreur de première espèce et nous conclurons à l'effet du facteur explicatif.

Nous écrivons le modèle avec interaction :

```
# modèle avec interaction sans constante
modelinter = OLS.from_formula(formula ='03~-1+vent+vent:T12', data=donnee).fit()
print(modelinter.summary2())
```

Nous enlevons la constante en écrivant -1 . Ensuite, il faut conserver une ordonnée à l'origine différente pour chacune des modalités du `vent`, ce qui est représenté par le facteur `vent` (ou une interaction de la variable 1 avec `vent`). Ensuite, nous ajoutons un coefficient directeur différent pour chacun des modalités du vent, ce qui est représenté par la variable `T12` en interaction avec `vent`. Cela donne :

Results: Ordinary least squares

	Coef.	Std.Err.	t	P> t	[0.025	0.975]
vent[EST]	45.6090	13.9343	3.2731	0.0021	17.4884	73.7295
vent[NORD]	106.6345	28.0341	3.8037	0.0005	50.0594	163.2095
vent[OUEST]	64.6840	24.6208	2.6272	0.0120	14.9972	114.3709
vent[SUD]	-27.0602	26.5389	-1.0196	0.3137	-80.6179	26.4976
vent[EST]:T12	2.7480	0.6342	4.3333	0.0001	1.4682	4.0278
vent[NORD]:T12	-1.6491	1.6058	-1.0269	0.3103	-4.8897	1.5916
vent[OUEST]:T12	0.3407	1.2047	0.2828	0.7787	-2.0905	2.7719
vent[SUD]:T12	5.3786	1.1497	4.6783	0.0000	3.0585	7.6988
Omnibus:	0.276		Durbin-Watson:		1.561	
Prob(Omnibus):	0.871		Jarque-Bera (JB):		0.465	
Skew:	0.007		Prob(JB):		0.793	
Kurtosis:	2.528		Condition No.:		168	

Si dans l'écriture du modèle, la constante est conservée, le logiciel va prendre comme modalité de référence la première modalité (définie par ordre lexicographique). Cela donne :

```
# modèle avec interaction avec constante
modelinter2 = OLS.from_formula(formula ='03~+vent+vent:T12', data=donnee).fit()
print(modelinter2.summary2())
```

Results: Ordinary least squares

	Coef.	Std.Err.	t	P> t	[0.025	0.975]
Intercept	45.6090	13.9343	3.2731	0.0021	17.4884	73.7295
vent[T.NORD]	61.0255	31.3061	1.9493	0.0580	-2.1528	124.2038
vent[T.OUEST]	19.0751	28.2905	0.6743	0.5038	-38.0174	76.1675
vent[T.SUD]	-72.6691	29.9746	-2.4244	0.0197	-133.1604	-12.1779

```

vent[EST]:T12      2.7480   0.6342   4.3333  0.0001     1.4682   4.0278
vent[NORD]:T12    -1.6491   1.6058  -1.0269  0.3103    -4.8897   1.5916
vent[OUEST]:T12     0.3407   1.2047   0.2828  0.7787    -2.0905   2.7719
vent[SUD]:T12      5.3786   1.1497   4.6783  0.0000     3.0585   7.6988
-----
Omnibus:           0.276     Durbin-Watson:       1.561
Prob(Omnibus):    0.871     Jarque-Bera (JB):    0.465
Skew:              0.007     Prob(JB):          0.793
Kurtosis:          2.528     Condition No.:     223
=====

```

Les coefficients des ordonnées à l'origine sont des effets différentiels par rapport à la modalité de référence (EST), exemple $61.02 + 45.60 = 106.62$ valeur de vent [EST] dans l'écriture précédente.

Le modèle avec une seule pente peut s'écrire :

```

# Modèle avec une seule pente
model2 = OLS.from_formula(formula='03~vent+T12',data=donnee).fit()
model2b = OLS.from_formula(formula='03~-1+vent+T12',data=donnee).fit()

```

Le modèle avec une seule ordonnée à l'origine peut s'écrire :

```

# Modèle avec une seule ordonnée à l'origine
model3 = OLS.from_formula(formula='03~vent:T12',data=donnee).fit()

```

A présent, choisissons la meilleure modélisation :

1. **égalité des pentes** : Nous effectuons un test entre les modèles modelinter2 et model2.

```

# Test d'égalité des pentes
import statsmodels.api as sm

anova1 = sm.stats.anova_lm(model2,modelinter2,test="F", typ="I")
print(anova1)

      df_resid          ssr  df_diff      ss_diff          F    Pr(>F)
0      45.0  12611.951221      0.0        NaN      NaN      NaN
1      42.0   9087.430706      3.0  3524.520514  5.429839  0.003011

```

nous concluons donc à l'effet du vent sur les pentes. Nous aurions obtenu les mêmes résultats avec model2b contre modelinter2 ou model2 contre model2b ou encore model2b contre modelinter.

2. **égalités des ordonnées à l'origine** : nous effectuons un test entre le modèle model3 et le modèle modelinter2.

```

# Test d'égalité des ordonnées à l'origine
anova2 = sm.stats.anova_lm(model3,modelinter2,test="F", typ="I")
print(anova2)

```

	df_resid	ssr	df_diff	ss_diff	F	Pr(>F)
0	45.0	11864.056775	0.0	NaN	NaN	NaN
1	42.0	9087.430706	3.0	2776.626069	4.277641	0.010082

Nous concluons donc à l'effet du vent sur les ordonnées à l'origine.

Enfin le graphique de résidus (fig 11.5) obtenu avec

```
# Graphique des résidus
from statsmodels.stats.outliers_influence import OLSInfluence

influ = OLSInfluence(model2)
plt.figure(figsize=(12,8))
plt.scatter(model2.fittedvalues,influ.resid_studentized_external)
plt.title('Résidus studentisés\nversus\nvaleurs prédictes')
plt.xlabel('fitted values of O3')
plt.ylabel('résidus')
plt.show()
```

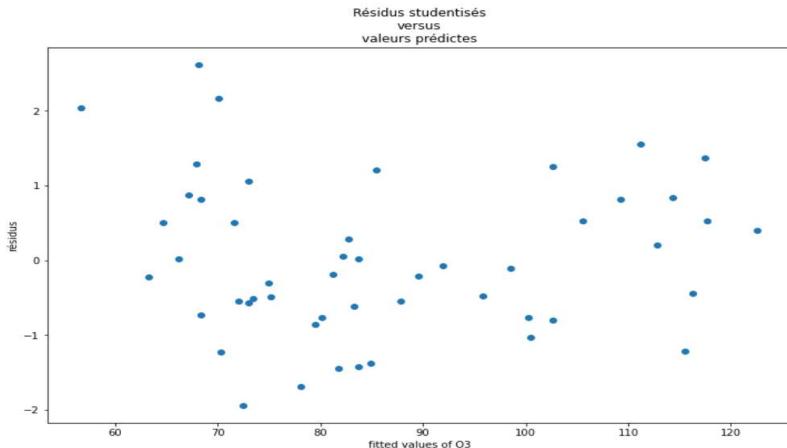


FIGURE 11.5 – Résidus studentisés

ne fait apparaître ni structure ni point aberrant. Nous conservons donc le modèle complet.

```
# modèle complet
model0 = OLS.from_formula(formula='O3~vent+T12+vent:T12',data=donnee).fit()
print(model0.summary2())
```

Results: Ordinary least squares

```
=====
Model:          OLS             Adj. R-squared:      0.621
Dependent Variable: O3              AIC:            418.0251
Date:        2022-02-13 15:21    BIC:            433.3213
No. Observations: 50              Log-Likelihood:   -201.01
Df Model:           7              F-statistic:       12.48
Df Residuals:        42             Prob (F-statistic): 1.61e-08
R-squared:         0.675          Scale:            216.37
```

	Coef.	Std.Err.	t	P> t	[0.025	0.975]
Intercept	45.6090	13.9343	3.2731	0.0021	17.4884	73.7295
vent[T.NORD]	61.0255	31.3061	1.9493	0.0580	-2.1528	124.2038
vent[T.OUEST]	19.0751	28.2905	0.6743	0.5038	-38.0174	76.1675
vent[T.SUD]	-72.6691	29.9746	-2.4244	0.0197	-133.1604	-12.1779
T12	2.7480	0.6342	4.3333	0.0001	1.4682	4.0278
vent[T.NORD]:T12	-4.3971	1.7265	-2.5468	0.0146	-7.8813	-0.9129
vent[T.OUEST]:T12	-2.4073	1.3614	-1.7682	0.0843	-5.1548	0.3402
vent[T.SUD]:T12	2.6306	1.3130	2.0035	0.0516	-0.0191	5.2803
Omnibus:	0.276		Durbin-Watson:		1.561	
Prob(Omnibus):	0.871		Jarque-Bera (JB):		0.465	
Skew:	0.007		Prob(JB):		0.793	
Kurtosis:	2.528		Condition No.:		407	

Intercept et T12 sont bien les valeurs de l'ordonnée à l'origine et de la pente pour le vent d'EST.

Remarque

Dans le modèle complet, nous utilisons bien les 3 variables vent, T12 et leur interaction. En effet, en écrivant ainsi, le logiciel, pourra imposer des contraintes lors de l'inversion de la matrice X.

Troisième partie

Régression logistique

CHAPITRE 12

INTRODUCTION À LA RÉGRESSION LOGISTIQUE

Sommaire

12.1 Retour sur le modèle linéaire gaussien	282
12.2 Vers le modèle linéaire généralisé	284

Tout au long de cette partie, on notera [voir [Rou15]] :

- $X = (1, X_1, \dots, X_p)'$: vecteur aléatoire de dimension $p + 1$, les marginales X_j sont les variables explicatives. On note également $x = (1, x_1, \dots, x_p)'$ une réalisation de X ;
- Y variable (univariée) à expliquer ;
- $(X_1, Y_1), \dots, (X_n, Y_n)$: un n-échantillon aléatoire (i.i.d. et de même loi que le couple (X, Y)), tel que $X_i = (1, X_{i1}, \dots, X_{ip})$;
- $(x_1, y_1), \dots, (x_n, y_n)$ une réalisation de $(X_1, Y_1), \dots, (X_n, Y_n)$.
- \mathbf{X} : la matrice des observations :

$$X = \begin{pmatrix} 1 & x_{i1} & \dots & x_{1p} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{pmatrix}$$

12.1 Retour sur le modèle linéaire gaussien

12.1.1 Rappels

Le modèle linéaire gaussien est donné par l'équation :

$$Y = X\beta + \varepsilon$$

qui se décline ligne à ligne (individu par individu) en

$$Y_i = \beta_0 + \sum_{j=1}^p \beta_j x_{i,j} + \varepsilon_i$$

Ce modèle a deux fonctions :

- Expliquer au mieux la variable Y comme une fonction linéaire de covariables X à laquelle s'ajoute un terme d'erreur individuelle, supposé gaussien et de variance constante ;
- Prédire la valeur de Y au vu d'un nouveau jeu de covariables (avec toutes les précautions d'usage quand on fait de la prédiction statistique).

Ce modèle est très populaire parce que son caractère gaussien se justifie souvent physiquement (via le théorème central limite), parce qu'il est assez facile à manier en pratique, et parce qu'il a des propriétés mathématiques bienvenues : notamment l'estimateur des moindres carrés ordinaires de β est égal à l'estimateur du maximum de vraisemblance.

Rappelons que ce modèle repose sur l'idée que les valeurs observées des covariables sont déterministes. Cela permet de dire, par exemple, que la loi de Y_i est gaussienne, d'espérance $\beta_0 + \sum_{j=1}^p \beta_j x_{i,j}$, et de variance σ^2 (hypothèse d'homoscédasticité).

Une autre approche est de dire que ces valeurs sont en fait des réalisations de variables aléatoires $X_{i,j}$. C'est un formalisme très naturel, mais qui complique un peu l'écriture : dans ce cas, c'est la loi conditionnelle de Y_i sachant $X_{i,1} = x_{i,1}, \dots, X_{i,p} = x_{i,p}$ qui est gaussienne d'espérance $\beta_0 + \sum_{j=1}^p \beta_j x_{i,j}$ (en faisant l'hypothèse supplémentaire que les $X_{i,j}$ sont indépendants de ε_i). Cette approche conditionnelle est souvent utilisé notamment en économétrie.

12.1.2 Limites

Parmi les hypothèses du modèle gaussien, on retient les suivantes :

- On peut **toujours** faire techniquement une régression linéaire, même quand cela n'est pas pertinent ;
- La variable à expliquer Y est gaussienne ;
- le bruit (l'erreur) est additif ;
- Les observations sont indépendantes ;
- Les observations sont de même variance (homoscédasticité) ;
- Les covariables sont supposées déterministes.

Dans le cours de modèle linéaire gaussien, il est possible de se ramener au modèle linéaire gaussien quand certaines de ces hypothèses ne sont pas vérifiées :

- transformation de variables pour se ramener à un bruit additif (typiquement : passage au logarithme dans le cas d'un bruit multiplicatif, nuages en trompette) ;

- Matrices de covariance plus générales pour ε quand la deuxième hypothèse n'est pas vérifiée ;
- Transformation de la variable Y pour la rendre gaussienne.

Il se trouve que cette dernière idée est inopérante dans des cas concrets extrêmement courants, en particulier quand la réponse en Y est intrinsèquement discrète : le modèle linéaire gaussien ne peut pas être utilisé pour expliquer Y avec l'aide des covariables (qui peuvent être continues ou non).

Quelques exemples illustratifs :

1. Retour à l'emploi en moins de trois mois ($Y_i = 1$ si oui, $Y_i = 0$ si non), à expliquer par l'entrée (ou non) dans un dispositif particulier, l'âge, le sexe, la CSP, le niveau d'études, le nombre de périodes de non emploi, etc.
2. Efficacité d'un traitement médical ($Y_i = 1$ si oui, $Y_i = 0$ si non), à expliquer par l'âge, le sexe, la CSP, etc.
3. Segmentation d'une clientèle en k catégories, l'affectation d'un individu dans une catégorie étant expliquer par l'âge, le sexe, la CSP, le revenu, les fréquentations, etc.
4. Caractère frauduleux d'une déclaration à l'ouverture d'un compte bancaire (régresseurs confidentiels, etc.)
5. Nombre de morts par accidentologie routière en un mois, à expliquer par l'état de la route, sa nature, etc.
6. Prêt accordé ou non, à expliquer par l'âge, le revenu, le taux d'endettement, le déficit budgétaire (pour un individu), la balance commerciale (pour un pays), etc.

Dans ces exemples, on peut s'intéresser notamment à l'effet spécifique d'un régresseur donné, par exemple le dispositif considéré dans l'exemple 1.

On peut remarquer que dans certains exemples (notamment dans l'exemple 1), c'est surtout l'**explication** Y par les covariables qui est intéressante, alors que pour d'autres c'est la partie **prédictive** qui l'est le plus (notamment dans les exemples 3 et 4). Mais à bien réfléchir, prédire une variable qui ne peut prendre que les valeurs 0 et 1 n'est pas intéressant d'un point de vue statistique, et il est intéressant d'essayer d'estimer la probabilité pour que cette variable prenne la valeur 1 (par exemple, la partie intervalle de confiance devient problématique). Quoi qu'il en soit, le modèle linéaire classique est clairement impuissant à traiter ces exemples.

12.2 Vers le modèle linéaire généralisé

Sur la base des remarques évoquées ci-dessous, nous reformulons l'objet de notre étude. Quand la variable à expliquer Y est binaire, sa loi est une loi de Bernoulli, et nous nous intéressons à l'influence des covariables X sur son paramètre, c'est-à-dire la probabilité $p(x)$ qu'elle vaille 1 si le vecteur des covariables est égal à x .

Dans le cas de l'exemple 2 ci-dessus, il est naturel de modéliser le nombre d'accidents par une loi de Poisson conditionnellement aux covariables (quitte à valider cette hypothèse a posteriori)

et l'objet d'intérêt devient le paramètre $\lambda(x)$ de cette loi pour une valeur donnée x des covariables.

Cette idée pourrait nous inciter à étudier des modèles du type :

$$p(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \quad (12.1)$$

et

$$\lambda(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \quad (12.2)$$

Mais cela ne peut être satisfaisant, parce que dans de tels modèles $p(x_i)$ peut être plus grand que 1 ou négatif, et $\lambda(x)$ négatif, ce qui est évidemment à proscrire.

On résout cette nouvelle difficulté par un changement de variable (cette fois-ci, c'est possible!), c'est-à-dire au moyen d'une transformation du paramètre permettant de couvrir tout \mathbb{R} . Voici trois exemples (non anodins) de telles transformations quand Y est une variable aléatoire de Bernoulli :

- $g(t) = \ln\left(\frac{t}{1-t}\right) = \text{logit}(t)$
- $g(t) = \Phi^{-1}(t)$ où Φ désigne la fonction de répartition de la loi normale centrée réduite ;
- $g(t) = \ln(-\ln(1-t))$ (on appelle parfois cette fonction **fonction log-log**)

Remarquons que pour le deuxième exemple, on pourrait prendre l'inverse de toute autre fonction de répartition d'une loi continue sur \mathbb{R} (**cauchy**, **student**, etc...). On s'intéresse alors au modèle donnée par :

$$g(p(x)) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \quad (12.3)$$

Dans le cas de l'intensité d'une loi de Poisson (conditionnellement aux valeurs prises par les cofacteurs), le même principe conduit à proposer le modèle

$$g(\lambda(x)) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \quad (12.4)$$

et $g(t) = \ln(t)$ est un choix assez naturel. Les fonctions g qui interviennent ci-dessus sont appelées **fonctions de lien**.

TABLE 12.1 – Exemples de fonction de lien

Loi	Nom du lien	Fonction de lien
Bernoulli/Binomiale	lien logit	$g(t) = \text{logit}(t) = \ln(t/(1-t))$
Poisson	lien log	$g(t) = \ln(t)$
Normale	lien identité	$g(t) = t$
Gamma	lien réciproque	$g(t) = -1/t$

12.2.1 Exemples de fonctions de liens

D'autres fonctions de lien que **logit** peuvent être utilisées dans le cas où la variable à expliquer Y est binaire. On trouve notamment dans la littérature les transformations :

- **probit**, qui n'est autre que l'inverse de la fonction de répartition de la loi normale centrée réduite :

$$\forall p \in [0,1], \quad \text{probit}(p) = \varepsilon \quad \text{avec} \quad \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\varepsilon} \exp\left(-\frac{1}{2}u^2\right) du = p$$

- **log-log** définie par :

$$\forall p \in [0,1], \quad \text{log-log}(p) = \log(-\log(1-p))$$

```
# Draw link function
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from scipy.special import logit
from scipy.stats import norm

def loglog(x):
    return np.log(-np.log(1-x))

p = np.linspace(0.05, 0.99, num=1000, endpoint=False)
logitvalue = logit(p)
probit = norm.ppf(p)
loglogvalue = loglog(p)

# graphique
plt.plot(p, logitvalue, color='steelblue', label='logit')
plt.plot(p, probit, color='red', label='probit', linestyle='--')
plt.plot(p, loglogvalue, color='green', label='log-log', linestyle='--')
plt.xlim([0,1])
plt.ylim([-4,4])
plt.xlabel('p')
plt.ylabel('g(p)')
plt.legend()
plt.show()
```

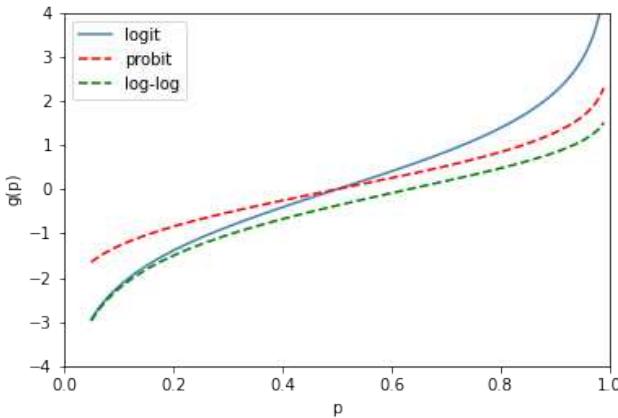


FIGURE 12.1 – Fonctions de liens : probit, logit, log-log

Des trois fonctions de lien présentées, la transformation **log-log** est bien appropriée aux cas où l'on souhaite modéliser les probabilités de succès de manière asymétrique. Les transformations **logit** et **probit** possèdent des propriétés identiques. Dans de nombreux cas, on préfère utiliser la transformation logistique. Plusieurs raisons motivent ce choix :

- D'un point de vue numérique, la transformation logistique est plus simple à manipuler (notamment pour l'écriture des estimateurs du maximum de vraisemblance) ;
- On a une interprétation claire des coefficients en terme d'odds ratio pour la transformation logistique ;
- Le modèle logistique est particulièrement bien adapté à un schéma d'échantillonnage rétrospectif.

12.2.2 Données

Autant que faire, on utilisera une base de données qui contient des données de 462 patients pour lesquels on souhaite prédire l'exposition à un infarctus. La base est disponible [ici](#). C'est d'ailleurs un site où des données sont disponibles afin de s'entraîner sur la modélisation en employant diverses techniques.

```
# Chargement des données
import pandas as pd

donnee = pd.read_csv('deseases.txt', sep=',', header=0, index_col=0)
print(donnee.head())
```

row.names	sbp	tobacco	ldl	adiposity	famhist	typea	obesity	alcohol
1	160	12.00	5.73	23.11	Present	49	25.30	97.20
2	144	0.01	4.41	28.61	Absent	55	28.87	2.06
3	118	0.08	3.48	32.28	Present	52	29.14	3.81

```

4      170    7.50  6.41     38.03 Present    51    31.99  24.26
5      134   13.60  3.50     27.78 Present    60    25.99  57.34

      age  chd
row.names
1      52  1
2      63  1
3      46  0
4      58  1
5      49  1

```

Description de la base :

- **sbp** : pression sanguine systolique (systolic blood pressure)
- **tobacco** : la qualité de tabac consommé cumulée (cumulative tobacco (kg))
- **ldl** : taux de cholestérol dans le sang (low density lipoprotein cholesterol)
- **adiposity** : adiposité
- **famhist** : antécédents familiaux : Présent s'il y a eu des antécédents familiaux (family history of heart disease (Present, Absent))
- **typea** : comportement de type A (type-A behavior)
- **obesity** : obésité
- **alcohol** : consommation courante d'alcool (current alcohol consumption)
- **age** : age au moment de l'attaque cardiaque (age at onset)
- **chd** : variable réponse codée 1 si la maladie du cœur est présente, 0 sinon (reponse, coronary heart disease)

Pour notre modélisation, on s'intéressera à cette dernière variable.

```

# infos
print(donnee.info())

<class 'pandas.core.frame.DataFrame'>
Int64Index: 462 entries, 1 to 463
Data columns (total 10 columns):
 Column      Non-Null Count Dtype
---  -----
0  sbp        462 non-null   int64
1  tobacco    462 non-null   float64
2  ldl        462 non-null   float64
3  adiposity  462 non-null   float64
4  famhist    462 non-null   object
5  typea      462 non-null   int64
6  obesity    462 non-null   float64
7  alcohol    462 non-null   float64
8  age        462 non-null   int64

```

```
9    chd      462 non-null    int64
dtypes: float64(5), int64(4), object(1)
memory usage: 39.7+ KB
```

On affiche les colonnes et leurs types.

```
# liste des colonnes et leurs types
print(donnee.dtypes)
```

```
sbp      int64
tobacco  float64
ldl      float64
adiposity  float64
famhist   object
typea    int64
obesity  float64
alcohol   float64
age      int64
chd      int64
dtype: object
```

Notre variable chd dans la base est de type `int`. Nous la transformons en variable catégorielle.

```
# transformation en variable catégorielle
donnee['chd'] = donnee['chd'].astype('category')
print(donnee.dtypes)
```

```
sbp      int64
tobacco  float64
ldl      float64
adiposity  float64
famhist   object
typea    int64
obesity  float64
alcohol   float64
age      int64
chd      category
dtype: object
```

CHAPITRE 13

RÉGRESSION LOGISTIQUE BINAIRE

Sommaire

13.1 Un cadre bayesien pour l'apprentissage supervisée	290
13.2 Analyse discriminante logistique	293
13.3 L'estimateur du maximum de vraisemblance	295
13.4 Précision des estimateurs	308
13.5 Première évaluation de la régression logistique : les pseudo- R^2	315

13.1 Un cadre bayesien pour l'apprentissage supervisé

13.1.1 Problématique - mise en oeuvre

Rappelons que dans le cadre de la discrimination binaire [voir [Rak11]], nous considérons que la variable dépendante Y ne prend que 2 modalités : positif (+) ou négatif (-). Nous cherchons à prédire correctement les valeurs de Y , mais nous pouvons également vouloir quantifier la propension (la probabilité) d'un individu à être positif (ou négatif).

Le classifieur naïf bayésien repose sur l'hypothèse forte que les variables explicatives X_j sont indépendantes conditionnellement à la classe de Y . Malgré cette hypothèse simpliste d'indépendance entre les regresseurs conditionnellement aux valeurs prises par Y , le classifieur bayésien naïf s'avère souvent plus performant que des modèles plus sophistiqués.

Si le nombre de regresseurs est grand, cette technique est particulière appropriée. En effet, en grande dimension, l'estimation des fonctions de densité devient très compliquée. Avec le classifieur bayésien naïf, chaque loi de probabilité des X_j conditionnellement aux valeurs de Y peut être estimée indépendamment en tant que loi de probabilité à une dimension.

Pour un individu ω , il s'agit de calculer les probabilités conditionnelles (probabilités a posteriori) :

$$\mathbb{P}[Y(\omega) = y_k | X(\omega)] \tag{13.1}$$

pour chaque modalité y_k de Y . On affecte à l'individu la modalité la plus probable y_{k^*} c'est-à-dire :

$$y_{k^*} = \arg \max_k \mathbb{P}[Y(\omega) = y_k | X(\omega)] \quad (13.2)$$

On associe donc l'individu à la classe la plus probable compte tenu de ses caractéristiques $X(\omega)$. Cette approche est optimale au sens de l'erreur théorique, mais un problème apparaît tout de suite : **comment estimer correctement les probabilités conditionnelles?**

Exemple : prédire chd en fonction de famhist

Pour nos données de la section 12.2.2, nous souhaitons prédire les valeurs de `chd` en fonction de `famhist`. La variable prédictive `famhist` étant binaire (et de manière plus générale catégorielle), nous pouvons utiliser les fréquences pour estimer les probabilités conditionnelles. Premièrement, on construit le tableau de contingence :

```
# transformation en variable catégorielle
donnee['chd'] = donnee['chd'].astype('category')
donnee["famhist"] = donnee["famhist"].astype('category')
# Tableau de contingence
crosstab = pd.crosstab(donnee.chd,donnee.famhist)
print(crosstab)

famhist  Absent  Present
chd
0          206      96
1          64       96
```

On calcule ensuite fréquences conditionnelles

```
# Fréquences conditionnelles
condmean = crosstab.apply(lambda x:100*x/sum(x), axis=0)
condmean.loc[['Total']] = condmean.sum(axis=0)
print(condmean.round(2))

famhist  Absent  Present
chd
0          76.3     50.0
1          23.7     50.0
Total     100.0    100.0
```

Pour `famhist = Absent`, nous avons les fréquences conditionnelles :

- $\mathbb{P}(chd = 0 / famhist = Absent) = 0.763$
- $\mathbb{P}(chd = 1 / famhist = Absent) = 0.237$

En vertu du principe bayesien, nous adoptons le règle suivante :

Si `famhist = Absent` alors `chd=0 (absent)`

De la même manière, pour `famhist = Present`, nous calculons :

- $\mathbb{P}(chd = 0/famhist = Present) = 0.50$
- $\mathbb{P}(chd = 1/famhist = Present) = 0.50$

Nous en déduisons :

Si `famhist = Present` alors `chd=1 (present)`

Maintenant nous avons un modèle de prédiction `chd = f(famhist)`, il faut en évaluer les performances. Pour cela, nous confrontons les vraies valeurs de la variable dépendante avec celles prédictes par le modèle.

```
# prédiction
import numpy as np
prediction = np.where(donnee.famhist=='Present', 1, 0)
# error rate
error = np.sum(donnee.chd!= prediction)/donnee.shape[0]
print("Erreur en resubstitution : %.4f" % (error))

Erreur en resubstitution : 0.3463
```

Nous en déduisons le taux d'erreur 0.3463 c'est-à-dire si nous classons un individu pris au hasard dans la population, nous avons 34.63% de chances de faire une prédiction erronée. A l'inverse, nous avons 65.37% de chances de faire une prédiction correcte.

Attention, il s'agit bien d'une erreur en resubstitution puisque le modèle a été élaboré (dans notre cas, les probabilités conditionnelles ont été calculées) à partir des mêmes données. Les performances annoncées sont donc sujettes à caution.

13.1.2 Limite de l'approche basée sur les fréquences

La démarche basée sur les fréquences est extrêmement séduisante par sa simplicité. Un simple comptage permet de produire les probabilités conditionnelles et déduire les règles d'affectation. Toutefois, elle n'est pas viable en situation réelle, lorsque nous avons plus d'une variable prédictive, pour différentes raisons :

1. Dans le cas où toutes les variables sont binaires, le nombre de probabilités à calculer devient rapidement prohibitif, impossible à gérer même sur les ordinateurs. Par exemple, si nous avons 20 variables, il faudrait procéder à $2 \times 2^{20} = 2.097.152$ comptages.
2. Et même si cela était possible, nous aurions la valeur 0 dans la plupart des cases de notre tableau croisé, ou tout du moins de très faibles effectifs, rendant inutilisables les estimations.
3. L'affaire se corse lorsque nous avons des descripteurs continus. Procéder par comptage global n'a plus de sens. Il faut passer par d'autres stratégies : soit en **discrétilisant** ces variables (les découper en intervalles) ; soit en **estimant par comptage** les probabilités, mais localement, en se limitant au voisinage de l'observation à classer (cf. par exemple la méthode des plus proches voisins, les noyaux de Parzen, etc.).

4. Et on ne parle même pas de la situation où l'on a un mélange de variables prédictives continues et catégorielles. La solution pourrait passer par un découpage en classes des variables continues, mais il faudrait proposer des découpages pertinents, au moins en relation avec la variable à prédire, et peut être aussi en relation avec les autres variables prédictives pour tenir compte des possibles interactions.
5. Enfin, en admettant que tous les problèmes ci-dessus aient été résolus, il reste un écueil : il n'y a pas de processus de sélection de variables inhérent à la méthode. Elle ne nous indique pas quelles sont les variables pertinentes qu'il faut conserver, quelles sont les variables qui ne servent rien et que l'on peut évacuer. Pourtant, cet aspect est incontournable dès que l'on est confronté à un problème un tant soit peu réaliste. L'expert du domaine a certes une idée plus ou moins vague des « bonnes » variables, mais bien souvent il compte sur les techniques numériques pour préciser ses idées.

13.2 Analyse discriminante logistique

13.2.1 Modèle linéaire généralisé : GLM

Nous nous plaçons dans un contexte de classification binaire [voir [\[Hil16\]](#)], c'est-à-dire que nous supposons qu'il existe seulement deux groupes à discriminer.

Exemple 13.1 Nous souhaitons expliquer la variable Y présence (1)/absence (0) d'une maladie cardio-vasculaire (chd) par l'âge des patients. Les données sont représentées sur la figure 13.1

```
# chd en fonction de l'age
from matplotlib import pyplot as plt
plt.scatter(donnee.age,donnee.chd, color='black')
plt.xlabel('age')
plt.ylabel('chd')
plt.savefig('logit2') # save image
plt.show()
```

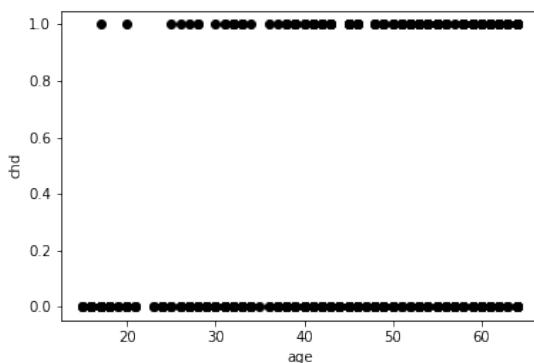


FIGURE 13.1 – Représentation directe de chd en fonction de l'âge

Définition

On suppose désormais qu'on observe $p + 1$ vecteurs aléatoires $(Y_i, X_{i,1}, \dots, X_{i,p})$ indépendants. On notera $X_i = (1, X_{i,1}, \dots, X_{i,p})$. On se dote par ailleurs d'un vecteur de paramètres $\beta = (\beta_0, \beta_1, \dots, \beta_p)$, et d'une fonction de lien g bijective et dérivable.

Definition 13.1 *Un modèle linéaire généralisé (GLM) établit une relation linéaire de type*

$$g(\mathbb{E}(Y_i/X_i = x_i)) = x_i \cdot \beta$$

ou, ce qui revient au même

$$g(\mathbb{E}(Y_i/X_i)) = X_i \cdot \beta$$

où le produit scalaire s'écrit $x_i \cdot \beta = \beta_0 + \beta_1 x_{i,1} + \dots + \beta_p x_{i,p}$

On supposera tout au long de ce document que la loi conditionnelle de Y_i sachant X_i appartient à la famille exponentielle (afin de garantir les bonnes qualités des estimateurs à étudier).

13.2.2 Exemples de modèle GLM

Exemple 13.2 (Le modèle linéaire gaussien)

On le retrouve en prenant des Y_i de loi normale (conditionnellement aux X_i), avec la fonction de lien identité $g(t) = t$. On a alors $\mathbb{E}(Y_i/X_i = x_i) = x_i \cdot \beta$, autrement dit $\mathcal{L}(Y_i/X_i = x_i)$ est la loi normale d'espérance $x_i \cdot \beta$ et de variance σ^2 supposée indépendante de i (homoscédasticité). Cela revient à dire que $Y_i = x_i \cdot \beta + \varepsilon_i$, où ε_i est une variable aléatoire de loi normale, centrée et de variance σ^2 .

Exemple 13.3 (Le modèle logistique)

Ici, les Y_i sont de loi de Bernoulli, avec la fonction de lien $g(t) = \ln(t/(1-t)) = \text{logit}(t)$. On a donc le modèle

$$\ln\left(\frac{p(x_i)}{1-p(x_i)}\right) = x_i \cdot \beta \quad (13.3)$$

équation qui s'inverse en

$$p(x_i) = \frac{e^{x_i \cdot \beta}}{1 + e^{x_i \cdot \beta}}$$

Exemple 13.4 (Le modèle probit)

Ici, les Y_i sont de loi de Bernoulli, avec la fonction de lien $g(t) = \Phi^{-1}(t)$ où Φ désigne la fonction de répartition de la loi normale centrée réduite. On a donc le modèle

$$\Phi^{-1}(p(x_i)) = x_i \cdot \beta \quad (13.4)$$

équation qui s'inverse en

$$p(x_i) = \Phi(x_i \cdot \beta)$$

Exemple 13.5 (Le modèle log-log (ou de Gomperz))

Ici, les Y_i sont de loi de Bernoulli, avec la fonction de lien $g(t) = \ln(-\ln(1-t))$. On a donc le modèle :

$$\ln(-\ln(1-p(x_i))) = x_i \cdot \beta \quad (13.5)$$

équation qui s'inverse en

$$p(x_i) = 1 - e^{-e^{x_i \cdot \beta}}$$

Exemple 13.6 (Le modèle log-linéaire, ou de Poisson)

Ici, les Y_i sont, conditionnellement aux X_i , de loi de Poisson, avec la fonction de lien $g(t) = \ln(t)$. On a donc le modèle

$$\ln(\lambda(x_i)) = x_i \cdot \beta \quad (13.6)$$

équation qui s'inverse en

$$\lambda(x_i) = e^{x_i \cdot \beta}$$

Remarque :

Notez que l'hypothèse selon laquelle la variable aléatoire Y_i est de loi de Poisson conditionnellement à chaque valeur possible des covariables n'entraîne pas que Y_i soit de loi (non conditionnelle) de Poisson.

13.3 L'estimateur du maximum de vraisemblance

13.3.1 Estimation des paramètres

Il s'agit donc d'expliquer une variable binaire Y par p régresseurs X_j , $1 \leq j \leq p$ auxquels on ajoute le régresseur constant $X_0 \equiv 1$, à l'aide du modèle de régression logistique, et ce au vu de n -copies (observations) indépendantes du vecteur $(Y, 1, X_1, \dots, X_p)$. Le modèle s'écrit donc :

$$\text{logit}(p_\beta(X)) = \ln\left(\frac{p_\beta(X)}{1-p_\beta(X)}\right) = X \cdot \beta$$

où $\beta = (\beta_0, \beta_1, \dots, \beta_p)', p_\beta(X) = \mathbb{P}(Y = 1/X)$, et $X \cdot \beta = \beta_0 + X_1\beta_1 + \dots + X_p\beta_p$. On prendra systématiquement des modèles contenant la constante comme régresseur d'ordre 0. L'écriture p_β signifie tout simplement que cela dépend de la valeur du paramètre choisi.

Remarque

Il est essentiel que le modèle soit identifiable (cf. partie II) ; dans le cas du modèle logistique, cela s'interprète par le fait que si on a deux valeurs différentes β et β' pour le paramètre, on doit trouver une observation x des régresseurs pour laquelle $p_\beta(x) \neq p_{\beta'}(x)$. Mathématiquement, cela se traduit par le fait que la matrice $X'X$, où $X = (1, X_1, \dots, X_p)$ est de rang $p+1$, ce qui implique en particulier que $n \geq p+1$ (on a davantage d'observations que de régresseurs).

Le calcul de la vraisemblance se fait comme dans le cas du modèle linéaire gaussien, même si formellement on calcule des vraisemblances conditionnelles : l'indépendance permet de dire que

la vraisemblance conditionnelle d'un n -uplet d'observations est le produit des vraisemblances de chaque observation. Dans le cas du modèle logistique, cela donne :

$$\begin{aligned}
L(Y_1, \dots, Y_n, \beta / X_1 = x_1, \dots, X_n = x_n) &= \prod_{i=1}^n p_\beta(x_i)^{Y_i} (1 - p_\beta(x_i))^{1-Y_i} \\
&= \prod_{i=1}^n \left(\frac{e^{x_i \beta}}{1 + e^{x_i \beta}} \right)^{Y_i} \left(1 - \frac{e^{x_i \beta}}{1 + e^{x_i \beta}} \right)^{1-Y_i} \\
&= \prod_{i=1}^n \left(\frac{e^{x_i \beta}}{1 + e^{x_i \beta}} \right)^{Y_i} \left(\frac{1}{1 + e^{x_i \beta}} \right)^{1-Y_i} \\
&= \prod_{i=1}^n \left(e^{x_i \beta} \right)^{Y_i} \left(\frac{1}{1 + e^{x_i \beta}} \right)
\end{aligned}$$

Après passage à la log-vraisemblance, on arrive à l'expression

$$\ln L(Y, \beta) = \sum_{i=1}^n (Y_i x_i \cdot \beta - \ln(1 + e^{x_i \beta}))$$

Le calcul de l'EMV se fait en annulant les dérivées partielles par rapport aux $\beta_j, 0 \leq j \leq p$. Remarquons qu'on peut écrire $Y_i x_i \cdot \beta$ sous la forme $\sum_{k=0}^p Y_i x_{ik} \beta_k$. Cette dernière sera utile pour le calcul des dérivées partielles. Le vecteur gradient au point β défini par $\nabla \mathcal{L}_n(\beta) = \left[\frac{\partial \mathcal{L}_n}{\partial \beta_0}(\beta), \dots, \frac{\partial \mathcal{L}_n}{\partial \beta_p}(\beta) \right]'$ s'obtient par dérivation

$$\begin{aligned}
\frac{\partial \mathcal{L}_n}{\partial \beta_j}(\beta) &= \sum_{i=1}^n \left[Y_i x_{ij} - \frac{x_{ij} e^{x_i \beta}}{1 + e^{x_i \beta}} \right] \\
&= \sum_{i=1}^n x_{ij} \left[Y_i - \frac{e^{x_i \beta}}{1 + e^{x_i \beta}} \right] \\
&= \sum_{i=1}^n x_{ij} (Y_i - p_\beta(x_i))
\end{aligned}$$

Ce qui donne en écriture matricielle

$$\begin{aligned}
\nabla \mathcal{L}_n(\beta) &= \sum_{i=1}^n x_i (Y_i - p_\beta(x_i)) \\
&= X'(Y - p_\beta(X))
\end{aligned}$$

L'estimateur du maximum de vraisemblance (s'il existe) est solution de l'équation (appelée équation du score) :

$$S(\beta) = \nabla \mathcal{L}_n(\beta) = X'(Y - p_\beta(X)) = 0$$

On rappelle que si cette équation admet une solution en β alors l'estimateur du maximum de vraisemblance est $\hat{\beta} = g(Y_1, \dots, Y_n)$. Cela ne peut se résoudre explicitement et nécessite l'emploi de méthodes numériques. Ce qui explique que l'on obtienne parfois des résultats différents d'un logiciel à l'autre : le résultat obtenu dépend de l'algorithme utilisé, du paramétrage adopté, et parfois même des choix d'implémentation de l'informaticien. Toutefois, l'existence et l'unicité de l'EMV sont assurées sous les hypothèses d'identifiabilité (voir plus haut) et de recouvrement.

Puisque $\hat{\beta}$ est un estimateur du maximum de vraisemblance, il en possède toutes les propriétés :

- Il est asymptotiquement sans biais ;
- Il est de variance minimale ;
- Il est asymptotiquement gaussien.

Ces éléments, notamment le dernier, sont très importants pour l'inférence statistique (intervalle de confiance, test de significativité, etc.)

Application

Pour illustrer nos propos, on souhaite déterminer la probabilité d'avoir une maladie du coeur compte tenu des descripteurs relevés.

$$\text{chd} = f(\text{sbp}, \text{tobacco}, \text{age}, \text{etc.})$$

```
## Estimation par Maximum de vraisemblance
# codage 0/1
donnee["famhist"] = donnee["famhist"].astype('category').cat.codes
x = donnee.drop(['chd'], axis=1) # suppression
y = donnee['chd']
# Importation de la classe de calcul
import statsmodels.api as sm
# on ajoute une colonne pour la constante
X = sm.add_constant(x)
# on ajuste le modèle
fullmodel = sm.Logit(y, X)
# lancer les calculs
fullresult = fullmodel.fit()

Optimization terminated successfully.
    Current function value: 0.510974
    Iterations 6
```

Commentaires :

- Par défaut, l'outil s'appuie sur l'algorithme de Newton-Raphson. D'autres procédures d'optimisation sont disponibles. 6 itérations ont été nécessaires pour maximiser la log-vraisemblance.

Les résultats détaillés de la régression sont affichés avec la commande `summary()`

Logit Regression Results			
Dep. Variable:	chd	No. Observations:	462
Model:	Logit	Df Residuals:	452
Method:	MLE	Df Model:	9
Date:	Fri, 22 Oct 2021	Pseudo R-squ.:	0.2080
Time:	23:40:20	Log-Likelihood:	-236.07
converged:	True	LL-Null:	-298.05
Covariance Type:	nonrobust	LLR p-value:	2.055e-22

	coef	std err	z	P> z	[0.025	0.975]
const	-6.1507	1.308	-4.701	0.000	-8.715	-3.587
sbp	0.0065	0.006	1.135	0.256	-0.005	0.018
tobacco	0.0794	0.027	2.984	0.003	0.027	0.132
ldl	0.1739	0.060	2.915	0.004	0.057	0.291
adiposity	0.0186	0.029	0.635	0.526	-0.039	0.076
famhist	0.9254	0.228	4.061	0.000	0.479	1.372
typea	0.0396	0.012	3.214	0.001	0.015	0.064
obesity	-0.0629	0.044	-1.422	0.155	-0.150	0.024
alcohol	0.0001	0.004	0.027	0.978	-0.009	0.009
age	0.0452	0.012	3.728	0.000	0.021	0.069

ou la commande [summary2\(\)](#)

Results: Logit

Model:	Logit	Pseudo R-squared:	0.208
Dependent Variable:	chd	AIC:	492.1400
Date:	2021-10-22 23:46	BIC:	533.4957
No. Observations:	462	Log-Likelihood:	-236.07
Df Model:	9	LL-Null:	-298.05
Df Residuals:	452	LLR p-value:	2.0548e-22
Converged:	1.0000	Scale:	1.0000
No. Iterations:	6.0000		

	Coef.	Std.Err.	z	P> z	[0.025	0.975]
const	-6.1507	1.3083	-4.7015	0.0000	-8.7149	-3.5866
sbp	0.0065	0.0057	1.1350	0.2564	-0.0047	0.0177
tobacco	0.0794	0.0266	2.9838	0.0028	0.0272	0.1315
ldl	0.1739	0.0597	2.9152	0.0036	0.0570	0.2909
adiposity	0.0186	0.0293	0.6346	0.5257	-0.0388	0.0760
famhist	0.9254	0.2279	4.0605	0.0000	0.4787	1.3720
typea	0.0396	0.0123	3.2138	0.0013	0.0154	0.0637
obesity	-0.0629	0.0442	-1.4218	0.1551	-0.1496	0.0238
alcohol	0.0001	0.0045	0.0271	0.9784	-0.0087	0.0089
age	0.0452	0.0121	3.7285	0.0002	0.0215	0.0690

Le logit estimé permettant de prédire l'occurrence d'une maladie cardiaque à partir de l'âge, etc., s'écrit :

$$\text{logit}_\beta(X) = -6.1507 + 0.0065 \times \text{sbp} + 0.0794 \times \text{tobacco} + \dots + 0.0452 \times \text{age}$$

13.3.2 Dimensions explicatives, variables explicatives

1. Variable explicative quantitative

Si on dispose d'une seule variable explicative X quantitative (non regroupés en classe) le modèle s'écrit

$$\text{logit}_\beta(x) = \beta_0 + \beta_1 x$$

Un seul coefficient (β_1) est alloué à X , cette variable est représentée par une seule colonne dans la matrice du design X. Sa dimension est donc égale à 1.

Exemple 13.7 Considérons les variables *chd* et *age* disponibles dans notre jeu de données.

```
# Importation de la classe de calcul
import statsmodels.api as sm
# on ajuste le modèle
lowmodel = sm.Logit(donnee.chd,sm.add_constant(donnee.age))
lowresult = lowmodel.fit()
print(lowresult.summary())

Optimization terminated successfully.
    Current function value: 0.568790
    Iterations 6
                    Logit Regression Results
=====
Dep. Variable:                  chd   No. Observations:             462
Model:                          Logit   Df Residuals:                  460
Method:                         MLE    Df Model:                      1
Date: Fri, 22 Oct 2021   Pseudo R-squ.:           0.1183
Time: 23:49:19                 Log-Likelihood:        -262.78
converged:                      True    LL-Null:            -298.05
Covariance Type:                nonrobust   LLR p-value: 4.496e-17
=====
              coef      std err       z     P>|z|      [0.025      0.975]
-----
const     -3.5217      0.416    -8.465     0.000     -4.337    -2.706
age        0.0641      0.009     7.513     0.000      0.047     0.081
=====
```

2. Variable explicative qualitative

Tout comme pour le modèle d'analyse de variance, une variable qualitative est représentée par les indicatrices associées aux différentes modalités. Considérons un modèle où la seule variable explicative est *famhist* :

$$\text{logit}_\beta(x) = \beta_0 + \beta_A 1_A(x) + \beta_P 1_P(x) \quad (13.7)$$

Mais aussi

$$\text{logit}_\beta(x) = (\beta_0 + \beta_A) + (\beta_P - \beta_A) 1_P(x)$$

Exemple 13.8 Considérons le cas où *famhist* explique *chd*

On effectue une régression logistique sur Python :

```

# Importation de la classe de calcul
import statsmodels.api as sm
# on ajuste le modèle
lowmodel2 = sm.Logit(donnee.chd,sm.add_constant(donnee.famhist))
lowresult2 = lowmodel2.fit()
print(lowresult2.summary())

Optimization terminated successfully.
    Current function value: 0.608111
    Iterations 5
            Logit Regression Results
=====
Dep. Variable:                      chd   No. Observations:             462
Model:                            Logit   Df Residuals:                  460
Method:                           MLE    Df Model:                      1
Date:          Fri, 22 Oct 2021   Pseudo R-squ.:           0.05740
Time:              23:51:33      Log-Likelihood:        -280.95
converged:                     True    LL-Null:           -298.05
Covariance Type:                nonrobust   LLR p-value:       4.937e-09
=====
              coef      std err      z      P>|z|      [0.025      0.975]
-----
const      -1.1690      0.143     -8.169      0.000     -1.449     -0.889
famhist      1.1690      0.203      5.751      0.000      0.771      1.567
=====
```

Le modèle estimé s'écrit donc :

$$\text{logit}\hat{p}(famhist) = \text{logit}\hat{p}_\beta(famhist) = -1.1690 + 1.1690 \times 1_{\text{famhist}=\text{Present}}$$

13.3.3 Interprétation des coefficients β

Sur la figure 13.2, nous avons représenté l'allure de la courbe représentative de la fonction $x \mapsto \frac{\exp(x\beta)}{1 + \exp(x\beta)}$ pour différentes valeurs du paramètre β .

```

# Représentation graphique
x = np.linspace(-5,5,50)
def f(t,beta):
    return((np.exp(t*beta))/(1+np.exp(t*beta)))

betachoice = [0,0.5,2,10]
mat = pd.DataFrame(np.zeros((len(x),len(betachoice))),
                    columns = ['beta=0','beta=0.5','beta=2','beta=10'])

for idx, value in enumerate(betachoice):
    mat.iloc[:,idx] = f(x,value)

# Représentation graphique
fig, axe = plt.subplots(2, 2, figsize=(12,7))
# beta=0
```

```

axe[0,0].plot(mat.iloc[:,0])
axe[0,0].set_ylim((0,1))
axe[0,0].set_xticks([])
axe[0,0].set_xlabel('beta=0')
axe[0,0].set_yticks(ticks=[0.3,0.7])
axe[0,0].set_yticklabels(labels=[str(x) for x in [0.3,0.7]])
# beta=0.5
axe[0,1].plot(mat.iloc[:,1])
axe[0,1].set_ylim((0,1))
axe[0,1].set_xticks([])
axe[0,1].set_xlabel('beta=0.5')
axe[0,1].set_yticks(ticks=[0.2,0.8])
axe[0,1].set_yticklabels(labels=[str(x) for x in [0.2,0.8]])
# beta=2
axe[1,0].plot(mat.iloc[:,2])
axe[1,0].set_xticks([])
axe[1,0].set_xlabel('beta=2')
axe[1,0].set_yticks(ticks=[0.0,1.0])
axe[1,0].set_yticklabels(labels=[str(x) for x in [0.0,1.0]])
# beta=10
axe[1,1].plot(mat.iloc[:,3])
axe[1,1].set_xticks([])
axe[1,1].set_xlabel('beta=10')
axe[1,1].set_yticks(ticks=[0.0,1.0])
axe[1,1].set_yticklabels(labels=[str(x) for x in [0.0,1.0]])
plt.show()

```

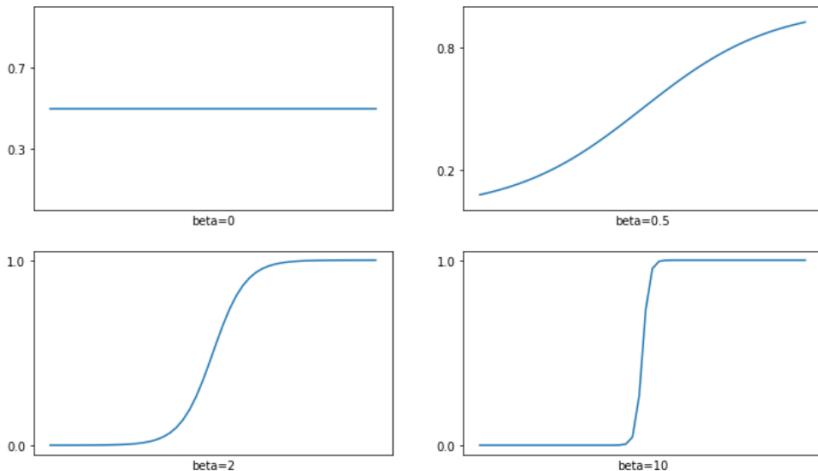


FIGURE 13.2 – $\mathbb{P}_\beta(Y = 1|X = x)$ pour différentes valeurs de β .

On remarque que :

- pour de faibles valeurs de β , on a une large plage de valeurs de x pour lesquelles la fonction se situe aux alentours de 0.5 (la fonction est même constante (0.5) dans le cas extrême $\beta = 0$).

Pour ces valeurs $p_\beta(x) = \mathbb{P}_\beta(Y = 1|X = x)$ sera proche de 0.5 et on peut donc penser qu'il sera difficile de discriminer ;

- Lorsque β augmente, la zone où la fonction est proche de 0.5 diminue et la fonction est proche de 0 ou de 1 pour un grand nombre de valeurs de x . Par conséquent, $\mathbb{P}_\beta(Y = 1|X = x)$ sera souvent proche de 1 ou 0, ce qui risque de minimiser d'éventuelles erreurs de prévisions.

On peut interpréter ainsi : *plus β est grand, mieux on discrimine*. Cependant, une telle interprétation dépend des valeurs que x prend (de son échelle). C'est pourquoi en général l'interprétation des coefficients β s'effectue en termes d'*odds ratio*. Les odds ratio sont des outils souvent appréciés dans le domaine de l'épidémiologie (mais pas toujours bien utilisés !).

Les odds ratio servent à mesurer l'effet d'une variable quantitative ou le contraste entre les effets d'une variables qualitative. L'idée générale est de raisonner en termes de probabilités ou de rapport de cotes (odds). Si on a, par exemple, une probabilité $p = 1/4$ de gagner à un jeu, cela signifie que sur 4 personnes une gagne et les trois autres perdent, soit un rapport de 1 gagnant sur trois perdants, c'est-à-dire $p/(1-p) = 1/3$. Ce rapport $p/(1-p)$ varie entre 0 (0 gagnant) et l'infini (que des gagnants) en passant par 1 (un gagnant pour un perdant).

1. Risque relatif, odds, odds ratio

Soit le tableau de contingence suivant mettant en liaison la variable cible Y binaire et une variable explicative catégorielle (binaire) :

		X	0	1	Total
		Y	a	b	a+b
		1	c	d	c+d
		Total	a+c	b+d	n

TABLE 13.1 – Tableau de contingence - Croisement Y vs X

Definition 13.2 (Risque relatif) On appelle **risque relatif**, le surcroît de chances d'être positif du groupe exposé par rapport au groupe témoin.

$$\begin{aligned} RR &= \frac{\mathbb{P}(Y = 1|X = 1)}{\mathbb{P}(Y = 1|X = 0)} \\ &= \frac{d}{a} \\ &= \frac{b + d}{a + c} \end{aligned}$$

Croisement chd vs famhist

```
contingence = pd.crosstab(donnee.chd, donnee.famhist)
contingence.loc['Total'] = contingence.sum(axis=0)
contingence.loc[:, 'Total']=contingence.sum(axis=1)
print(contingence)

famhist      0      1  Total
chd
0          206    96   302
```

1	64	96	160
Total	270	192	462

On calcule le risque relatif :

$$RR = \frac{96/192}{206/270} = 0.66$$

```
# Risque relatif
num_rr = contingence.values[1,1]/contingence.values[2,1]
deno_rr = contingence.values[0,0]/contingence.values[2,0]
risque_relatif = num_rr/deno_rr
print('Risque relatif : %.2f' %(risque_relatif))

Risque relatif : 0.66
```

Interprétation

Definition 13.3 (Odds) L'*odds* ou rapport de chances est défini comme un rapport de probabilités dans un groupe. Par exemple, dans le groupe exposé, il s'écrit :

$$\begin{aligned} odds(1) &= \frac{\mathbb{P}(Y=1/X=1)}{\mathbb{P}(Y=0/X=1)} = \frac{\frac{d}{b+d}}{\frac{b}{b+d}} = \frac{d}{b} \\ &= \frac{96}{96} = 1 \end{aligned}$$

```
# odds
odds = contingence.values[1,1]/contingence.values[0,1]
print('odds(1) : %.2f' % (odds))

odds(1) : 1.00
```

Interprétation :

Definition 13.4 (Odds-ratio) L'*odds ratio* est égal au rapport entre l'*odds* du groupe exposé et l'*odds* du groupe témoin.

$$\begin{aligned} OR &= \frac{odds(1)}{odds(0)} = \frac{\frac{d}{b}}{\frac{a}{c}} = \frac{d \times c}{b \times a} \\ &= \frac{96 \times 206}{64 \times 96} = 3.22 \end{aligned}$$

```
# odds ratio
num_or = contingence.values[1,1]*contingence.values[0,0]
deno_or = contingence.values[1,0]*contingence.values[0,1]
odds_ratio = num_or/deno_or
print('odds ratio : %.2f' % (odds_ratio))

odds ratio : 3.22
```

Definition 13.5 (log odds-ratio) Il s'agit simplement du logarithme de l'odds-ratio. Développons son expression, nous verrons ainsi le rapport avec la régression logistique :

$$\begin{aligned}
 \ln(OR) &= \ln\left(\frac{\text{odds}(1)}{\text{odds}(0)}\right) \\
 &= \ln(\text{odds}(1)) - \ln(\text{odds}(0)) \\
 &= \ln\left(\frac{\mathbb{P}(Y = 1/X = 1)}{\mathbb{P}(Y = 0/X = 1)}\right) - \ln\left(\frac{\mathbb{P}(Y = 1/X = 0)}{\mathbb{P}(Y = 0/X = 0)}\right) \\
 &= \ln\left(\frac{\mathbb{P}(Y = 1/X = 1)}{1 - \mathbb{P}(Y = 1/X = 1)}\right) - \ln\left(\frac{\mathbb{P}(Y = 1/X = 0)}{1 - \mathbb{P}(Y = 1/X = 0)}\right) \\
 &= \text{logit}(1) - \text{logit}(0)
 \end{aligned}$$

On constate que le log-odds ratio peut s'interpréter comme un écart entre 2 logit.

2. Cas de la régression simple

On suppose que notre variable cible Y est expliquée par une seule variable explicative X. On a le modèle suivant :

$$\text{logit}p_\beta(X) = \beta_0 + \beta_1 X$$

L'interprétation des coefficients dépend du type de la variable explicative X ;

Cas 1 : X est une variable explicative binaire

Ce cas est en relation directe avec le tableau 13.1. Dans cette configuration, le coefficient β_1 correspond au logarithme de l'odds-ratio calculé à partir du tableau de contingence 13.1. L'idée est relativement simple :

$$\begin{aligned}
 X = 1 \longrightarrow \text{logit}(1) &= \beta_0 + \beta_1 \times 1 &= \beta_0 + \beta_1 \\
 X = 0 \longrightarrow \text{logit}(0) &= \beta_0 + \beta_1 \times 0 &= \beta_0 \\
 \implies \ln(OR) &= \text{logit}(1) - \text{logit}(0) &= \beta_1 \\
 \implies OR &= \exp(\beta_1)
 \end{aligned}$$

Application :

En reprenant l'exemple croisant `chd` et `famhist`, l'odds-ratio était égal à $OR = 3.22$. En utilisant, le modèle de la section 13.3.2, nous obtenons $\hat{\beta}_1 = 1.1690$. En prenant l'exponentielle, nous obtenons :

```
# odds ratio
OR = np.exp(lowresult2.params[1])
print('OR : %.2f' % (OR))

OR : 3.22
```

Ainsi, la régression logistique nous permet de mesurer directement le surcroît de risque associé à un facteur explicatif binaire :

- Si $\hat{\beta}_j < 0 \longrightarrow OR < 1$: il y a une diminution du risque ;
- Si $\hat{\beta}_j > 0 \longrightarrow OR > 1$: il y a une augmentation du risque ;

Cas 2 : X est une variable explicative quantitative

Pour comprendre l'interprétation des coefficients dans le cas d'une variable explicative quantitative, voyons l'évolution du logit lorsqu'on fait varier X d'une unité.

$$\begin{aligned}\text{logit}(X+1) &= \beta_0 + \beta_1 \times (X+1) = \beta_0 + \beta_1 X + \beta_1 \\ \text{logit}(X) &= \beta_0 + \beta_1 X \\ \implies \text{logit}(X+1) - \text{logit}(X) &= \beta_1\end{aligned}$$

Dans ce cas, la quantité $\exp(\beta_1)$ s'interprète comme l'odds ratio consécutif à l'augmentation d'une unité de la variable explicative. Par conséquent, si l'on augmente de b unités la variable explicative, l'odds-ratio devient alors $\exp(b \times \beta_1)$.

Application :

En reprenant l'exemple de la section 13.3.2 où nous avons essayé de prédire `chd` en fonction de `age`, nous avons obtenu $\hat{\beta}_1 = 0.0641$ et par conséquent :

```
# odds ratio
OR = np.exp(lowresult.params[1])
print('OR : %.2f' % (OR))

OR : 1.07
```

13.3.4 Coefficients standardisés en régression logistique

Lorsque les explicatives sont exclusivement quantitatives, il peut être intéressant de comparer leur impact sur la variable dépendante ; Quelle est celle qui joue le rôle le plus important ? Dans quel sens ?.

Comparer les odds-ratio paraît une solution immédiate. Mais comme les explicatives ne sont pas exprimées sur une même échelle, la variation d'une unité n'a absolument pas la même signification d'une variable à l'autre. Les odds ne sont pas comparable en l'état. La solution la plus simple est de centrer et réduire les explicatives. Ainsi, nous pouvons mieux jauger leur influence et, de plus, nous pouvons disposer d'interprétations sous forme de variations d'écart-type.

Dans cette section, nous souhaitons mettre en place un dispositif qui permet de :

- Comparer les influences respectives des variables explicatives ;
- Mesurer l'impact de la variation d'un écart-type d'une variable explicative sur le logit, soit en termes **absolus** c'est-à-dire écarts absolu entre logit (l'exponentielle de l'écart entre deux logit est un odds-ratio), soit en termes **relatifs**, c'est-à-dire variation en écarts-type du logit.

Modélisation avec uniquement les variables explicatives quantitatives

```
# Selection des variables explicatives quantitatives
donnee.famhist = donnee.famhist.astype('category')
Xquant = donnee.drop('chd', axis=1).select_dtypes(np.number)
print(Xquant.info())
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 462 entries, 1 to 463
Data columns (total 8 columns):
 Column    Non-Null Count Dtype  
---      
0   sbp      462 non-null   int64  
1   tobacco   462 non-null   float64 
2   ldl      462 non-null   float64 
3   adiposity 462 non-null   float64 
4   typea    462 non-null   int64  
5   obesity   462 non-null   float64 
6   alcohol   462 non-null   float64 
7   age      462 non-null   int64  
dtypes: float64(5), int64(3)
memory usage: 32.5 KB

```

On garde les 8 variables explicatives quantitatives. On entraîne notre modèle

```

# Importation de la classe de calcul
import statsmodels.api as sm
# on ajuste le modèle
fullmodel2 = sm.Logit(y,sm.add_constant(Xquant))
# lancer les calculs
fullresult2 = fullmodel2.fit()
# résumé des résultats
print(fullresult2.summary())

```

Optimization terminated successfully.

```

    Current function value: 0.529096
    Iterations 6

```

Logit Regression Results

Dep. Variable:	chd	No. Observations:	462			
Model:	Logit	Df Residuals:	453			
Method:	MLE	Df Model:	8			
Date:	Fri, 22 Oct 2021	Pseudo R-squ.:	0.1799			
Time:	00:03:08	Log-Likelihood:	-244.44			
converged:	True	LL-Null:	-298.05			
Covariance Type:	nonrobust	LLR p-value:	1.415e-19			
	coef	std err	z	P> z	[0.025	0.975]
const	-6.0669	1.272	-4.771	0.000	-8.559	-3.575
sbp	0.0056	0.006	1.005	0.315	-0.005	0.017
tobacco	0.0727	0.026	2.762	0.006	0.021	0.124
ldl	0.1925	0.059	3.239	0.001	0.076	0.309
adiposity	0.0171	0.028	0.600	0.548	-0.039	0.073
typea	0.0405	0.012	3.350	0.001	0.017	0.064
obesity	-0.0579	0.043	-1.348	0.178	-0.142	0.026
alcohol	0.0014	0.004	0.328	0.743	-0.007	0.010

age	0.0507	0.012	4.304	0.000	0.028	0.074
-----	--------	-------	-------	-------	-------	-------

Cas 1 : Standardisation sur les explicatives seulement

Nous calculons le coefficient standardisé de la manière suivante :

$$\hat{\beta}_j^{\text{std.1}} = \hat{\beta}_j \times \hat{\sigma}_j \quad (13.8)$$

où $\hat{\sigma}_j$ est l'écart type empirique lié à la variable explicative quantitative X_j .

```
# Coefficients standardisés
params_std1 = fullresult2.params[-1]*Xquant.std()
print(params_std1)
```

sbp	1.038145
tobacco	0.232638
ldl	0.104892
adiposity	0.394095
typea	0.497261
obesity	0.213424
alcohol	1.239974
age	0.739948

dtype: float64

Cas 2 : Standardisation sur les explicatives et l'écart-type du logit

Une autre standardisation est proposée dans la littérature :

$$\hat{\beta}_j^{\text{std.2}} = \hat{\beta}_j \times \frac{\hat{\sigma}_j}{\hat{\sigma}_{\text{logit}}} \quad (13.9)$$

```
# Coefficients standardisés
params_std2 = params_std1/fullresult2.fittedvalues.std()
print(params_std2)

sbp          0.845657
tobacco      0.189503
ldl          0.085444
adiposity    0.321023
typea        0.405061
obesity      0.173852
alcohol       1.010063
age           0.602750
dtype: float64
```

Cas 3 : Standardisation sur les variables explicatives et l'écart-type théorique de la loi de répartition logistique

La dernière standardisation vaut surtout parce qu'elle est proposée dans le logiciel SAS.

$$\hat{\beta}_j^{\text{std.1}} = \hat{\beta}_j \times \frac{\hat{\sigma}_j}{\hat{\sigma}_{\text{théorique}}}$$

```
# Coefficients standardisés
```

Comme pour toutes les autres standardisations, les coefficients permettent de comparer l'impact des explicatives. Mais elles ne s'interprètent pas en termes de variation du logit.

13.4 Précision des estimateurs

13.4.1 Loi asymptotique des estimateurs

Le comportement asymptotique de l'estimateur du maximum de vraisemblance $\hat{\beta}$ est :

$$\sqrt{n}(\hat{\beta} - \beta) \xrightarrow{\mathcal{L}} \mathcal{N}(0, \mathcal{I}(\beta)^{-1})$$

où \mathcal{I} est la matrice d'information de Fisher au point β . On déduit :

$$(\hat{\beta} - \beta)' n \mathcal{I}(\beta) (\hat{\beta} - \beta) \xrightarrow{\mathcal{L}} \chi_{p+1}^2$$

Un tel résultat n'est pas utilisable tel quel puisque la matrice $\mathcal{I}(\beta)$ est inconnue. On remarque que d'après la loi des grands nombres

$$\begin{aligned} \hat{\mathcal{I}}_{kl} &= -\frac{1}{n} \sum_{i=1}^n \frac{\partial^2}{\partial \beta_k \partial \beta_l} \mathcal{L}_1(Y_i; \beta) = \frac{1}{n} \frac{\partial^2}{\partial \beta_k \partial \beta_l} \sum_{i=1}^n \mathcal{L}_1(Y_i; \beta) = \frac{1}{n} \frac{\partial^2}{\partial \beta_k \partial \beta_l} \sum_{i=1}^n \mathcal{L}_n(Y_1, \dots, Y_n; \beta) \\ &= \frac{1}{n} (X' W_\beta X)_{kl} \end{aligned}$$

converge presque sûrement vers $\mathcal{I}(\beta)_{kl}$.

Comme $\hat{\beta}$ converge faiblement vers β , on obtient grâce au théorème de Slutsky et aux opérations classiques sur la convergence en loi

$$(\hat{\beta} - \beta)' \hat{\Sigma} (\hat{\beta} - \beta) \xrightarrow{\mathcal{L}} \chi_{p+1}^2 \quad (13.10)$$

avec $\hat{\Sigma} = (X' W_{\hat{\beta}} X)$

13.4.2 Intervalle de confiance

On déduit du paragraphe précédent qu'un estimateur de la variance de $\hat{\beta}_j$ est donné par le $j^{\text{ème}}$ terme de la diagonale de $\hat{\Sigma}^{-1}$. Notons par $\hat{\sigma}_j^2$ cet estimateur. Il vient :

$$\frac{(\hat{\beta}_j - \beta_j)^2}{\hat{\sigma}_j^2} \xrightarrow{\mathcal{L}} \chi_1^2 \quad \text{ou encore} \quad \frac{\hat{\beta}_j - \beta_j}{\hat{\sigma}_j} \xrightarrow{\mathcal{L}} \mathcal{N}(0, 1) \quad (13.11)$$

Un intervalle de confiance (asymptotique) de niveau $1 - \alpha$ pour β_j est donné par :

$$IC_{16\alpha}(\beta_j) = \left[\hat{\beta}_j - z_{1-\alpha/2} \times \hat{\sigma}_j; \hat{\beta}_j + z_{1-\alpha/2} \times \hat{\sigma}_j \right]$$

où $z_{1-\alpha/2}$ représente le quantile de niveau $1 - \frac{\alpha}{2}$ de la loi normale $\mathcal{N}(0, 1)$.

Pour obtenir des intervalles de confiance à 95% des coefficients de notre modèle global, nous faisons appel à la méthode `conf_int()` de l'objet résultat fourni par `fit()`.

```
# intervalle de confiance des coefficients à 95%
print(fullresult.conf_int(alpha=0.05))
```

	0	1
const	-8.714863	-3.586578
sbp	-0.004727	0.017735
tobacco	0.027236	0.131517
ldl	0.056989	0.290859
adiposity	-0.038820	0.075993
famhist	0.478706	1.372034
typea	0.015448	0.063742
obesity	-0.149634	0.023814
alcohol	-0.008665	0.008909
age	0.021451	0.068999

Remarque :

La validité de ces intervalles est relative puisqu'il s'agit d'une approximation valable asymptotiquement. Il est toujours possible de compléter cette étude par un bootstrap afin d'obtenir d'autres intervalles de confiance dans le cas où ceux-ci sont parfaitement importants. Cela dit, en pratique, on se contente de l'intervalle de confiance bâti grâce à la matrice d'information de Fisher.

On déduit également de l'équation 13.11 les tests de nullité des coefficients du modèle. On note :

$$\begin{cases} H_0 : \beta_j = 0 \\ H_1 : \beta_j \neq 0 \end{cases}$$

Alors sous H_0 :

$$\frac{\hat{\beta}_j}{\hat{\sigma}_j} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1)$$

On rejette H_0 si la valeur observée de $\frac{\hat{\beta}_j}{\hat{\sigma}_j}$ dépasse en valeur absolue le quantile d'ordre $1-\alpha/2$ de la loi $\mathcal{N}(0,1)$

13.4.3 Test de nullité de q coefficients libres

La théorie du maximum de vraisemblance nous donnant la loi (asymptotique) des estimateurs, il est possible de tester la significativité des variables explicatives. Pour cela, trois tests sont généralement utilisés :

- Le test de Wald ;
- Le test du rapport de vraisemblance ou de la déviance.
- Le test du score.

Les hypothèses s'écrivent :

$$\begin{cases} H_0 : \beta_{j1} = \beta_{j2} = \cdots = \beta_{jq} = 0 \\ H_1 : \exists k \in 1, \dots, q : \beta_{jk} \neq 0 \end{cases}$$

Pour alléger les notations, nous supposons sans perte de généralité que nous testons la nullité des q premiers coefficients du modèle :

$$\begin{cases} H_0 : \beta_0 = \beta_1 = \dots = \beta_{q-1} = 0 \\ H_1 : \exists k \in \{0, \dots, q-1\} : \beta_k \neq 0 \end{cases}$$

Pour ces 3 tests, on rejette l'hypothèse nulle si la valeur observée de la statistique de test dépasse le quantile d'ordre $1 - \alpha$ de la loi de \mathcal{X}_q^2 .

1. Test de Wald

Il est basé sur l'équation 13.10. On note $\beta_{0,\dots,q-1}$ le vecteur composé des q premières composantes de β et $\hat{\Sigma}_{0,\dots,q}^{-1}$ la matrice bloc composée des q premières lignes et colonnes de $\hat{\Sigma}^{-1}$. Il est facile de voir que sous H_0 :

$$\hat{\beta}'_{0,\dots,q-1} \hat{\Sigma}_{0,\dots,q}^{-1} \hat{\beta}_{0,\dots,q-1} \xrightarrow{\mathcal{L}} \mathcal{X}_q^2$$

Application :

Nous testons la nullité des coefficients de (sbp, adiposity, obesity, alcohol). Tout d'abord nous affichons la matrice des variances - covariances des coefficients estimés.

```
# Matrice - covariance des coefficients estimés
cov_params = fullresult.cov_params()
print(cov_params)
```

	const	sbp	tobacco	ldl	adiposity	famhist	\
const	1.711544	-0.003605	-0.000076	-0.006260	0.012046	-0.016712	
sbp	-0.003605	0.000033	0.000005	-0.000005	-0.000010	0.000065	
tobacco	-0.000076	0.000005	0.000708	-0.000041	-0.000006	0.000560	
ldl	-0.006260	-0.000005	-0.000041	0.003560	-0.000380	-0.000599	
adiposity	0.012046	-0.000010	-0.000006	-0.000380	0.000858	0.000145	
famhist	-0.016712	0.000065	0.000560	-0.000599	0.000145	0.051936	
typea	-0.008840	0.000004	0.000010	-0.000006	0.000032	0.000041	
obesity	-0.027749	-0.000011	0.000016	-0.000079	-0.000962	-0.000461	
alcohol	-0.000111	-0.000003	-0.000023	0.000025	-0.000007	-0.000076	
age	-0.005663	-0.000013	-0.000085	0.000034	-0.000172	-0.000219	
	typea	obesity	alcohol	age			
const	-0.008840	-0.027749	-0.000111	-0.005663			
sbp	0.000004	-0.000011	-0.000003	-0.000013			
tobacco	0.000010	0.000016	-0.000023	-0.000085			
ldl	-0.000006	-0.000079	0.000025	0.000034			
adiposity	0.000032	-0.000962	-0.000007	-0.000172			
famhist	0.000041	-0.000461	-0.000076	-0.000219			
typea	0.000152	-0.000067	-0.000001	0.000022			
obesity	-0.000067	0.001958	0.000008	0.000161			
alcohol	-0.000001	0.000008	0.000020	0.000006			
age	0.000022	0.000161	0.000006	0.000147			

Nous affichons ensuite les écarts-types des coefficients estimés.

```
# Ecarts-types des coefficients estimés
std_params = np.sqrt(np.diag(cov_params))
print(std_params)

[1.30826006 0.0057304 0.02660284 0.05966174 0.02928941 0.22789401
 0.01232023 0.04424774 0.00448322 0.01212975]
```

Statsmodels renvoie les écarts-types des coefficients estimés grâce à l'outil bse

```
# Ecart-type des coefficients estimés
print(fullresult.bse)
```

```
const      1.308260
sbp        0.005730
tobacco    0.026603
ldl        0.059662
adiposity   0.029289
famhist     0.227894
typea       0.012320
obesity     0.044248
alcohol     0.004483
age         0.012130
dtype: float64
```

Nous récupérons ensuite la sous-partie de la matrice de variance-covariance qui nous concerne :

```
# indice des coefficients concernés
index = [1,4,7,8]
# Sous-matrice
cov_subparams = np.zeros(shape=(4,4))
for i in range(len(index)):
    for j in range(len(index)):
        cov_subparams[i,j] = cov_params.values[index[i],index[j]]
# Vérification
print(cov_subparams)

[[ 3.28374597e-05 -9.98771454e-06 -1.14446051e-05 -3.25941453e-06]
 [-9.98771454e-06  8.57869499e-04 -9.62074113e-04 -7.32511986e-06]
 [-1.14446051e-05 -9.62074113e-04  1.95786278e-03  7.71384474e-06]
 [-3.25941453e-06 -7.32511986e-06  7.71384474e-06  2.00992466e-05]]
```

Nous inversons cette sous-matrice :

```
# Inversion de cette matrice
inv_cov_subparams = np.linalg.inv(cov_subparams)
print(inv_cov_subparams)

[[31670.56009035 1332.54203622 819.01670525 5307.20123261]
 [1332.54203622 2656.8916662 1310.67465765 681.36961829]
 [819.01670525 1310.67465765 1158.94807814 165.69866725]
 [5307.20123261 681.36961829 165.69866725 50798.48671085]]
```

Nous récupérons également le sous-vecteur des coefficients estimés :

```
# Coefficients estimés
sub_params = fullresult.params[index]
print(sub_params)

sbp      0.006504
adiposity 0.018587
obesity   -0.062910
alcohol    0.000122
dtype: float64
```

Nous calculons la forme quadratique correspondant à la statistique de test ainsi que le p-value :

```
# Statistique de test (Forme quadratique)
wald_stat = np.dot(sub_params, np.dot(inv_cov_subparams, sub_params))
print('Statistique de test de Wald : %.4f' % (wald_stat))
# p-value avec ddl=len(index)
import scipy.stats as stats
pvalue = 1.0 - stats.chi2.cdf(wald_stat, len(index))
print('p-value associé : %.4f' % (pvalue))

Statistique de test de Wald : 3.4409
p-value associé : 0.4869
```

Statsmodels propose l'outil `wald_test()` pour réaliser les tests généralisés. L'enjeu réside alors dans la définition de la matrice **R** permettant de spécifier les coefficients à tester :

- sbd est en position 1 ;
- adiposity en position 4 ;
- obesity est en position 7 ;
- alcohol est en position 8.

```
# Matrice des coefficients à tester
R = np.array([[0,1,0,0,0,0,0,0,0], 
              [0,0,0,0,1,0,0,0,0], 
              [0,0,0,0,0,0,0,1,0], 
              [0,0,0,0,0,0,0,0,1]])
# Wald test
wald_test = fullresult.wald_test(R)
print(wald_test)

<Wald test (chi2): statistic=[[3.44085405]], p-value=0.48692841280971455,
df_denom=4>
```

Les résultats sont complètement cohérents.

2. Test du rapport de vraisemblance ou de la déviance

La statistique de test est basée sur la différence des rapports de vraisemblance entre le modèle complet et le modèle sous H_0 . On note $\hat{\beta}_{H_0}$ l'estimateur du maximum de vraisemblance contraint par H_0 (il s'obtient en supprimant les q premières variables du modèle). On a alors sous H_0 :

$$2 \left(\mathcal{L}_n(\hat{\beta}) - \mathcal{L}_n(\hat{\beta}_{H_0}) \right) \xrightarrow{\mathcal{L}} \chi_q^2$$

Application :

On définit une nouvelle matrice X sans les variables sbp, adiposity, obesity et alcohol.

```
# Matrice X sans les variables sbp, adiposity, obesity et alcohol
Xbis = X.drop(['sbp', 'adiposity', 'obesity', 'alcohol'], axis=1)
print(Xbis.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 462 entries, 1 to 463
Data columns (total 5 columns):
 Column    Non-Null Count  Dtype  
---      
0  tobacco   462 non-null   float64 
1  ldl       462 non-null   float64 
2  famhist   462 non-null   int8    
3  typea    462 non-null   int64  
4  age       462 non-null   int64  
dtypes: float64(2), int64(2), int8(1)
memory usage: 18.5 KB
None
```

Nous réalisons ensuite la régression sans les 4 variables.

```
# Régression sans les 4 variables
# Importation de la classe de calcul
import statsmodels.api as sm
# on ajuste le modèle
fullmodel3 = sm.Logit(y,sm.add_constant(Xbis))
# lancer les calculs
fullresult3 = fullmodel3.fit()
# résumé des résultats
print(fullresult3.summary())
```

Optimization terminated successfully.

```
    Current function value: 0.514811
    Iterations 6
```

Logit Regression Results

Dep. Variable:	chd	No. Observations:	462
Model:	Logit	Df Residuals:	456
Method:	MLE	Df Model:	5
Date:	Fri, 22 Oct 2021	Pseudo R-squ.:	0.2020

Time:	22:58:54	Log-Likelihood:	-237.84			
converged:	True	LL-Null:	-298.05			
Covariance Type:	nonrobust	LLR p-value:	2.554e-24			
<hr/>						
	coef	std err	z	P> z	[0.025	0.975]
const	-6.4464	0.921	-7.000	0.000	-8.251	-4.642
tobacco	0.0804	0.026	3.106	0.002	0.030	0.131
ldl	0.1620	0.055	2.947	0.003	0.054	0.270
famhist	0.9082	0.226	4.023	0.000	0.466	1.351
typea	0.0371	0.012	3.051	0.002	0.013	0.061
age	0.0505	0.010	4.944	0.000	0.030	0.070

Nous pouvons par la suite réaliser le test en calculant la statistique du rapport de vraisemblance.

```
# Statistique de test
deviance_test = 2*(fullresult.llf - fullresult3.llf)
print('Statistique du test : %.4f' %(deviance_test))
# Degré de liberté
ddl = fullresult3.df_resid - fullresult.df_resid
print('Degré de liberté du test : %.i' %(ddl))
# p-value associée
pvalue = 1 - stats.chi2.cdf(deviance_test, ddl)
print('p-value associée : %.4f' %(pvalue))

Statistique du test : 3.5455
Degré de liberté du test : 4
p-value associée : 0.4710
```

3. Test du score

On cherche ici à vérifier si la fonction de score (gradient de la log-vraisemblance) est proche de 0 sous H_0 . Sous H_0 , on a :

$$S(\hat{\beta}_{H_0})' \hat{\Sigma}_{H_0}^{-1} S(\hat{\beta}_{H_0}) \xrightarrow{\mathcal{L}} \chi_q^2$$

où $\hat{\Sigma}_{H_0} = X' W_{\hat{\beta}_{H_0}} X$.

Remarque :

En définitive :

- Le test du score revient à tester la nullité de la pente en $\hat{\beta}_{H_0}$ ($\hat{\beta}$ sou H_0) ;
- Le test de Wald la nullité de la distance entre $\hat{\beta}$ et $\hat{\beta}_{H_0}$;
- Le test du rapport de vraisemblance la nullité de la différence entre les vraisemblances en ces deux points.

13.5 Première évaluation de la régression logistique : les pseudo- R^2

Une question cruciale est de pouvoir déterminer si le modèle obtenu est « intéressant » ou non.

13.5.1 Estimation du paramètre β_0 et de la déviance du modèle trivial

Rappelons que dans le cadre binaire, pour un individu donné, sa probabilité a priori d'être positif s'écrit $\mathbb{P}[Y(\omega) = +] = \pi(\omega)$. Lorsqu'il ne peut y avoir d'ambiguités, nous la noterons simple π .

Lorsque l'échantillon est issu d'un tirage aléatoire dans la population, sans distinction des classes d'appartenance, si n_+ est le nombre d'observations positives dans Ω , π peut être estimée par $\frac{n_+}{n_-}$. On parle alors de **schéma de mélange**.

On rappelle également que la **probabilité a posteriori** d'un individu ω d'être positif c'est-à-dire sachant les valeurs prises par les descripteurs est notée $\mathbb{P}[Y(\omega) = + | X(\omega)] = p(\omega)$. Sans confusion, nous avons noté p .

Le modèle trivial est réduit à la seule constante c'est-à-dire :

$$\text{logit}(M_0) = \ln\left(\frac{p}{1-p}\right) = \beta_0$$

Nous ne tenons pas compte des variables explicatives X_j . De fait :

$$\frac{p}{1-p} = \frac{\pi}{1-\pi} \times \frac{\mathbb{P}(X/Y = +)}{\mathbb{P}(X/Y = -)} = \frac{\pi}{1-\pi}$$

On devine aisément l'**estimateur** $\hat{\beta}_0$ de la régression :

$$\hat{\beta}_0 = \ln\left(\frac{\hat{\pi}}{1-\hat{\pi}}\right) = \ln\left(\frac{n_+}{n_-}\right)$$

Le nombre de positifs n_+ et négatifs n_- dans l'échantillon suffit pour estimer le paramètre du modèle trivial. Pour prédire la probabilité a posteriori pour un individu d'être positif $\hat{p}(\omega)$, nous utilisons simplement la proportion des positifs $\hat{\pi} = \frac{n_+}{n_-}$ dans la base, soit :

$$\hat{p}(\omega) = \hat{\pi}, \forall \omega$$

Réécrivons la log-vraisemblance du modèle trivial, nous obtenons :

$$\begin{aligned} \mathcal{L}_0 &= \sum_{\omega} [Y \times \ln(\hat{\pi}) + (1 - Y) \times \ln(1 - \hat{\pi})] \\ &= \sum_{\omega} Y \times \ln(\hat{\pi}) + \sum_{\omega} (1 - Y) \times \ln(1 - \hat{\pi}) \\ &= n_+ \times \ln(\hat{\pi}) + n_- \times \ln(1 - \hat{\pi}) \\ &= n \times \ln(1 - \hat{\pi}) + n_+ \ln\left(\frac{\hat{\pi}}{1 - \hat{\pi}}\right) \end{aligned}$$

Nous pouvons en déduire la **déviance** du modèle trivial :

$$D_0 = -2 \times \mathcal{L}_0$$

Estimation directe

Reprendons l'exemple de notre base de données :

```
# Effectifs théoriques
print(donnee.chd.value_counts())

0    302
1    160
Name: chd, dtype: int64
```

Nous y retrouvons $n_+ = 160$ observations positives parmi $n = 462$. Nous obtenons directement :

- Le nombre de négatifs $n_- = n - n_+ = 302$
- La proportion de positifs $\hat{\pi} = \frac{160}{462} = 0.3463$
- L'estimation de la constante $\beta_0 = \ln\left(\frac{n_+}{n_-}\right) = \ln\left(\frac{160}{302}\right) = -0.6353$
- La log-vraisemblance $\mathcal{L}_0 = 462 \times \ln(1 - 0.3463) + 160 \times \ln\left(\frac{0.3463}{1 - 0.3463}\right) = -298.05$
- La déviance $D_0 = -2 \times \mathcal{L}_0 = -2 \times (-298.05) = 596.1$

Estimation usuelle

Par curiosité, nous souhaitons vérifier si les résultats de l'estimation directe concordent avec ceux de la procédure usuelle en utilisant Python.

```
# création du vecteur constant
const = pd.DataFrame(np.ones(len(donnee.chd)), columns=['const'],
                      index=donnee.index)

# Importation de la classe de calcul
import statsmodels.api as sm
# on ajuste le modèle
nullmodel = sm.Logit(y,const)
# lancer les calculs
nullresult = nullmodel.fit()
# résumé des résultats
print(nullresult.summary())
```

```
Optimization terminated successfully.
      Current function value: 0.645139
      Iterations 4
```

Logit Regression Results

Dep. Variable:	chd	No. Observations:	462
Model:	Logit	Df Residuals:	461
Method:	MLE	Df Model:	0
Date:	Mon, 18 Oct 2021	Pseudo R-squ.:	1.799e-12

Time:	22:48:43	Log-Likelihood:	-298.05		
converged:	True	LL-Null:	-298.05		
Covariance Type:	nonrobust	LLR p-value:	nan		
<hr/>					
coef	std err	z	P> z	[0.025	0.975]
const	-0.6353	0.098	-6.497	0.000	-0.827 -0.444
<hr/>					

Les résultats sont totalement cohérents avec l'approche directe : l'estimation $\hat{\beta}_0 = -0.6353$ et la log-vraisemblance $\mathcal{L}_0 = -298.05$. Ce qui est plutôt encourageant. Le calcul direct nous épargne une optimisation compliquée

13.5.2 Quelques pseudo- R^2

Les pseudo- R^2 sont des indicateurs similaires dans l'esprit au R^2 de la régression linéaire. Ils sont basés sur la confrontation entre la log-vraisemblance du modèle et de celle du modèle trivial composé uniquement de la constante (`null model`). *Grosso modo*, il s'agit de varifier si notre modèle fait mieux que le modèle trivial c'est-à-dire s'il présente une vraisemblance ou une log-vraisemblance plus favorable.

L'analogie avec le coefficient de détermination R^2 de la régression linéaire multiple est tout à fait intéressante. En effet, il est usuellement interprété comme **la part de la variance expliquée par le modèle**. Mais il peut être également compris comme une confrontation entre les performances du modèle analysé (traduite par la somme des carrés des résidus $SCR = \sum_{\omega} (Y - \hat{Y})^2$)

et celles du modèle par défaut réduite à la simple constante (dans ce cas, la constante est estimée par la moyenne de l'endogène \bar{Y} , la somme des carrés totaux correspond donc à la somme des carrés des résidus du modèle réduit à la simple constante $SCT = \sum_{\omega} (Y - \bar{Y})^2$). N'oublions pas

que $R^2 = \frac{SCT - SCR}{SCT} = 1 - \frac{SCR}{SCT}$. Sa définition répond exactement à la notion d'efficacité prédictive. Plusieurs formes de pseudo- R^2 sont proposées dans la littérature.

1. R^2 de McFadden

Sa formule est donnée par :

$$R_{MF}^2 = 1 - \frac{\mathcal{L}_n}{\mathcal{L}_0}$$

où \mathcal{L}_n et \mathcal{L}_0 représentent respectivement la log-vraisemblance du modèle étudié et du modèle trivial.

Remarque :

- $R_{MF}^2 = 0$ si $\mathcal{L}_n = \mathcal{L}_0$, on ne fait pas mieux que le modèle trivial ;
- $R_{MF}^2 = 1$ si $\mathcal{L}_n = 0$, notre modèle est parfait ;
- L'analogie avec le R^2 de la régression linéaire multiple est totale.

```

#accès à la log-vraisemblance du modèle
print("Log-vraisemblance du modèle : %.2f" % (fullresult.llf))
#log-vraisemblance du null modèle
print("Log-vraisemblance du null modèle : %.2f" % (fullresult.llnull))

Log-vraisemblance du modèle : -236.07
Log-vraisemblance du null modèle : -298.05

```

On applique la formule de calcul :

```

#R2 de McFadden
r2Mcfadden = 1 - fullresult.llf / fullresult.llnull
print("R2 de McFadden : %.2f" % (r2Mcfadden))

R2 de McFadden : 0.21

```

L'outil fourni directement la valeur du R^2 de McFadden.

```

#qui est fourni directement par l'outil
print("R2 de McFadden : %.2f" % (fullresult.prsquared))

R2 de McFadden : 0.21

```

2. R^2 de Cox and Snell

Sa formule de calcul est donnée par :

$$R_{CS}^2 = 1 - \left(\frac{L_0}{L_n} \right)^n$$

où L_n et L_0 représentent respectivement la vraisemblance du modèle étudié et du modèle trivial. Sa valeur minimale est égale à 0 et sa valeur maximale est atteinte lorsque L_n vaut 1, c'est-à-dire $\max[R_{CS}^2] = 1 - L_0^n$. L'indicateur n'est pas normalisé, ce qui est un peu gênant.

```

# Vraisemblance du modèle null
L0 = np.exp(fullresult.llnull)
# Vraisemblance du modèle étudié
Ln = np.exp(fullresult.llf)
# R2 de Cox and Snell
r2CoxSnell = 1.0 - (L0/Ln)**(2.0/donnee.shape[0])
print("R2 de Cox - Snell : %.2f" % (r2CoxSnell))

R2 de Cox - Snell : 0.24

```

3. R^2 de Nagelkerke

Sa formule de calcul est :

$$R_N^2 = \frac{R_{CS}^2}{\max[R_{CS}^2]} = \frac{\frac{R_{CS}^2}{2}}{1 - L_0^n}$$

C'est un indicateur compris entre 0 et 1. Le R^2 de Nagelkerke est une simple normalisation du R^2 de Cox and Snell.

```
# max du R2 de Cox and Snell
maxR2CoxSnell = 1.0 - (L0)**(2.0/len(y))
# R2 de Nagelkerke
r2Nagelkerke = r2CoxSnell / maxR2CoxSnell
print("R2 de Nagelkerke : %.2f" % (r2Nagelkerke))

R2 de Nagelkerke : 0.32
```

CHAPITRE 14

SÉLECTION ET VALIDATION DES MODÈLES

Sommaire

14.1 Sélection ou choix de modèle	320
14.2 Validation du modèle	328

14.1 Sélection ou choix de modèle

Si on se restreint à des modèles logistiques, sélectionner un modèle revient à choisir les variables qui vont constituer le modèle.

14.1.1 La déviance

Bien souvent, on utilise la quantité

$$D_M = -2\mathcal{L}_n(\hat{\beta})$$

est appelée **déviance** (ou déviance résiduelle, en anglais *residual deviance*). Contrairement à la log-vraisemblance, elle est positive. L'objectif de l'algorithme d'optimisation est de minimiser cette déviance. On peut faire le parallèle avec la **somme des carrés des résidus** de la régression linéaire multiple. La **null deviance** D_0 calculée sur le modèle uniquement composée de la **constante** correspondrait à la **somme des carrés totaux**.

Exemple 14.1 (Calcul de déviance)

Pour illustrer nos propos, on extrait la log-vraisemblance issue de notre modèle précédent.

```
#accès à la log-vraisemblance du modèle
print("Log-vraisemblance du modèle : %.2f" % (fullresult.llf))

Log-vraisemblance du modèle : -236.07
```

On applique ensuite la formule de la déviance

```
# Déviance
deviance = -2*fullresult.llf
print('Deviance : %.2f' % (deviance))

Deviance : 472.14
```

Remarque :

Dans certains ouvrages, on définit la déviance D de manière plus générique :

$$\begin{aligned} D &= 2 \times \ln \left[\frac{L(\text{Modèle saturé})}{L(\text{Modèle étudié})} \right] \\ &= -2 \times \hat{\mathcal{L}}(\text{Modèle étudié}) - [-2 \times \mathcal{L}(\text{Modèle saturé})] \\ &= D_M - [-2 \times \mathcal{L}(\text{Modèle saturé})] \end{aligned}$$

Un modèle saturé pour des données individuelles est un modèle reconstituant parfaitement les valeurs de la variable dépendante. Sa vraisemblance est égale à 1 et sa log-vraisemblance 0. Dans ce contexte, $D = D_M$.

14.1.2 Test de déviance entre 2 modèles emboîtés

Rappelons que par définition un modèle emboîté dans un autre plus général (ou plus grand) lorsqu'il est un cas particulier de ce modèle en général.

Exemple 14.2 *Les modèles logistiques définis par*

$$\text{logit}(p_\beta(x)) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

et

$$\text{logit}(p_\beta(x)) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

sont emboîtés l'un dans l'autre.

En effet, il est facile de voir que faire un test entre modèles emboîtés est équivalent à tester la nullité de certains coefficients du grand modèle. On peut ainsi, les tests de Wald, du maximum de vraisemblance ou du score pour tester deux modèles emboîtés. Si par exemple $\mathcal{M}_1 \mathcal{M}_2$, alors on a :

$$-2 \left(\mathcal{L}_n(\hat{\mathcal{M}}_1) - \mathcal{L}_n(\hat{\mathcal{M}}_2) \right) \xrightarrow{\mathcal{L}} \chi_{p_2-p_1}$$

où p_j désigne la dimension du modèle M_j .

Remarque :

La statistique de test peut s'écrire $D_{M_1} - D_{M_2}$, c'est pourquoi ce test est également appelé test de déviance.

14.1.3 Critère de choix de modèle

Le test que nous venons d'étudier permet de sélectionner un modèle parmi deux modèles emboîtés. Or, à partir de p variables explicatives, il est possible de définir un grand nombre de modèles logistiques qui ne sont pas forcément emboîtés. L'utilisation d'un simple test de déviance se révèle alors insuffisante. On a recours à des critères de choix de modèles qui permettent de comparer des modèles qui ne sont pas forcément emboîtés les uns dans les autres.

Les critères AIC et BIC sont les plus utilisés. Ces critères sont basés sur la philosophie suivante : plus la vraisemblance est grande, plus grande est donc la log-vraisemblance et meilleur est le modèle (en terme d'ajustement). Cependant la vraisemblance augmente avec la complexité du modèle, et choisir le modèle qui maximise la vraisemblance revient à choisir le modèle saturé. Ce modèle est clairement sur-paramétré, il "sur-ajuste" les données (overfitting).

Pour choisir des modèles plus parcimonieux, une stratégie consiste à pénaliser la vraisemblance par une fonction du nombre de paramètres.

- Par définition l'AIC (Akaike Informative Criterion) pour un modèle \mathcal{M} de dimension p est défini par

$$AIC(\mathcal{M}) = -2\mathcal{L}_n(\hat{\mathcal{M}}) + 2p$$

```
# AIC
aic = -2*fullresult.llf + 2*fullresult.params.shape[0]
print('AIC : %.2f' %(aic))
AIC : 492.14
```

Par défaut, `Statsmodels` calcul l'AIC et le retourne grâce à l'outil `aic`.

```
# AIC
print('AIC : %.2f' % (fullresult.aic))
AIC : 492.14
```

- Le critère de choix de modèle le BIC (Bayesian Informative Criterion) pour un modèle \mathcal{M} de dimension p est défini par :

$$AIC(\mathcal{M}) = -2\mathcal{L}_n(\hat{\mathcal{M}}) + p \log(n)$$

```
# BIC
import numpy as np
bic = -2*fullresult.llf + fullresult.params.shape[0]*np.log(donnee.shape[0])
print('BIC : %.2f' %(bic))
BIC : 533.50
```

Par défaut, `Statsmodels` calcul le BIC et le retourne grâce à l'outil `bic`.

```
# BIC
print('BIC : %.2f' % (fullresult.bic))
BIC : 533.50
```

On choisira le modèle qui possède le plus petit AIC ou BIC. L'utilisation de ces critères est simple. Pour chaque modèle concurrent, le critère de modèles est calculé et le modèle qui présente le plus faible est sélectionné.

Remarque

- Remarquons que certains logiciels utilisent $-AIC$ et $-BIC$ il est donc prudent de bien vérifier dans quel sens doivent être optimisés ces critères (maximisation ou minimisation). Ceci peut être fait aisément en comparant un modèle très mauvais (composé uniquement de la constante par exemple) à un bon modèle et de vérifier dans quel sens varient les critères de choix.
- Les pénalités ne sont pas choisies au hasard... Le critère BIC est issue de la théorie bayésienne. En deux mots, les modèles candidats sont vus comme des variables aléatoires sur lesquels on met une loi de probabilité a priori. Dans ce contexte, choisir le modèle qui possède le plus petit BIC revient à choisir le modèle qui maximise la probabilité a posteriori.

14.1.4 Subdivision apprentissage - test

Ce critère mesure la performance d'un modèle en terme de prévision. Pour rappel, nous avons construits nos modèle précédents sur l'ensemble de nos données. Or dans ce contexte, les classificateurs plus complexes ayant tendance à "coller" aux données laissent à penser, à tort, qu'ils présentent de meilleures performances. En règle générale, plus une observation pèse sur son propre classement en généralisation, plus optimiste sera le taux d'erreur en resubstitution. Bref, le taux d'erreur en resubstitution est totalement inutilisable dès lors que l'on souhaite comparer les performances de modèles de complexité différente (ou reposant sur des représentations différentes ex. arbre de décision vs. régression logistique).

Parmi les solutions envisageables, la plus simple consiste à évaluer le classifieur sur des données à part qui n'ont pas participé au processus d'apprentissage. Nous procédons de la manière suivante lorsque l'on dispose d'un échantillon Ω de taille n :

1. Nous tirons au hasard n_1 individus parmi n , il s'agit de l'**échantillon d'apprentissage**, nous les utilisons pour construire le modèle de prédiction M_1 . On dédie généralement 70% des données à l'apprentissage.
2. Sur les $n_2 = n - n_1$ observations restantes, l'**échantillon test**, nous appliquons le modèle M_1 afin d'avoir les valeurs prédictes les confronter aux valeurs observées. Habituellement $\frac{n_2}{n} = 1 - \frac{n_1}{n} = 30\%$.

Principal atout de cette approche, les indicateurs ainsi obtenus sont non-biaisés. Ils permettent de comparer les mérites respectifs de plusieurs modèles, même s'ils sont de complexité différente, même s'ils ne reposent pas sur des systèmes de représentation identiques (ex. un classifieur linéaire vs. un classifieur non linéaire). C'est la démarche à privilégier si l'on dispose de suffisamment d'observations.

Et c'est bien là le principal défaut de cette démarche. Lorsque nous travaillons sur un petit échantillon, en réservant une partie pour l'évaluation pénalise la construction du modèle, sans pour autant que l'on ait une évaluation fiable des performances puisque l'effectif est trop faible. Nous sommes face à 2 exigences contradictoires :

- Réserver une grande partie des données à l'apprentissage favorise la construction d'un modèle de bonne qualité. En revanche, l'échantillon test sera trop réduit pour espérer obtenir une estimation viable des performances en prédiction.

- Réserver une fraction plus forte au test permet certes d'obtenir une évaluation fiable. Mais dans ce cas nous nous tirons une balle dans le pied (aïe!) car le modèle élaboré peut être dégradé faute d'informations (d'observations) suffisantes.

Bref, les proportions habituellement mises en avant (70% vs. 30%) ne doivent pas être prises au pied de la lettre. Tout est affaire de compromis : il en faut suffisamment pour l'apprentissage afin de produire un modèle consistant ; il en faut suffisamment pour le test afin d'obtenir une évaluation fiable des performances. Les « bonnes » proportions dépendent souvent des caractéristiques du classifieur et des données analysées (rapport entre le nombre d'observations et le nombre de variables, degré de difficulté du concept à apprendre, etc.).

Pour trouver le meilleur compromis **biais - variance**, il est recommandé de subdiviser notre dataset en deux parties :

- Une partie qui sera utilisée pour entraîner le modèle avec différents niveaux de complexité et différentes valeurs d'hyperparamètres qu'on appelle **échantillon d'apprentissage** ou (**train set** en anglais).
- Une partie pour mesurer l'erreur commise par la meilleure règle de prédiction et la comparer à l'erreur commise par des règles de familles différentes sur des données qui n'ont pas du tout participé à la construction des règles : C'est **l'échantillon test** ou (**test set** en anglais).

Sous scikit-learn, on utilise la fonction **train_test_split** du module **model_selection** qui permet de séparer le dataset en train set et en test set. Cette méthode mélange le dataset de façon aléatoire avant d'effectuer le tirage. La part des observations consacrée à test set dépend de la taille totale du dataset. Avec un test set trop grand, on peut nuire à la qualité de l'apprentissage. En général, on met 70% de notre dataset dans le train set et 30% sur le test set.

```
# Modélisation sur le train set
from sklearn.model_selection import train_test_split

XTrain, XTest, yTrain, yTest = train_test_split(X,y, test_size=.3,
                                                random_state=2021)

# Importation de la classe de calcul
import statsmodels.api as sm
# on ajoute une colonne pour la constante
x_stat = sm.add_constant(XTrain)
# on ajuste le modèle
model = sm.Logit(yTrain, x_stat)
# lancer les calculs
result = model.fit()
# résumé des résultats
print(result.summary())

Optimization terminated successfully.
    Current function value: 0.505964
    Iterations 6
    Logit Regression Results
=====
```

Dep. Variable:	chd	No. Observations:	323			
Model:	Logit	Df Residuals:	313			
Method:	MLE	Df Model:	9			
Date:	Wed, 20 Oct 2021	Pseudo R-squ.:	0.2331			
Time:	01:24:13	Log-Likelihood:	-163.43			
converged:	True	LL-Null:	-213.10			
Covariance Type:	nonrobust	LLR p-value:	2.128e-17			
<hr/>						
	coef	std err	z	P> z	[0.025	0.975]
const	-5.6349	1.562	-3.608	0.000	-8.696	-2.574
sbp	0.0055	0.007	0.773	0.440	-0.008	0.019
tobacco	0.0777	0.032	2.417	0.016	0.015	0.141
ldl	0.1789	0.072	2.488	0.013	0.038	0.320
adiposity	0.0157	0.036	0.436	0.663	-0.055	0.086
famhist	0.9652	0.274	3.521	0.000	0.428	1.502
typea	0.0277	0.016	1.783	0.075	-0.003	0.058
obesity	-0.0573	0.051	-1.122	0.262	-0.157	0.043
alcohol	-0.0040	0.005	-0.778	0.437	-0.014	0.006
age	0.0532	0.015	3.671	0.000	0.025	0.082

Remarque :

Lorsque les effectifs sont très faibles, nous avons intérêt à construire le modèle \mathcal{M} sur la totalité des données, puis à utiliser des techniques de ré-échantillonnage pour en mesurer les performances (ex. **la validation croisée, le bootstrap**). L'intérêt est double. Nous utilisons la totalité des données (la totalité de l'information disponible) pour construire le classifieur. Et nous pouvons obtenir une évaluation (plus ou moins) faiblement biaisée de son erreur de prédiction.

Le principe de la validation croisée de **moyenner** le pourcentage de mal classés à l'aide de plusieurs découpages de l'échantillon. Plus précisément, on divise l'échantillon initial en K sous-échantillons E_k de même taille et on effectue K procédures d'apprentissage-validation pour les quelles :

- L'échantillon test sera constitué d'une division E_k ;
- L'échantillon d'apprentissage sera constituée de l'ensemble des autres divisions $E - E_k$.

On obtient ainsi une prévision pour chaque individu de la division E_k et une fois les K procédures apprentissage-validation effectuées, on a une prévision pour tous les individus de l'échantillon. Il suffit alors de comparer ces prévisions aux valeurs observées pour obtenir une estimation de la probabilité d'erreur de la règle. Le modèle retenu sera le modèle qui conduit à l'estimation minimale.

Bien entendu le choix du nombre K de parties n'est pas anodin :

- Plus K est faible, plus la capacité de prévision sera évaluée dans de nombreux cas puisque le nombre d'observations dans la validation sera élevé, mais moins l'estimation sera précise puisque moins de données seront utilisées pour estimer les paramètres du modèle ;

- Au contraire, un K élevé conduit à peu d'observations dans la validation et donc à une plus grande variance dans l'estimation de la probabilité d'erreur.

Sous [Python](#), le package [scikit-learn](#) à travers le module `model_selection` propose plusieurs outils permettant d'effectuer une validation croisée. Cependant, avec [Statsmodels](#), il est impossible d'effectuer une validation croisée.

14.1.5 Sélection automatique

Les procédures que nous venons d'étudier permettent de sélectionner un modèle à partir d'une famille de modèles donnée. Une autre approche de la sélection de modèle consiste à chercher parmi les variables X_1, \dots, X_p , celles qui « expliquent le mieux » Y . Par exemple, pour la régression logistique, nous pourrions nous poser le problème de chercher le meilleur sous-ensemble des p variables explicatives pour un critère C donné (AIC, BIC...). Le nombre de sous ensembles de p variables étant 2^p , nous serions en présence de 2^p modèles logistiques possibles, c'est-à-dire 2^p modèles différents. Bien entendu, nous sélectionnerions le modèle qui optimiseraient le critère C . Cependant, dans de nombreuses situations, p est grand et par conséquent le nombre de modèles considérés est « très grand ». Les algorithmes d'optimisation du critère C deviennent très coûteux en temps de calcul. On préfère alors souvent utiliser des méthodes de recherche *pas à pas*.

1. forward selection

A chaque pas, une variable est ajoutée au modèle.

- Si la méthode ascendante utilise un test de déviance, nous rajoutons la variable X_j dont la valeur p (probabilité critique) associée à la statistique de test de déviance qui compare les 2 modèles est minimale. Nous nous arrêtons lorsque toutes les variables sont intégrées ou lorsque la valeur p est plus grande qu'une valeur seuil.
- Si la méthode ascendante utilise un critère de choix, nous ajoutons la variable X_j dont l'ajout au modèle conduit à l'optimisation la plus grande du critère de choix. Nous nous arrêtons lorsque toutes les variables sont intégrées ou lorsque qu'aucune variable ne permet l'optimisation du critère de choix.

2. Backward selection

A la première étape toutes les variables sont intégrées au modèle.

- Si la méthode descendante utilise un test de déviance, nous éliminons ensuite la variable X_j dont la valeur p associée à la statistique de test de déviance est la plus grande. Nous nous arrêtons lorsque toutes les variables sont retirées du modèle ou lorsque la valeur p est plus petite qu'une valeur seuil.
- Si la méthode descendante utilise un critère de choix, nous retirons la variable X_j dont le retrait du modèle conduit à l'augmentation la plus grande du critère de choix. Nous nous arrêtons lorsque toutes les variables sont retirées ou lorsque qu'aucune variable ne permet l'augmentation du critère de choix.

```
# Backward Selection based on p-value
import statsmodels.api as sm
from statsmodels.genmod.generalized_linear_model import GLM
```

```

from statsmodels.genmod import families

def backward_selected(X,y,
                     threshold_out = 0.05,
                     verbose=True):
    included=list(X.columns)
    while True:
        changed=False
        model = GLM(y,sm.add_constant(pd.DataFrame(X[included])),
                    family=families.Binomial()).fit(attach_wls=True, atol=1e-10)
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.idxmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature,
                                                               worst_pval))
        if not changed:
            break
    model = GLM(y,sm.add_constant(pd.DataFrame(X[included])),
                family=families.Binomial()).fit(attach_wls=True, atol=1e-10)
    return model.summary2()

backward = backward_selected(XTrain, yTrain)
print(backward)

Drop adiposity           with p-value 0.662555
Drop alcohol             with p-value 0.450036
Drop sbp                 with p-value 0.473781
Drop obesity              with p-value 0.281792
Drop typea               with p-value 0.106317
Results: Generalized linear model
=====
Model:          GLM      AIC:       341.9972
Link Function: logit    BIC:      -1505.2962
Dependent Variable: chd   Log-Likelihood: -166.00
Date: 2021-11-07 11:33 LL-Null:     -213.10
No. Observations: 323    Deviance:    332.00
Df Model:        4      Pearson chi2: 328.
Df Residuals:   318    Scale:      1.0000
Method:         IRLS

Coef.  Std.Err.      z    P>|z|    [0.025  0.975]
-----
const   -4.5425    0.6057  -7.4995  0.0000  -5.7296  -3.3553
tobacco 0.0712    0.0310   2.2973  0.0216   0.0105   0.1319
ldl     0.1768    0.0663   2.6667  0.0077   0.0469   0.3068

```

famhist	0.9528	0.2706	3.5215	0.0004	0.4225	1.4832
age	0.0540	0.0119	4.5313	0.0000	0.0307	0.0774

3. Stepwise selection

Idem que l'ascendante, sauf que l'on peut éliminer des variables déjà introduites. En effet, il peut arriver que des variables introduites au début de l'algorithme ne soient plus significatives après introduction de nouvelles variables. Remarquons qu'en général la variable « constante » est toujours présente dans le modèle.

14.2 Validation du modèle

14.2.1 Analyse des résidus

On se donne M_β un modèle logistique. Pour simplifier les notations, on écrira $p_i = p_\beta(x_i)$ et $\hat{p}_i = p_{\hat{\beta}}(x_i)$.

Les différents types de résidus

A l'image de la régression linéaire plusieurs types de résidus sont proposés par les logiciels. Le premier, le plus simple à calculer est tout simplement $Y_i - \hat{p}_i$. Ces résidus sont appelés *résidus bruts*. Ils permettent de mesurer l'ajustement du modèle sur chaque observation. Ces résidus n'ayant pas la même variance, ils sont difficiles à comparer. En effet, on rappelle que $V_\beta(Y|X = x_i) = p_i(1-p_i)$. Par conséquent, la variance de tels résidus risquent d'être élevées pour des valeurs de p_i proches de 1/2. Un moyen de pallier à cette difficulté est de considérer les *résidus de Pearson* :

$$RP_i = \frac{Y_i - \hat{p}_i}{\sqrt{\hat{p}_i(1 - \hat{p}_i)}} \quad (14.1)$$

Par définition, on standardise les résidus par la variance théorique de Y_i . Cependant, comme \hat{p}_i est aléatoire, il est évident que $V_\beta(Y_i - p_i) \neq V_\beta(Y_i - \hat{p}_i)$. En effet, en notant

$$\begin{cases} \varepsilon_i = Y_i - p_i \\ \hat{\varepsilon}_i = Y_i - \hat{p}_i \end{cases}$$

On a :

Hypothèses	Réalité
$\mathbb{E}(\varepsilon_i) = 0$	$\mathbb{E}(\hat{\varepsilon}_i) \simeq 0$
$V(\varepsilon) = p_i(1 - p_i)$	$V(\hat{\varepsilon}) \simeq p_i(1 - p_i)(1 - h_{ii})$

où h_{ii} est l'élément de la $i^{\text{ème}}$ ligne et de la $i^{\text{ème}}$ colonne de la matrice $H = X(X'W_{\hat{\beta}}X)^{-1}X'W_{\hat{\beta}}$

Il est par conséquent intéressant de considérer la version **standardisée** des résidus de Pearson :

$$\frac{Y_i - \hat{p}_i}{\sqrt{\hat{p}_i(1 - \hat{p}_i)(1 - h_{ii})}}$$

Ces résidus seront en effet plus facile à analyser (leur distribution étant « presque » centrée réduite).

Les **résidus de déviance** sont définis par :

$$\begin{aligned} rd_i &= \text{signe}(Y_i - \hat{p}_i) \sqrt{2\mathcal{L}_1(Y_i, \hat{\beta})} \\ &= \text{signe}(Y_i - \hat{p}_i) \sqrt{2(Y_i \log \hat{p}_i + (1 - Y_i) \log(1 - \hat{p}_i))} \end{aligned}$$

Là encore pour tenir compte de la variabilité ces résidus sont standardisés :

$$\frac{rd_i}{\sqrt{1 - h_{ii}}}$$

Ces deux types de résidus de déviance sont ceux qui sont en général conseillés.

Index plot

Pour la régression logistique, les résidus de déviance sont souvent préférés. De nombreuses études expérimentales ont montré qu'ils approchent mieux la loi normale que les résidus de Pearson. Pour cette raison, ces résidus prennent généralement des valeurs qui varient entre -2 et 2. Nous pourront construire un *index plot* pour détecter des valeurs aberrantes. Ce graphique ordonne les résidus en fonction du numéro de leur observation. Les points pour lesquels on observe un résidu élevé (hors de [-2,2] par exemple) devront faire l'objet d'une étude approfondie.

Graphique prédiction linéaire-résidus

Ce graphique qui représente $X'\hat{\beta}$ en abscisse et $\hat{\epsilon}$ en ordonné permet de détecter les valeurs aberrantes mais aussi les structurations suspectes. Si une structuration suspecte apparaît, il sera peut être adéquat d'ajouter une nouvelle variable afin de prendre en compte cette structuration. Dans le cas des données individuelles, ce type de graphique donne toujours des structurations et n'est donc pas conseiller.

Résidus partiels

Les résidus partiels sont définis par :

$$\hat{\epsilon}_j^P = \frac{Y_i - \hat{p}_i}{\hat{p}_i(1 - \hat{p}_i)} + \hat{\beta}_j X_j$$

L'analyse consiste à tracer pour toutes les variables j les points avec en abscisse la variable j et en ordonnée les résidus partiels. Si le tracé est linéaire alors tout est normal. Si par contre une tendance non linéaire se dégage, il faut remplacer la variable j par une fonction de celle ci donnant la même tendance que celle observée.

Mallows (1986) propose d'utiliser les résidus partiels augmentés qui dans certaines situations permettent de mieux dégager cette tendance. Les résidus partiels augmentés pour la $j^{\text{ème}}$ variable nécessitent un nouveau modèle logistique identique mis à part le fait qu'une variable explicative supplémentaire est ajoutée : $X_{p+1} = X_j^2$ la $j^{\text{ème}}$ variable élevée au carré. Le nouveau vecteur de coefficient β du modèle est estimé et les résidus partiels sont alors définis comme :

$$\hat{\epsilon}_{.j}^{PA} = \frac{Y_i - \hat{p}_i}{\hat{p}_i(1 - \hat{p}_i)} + \hat{\beta}_j X_j + \hat{\beta}_{p+1} X_j^2$$

L'analyse des diagrammes est identique à ceux des résidus partiels.

14.2.2 Points leviers et points influents

Ces notions sont analogues à celles du modèle linéaire.

Points leviers

Par définition les points leviers sont les points du design qui déterminent très fortement leur propre estimation. Nous avons vu que l'algorithme d'estimation des paramètres effectue à chaque étape une régression linéaire et s'arrête lorsque le processus devient stationnaire :

$$\hat{\beta} = (X'W_{\hat{\beta}}X)^{-1}X'W_{\hat{\beta}}Z$$

et la prédiction linéaire est :

$$X\hat{\beta} = X(X'W_{\hat{\beta}}X)^{-1}X'W_{\hat{\beta}}Z = HZ$$

où H est une matrice de projection selon la métrique $W_{\hat{\beta}}$. Comme nous transformons $X\hat{\beta}$ par une fonction monotone, des $X\hat{\beta}$ extrêmes entraînent des valeurs de \hat{p} extrêmes. Nous allons donc utiliser la même méthode de diagnostic que celle de la régression simple avec une nouvelle matrice de projection H . Pour la $i^{\text{ème}}$ prédiction linéaire nous avons :

$$[X\hat{\beta}]_i = H_{ii}Z_i + \sum_{j \neq i} H_{ij}Z_j$$

Si H_{ii} est grand relativement aux H_{ij} , $j \neq i$ alors la $i^{\text{ème}}$ observation contribue fortement à la construction de $[X\hat{\beta}]_i$. On dira que le *poids* de l'observation i sur sa propre estimation vaut H_{ii} .

Comme H est un projecteur nous savons que $0 \leq H_{ii} \leq 1$. Nous avons les cas extrêmes suivants :

- Si $H_{ii} = 1$, \hat{p}_i est entièrement déterminé par Y_i car $H_{ij} = 0$ pour tout j .
- Si $H_{ii} = 0$, Y_i n'a pas d'influence sur \hat{p}_i .

La trace d'un projecteur étant égale à la dimension du sous espace dans lequel on projette, on a $\text{tr}(H) = \sum_i H_{ii} = p$. Donc en moyenne H_{ii} vaut p/n . Pour dire que la valeur de H_{ii} contribue trop fortement à la construction de \hat{p}_i , il faut un seuil au delà duquel le point est un point levier. Par habitude, si $H_{ii} > 2p/n$ ou si $H_{ii} > 3p/n$ alors le $i^{\text{ème}}$ point est déclaré comme un point levier.

En pratique un tracé de H_{ii} est effectué et l'on cherche les points dont le H_{ii} est supérieur à $3p/n$ ou $2p/n$. Ces points sont **leviers** et leur valeur influe fortement sur leur propre prévision.

Points influents

Les points influents sont des points qui influent sur le modèle de telle sorte que si on les enlève, alors l'estimation des coefficients sera fortement changée. La mesure la plus classique d'influence est la **distance de Cook**. Il s'agit d'une distance entre le coefficient estimé avec toutes les observations et celui estimé avec toutes les observations sauf une. La distance de Cook pour l'individu i est définie par :

$$D_i = \frac{1}{p}(\hat{\beta}_{(i)} - \hat{\beta})'X'W_{\hat{\beta}}X(\hat{\beta}_{(i)} - \hat{\beta}) \approx \frac{r_{P_i}^2 H_{ii}}{p(1 - H_{ii})^2}$$

où r_{P_i} est le résidu de Pearson pour le $i^{\text{ème}}$ individu.

Les distances de Cook sont généralement représentées comme sur la figure ???. Si une distance se révèle grande par rapport aux autres, alors ce point sera considéré comme influent. Il convient alors de comprendre pourquoi il est influent, soit :

- il est levier ;
- il est aberrant ;
- (les deux !)

Dans tous les cas il convient de comprendre si une erreur de mesure, une différence dans la population des individus est à l'origine de ce phénomène. Eventuellement pour obtenir des conclusions robustes il sera bon de refaire l'analyse sans ce(s) point(s).

14.2.3 Application

Dans le cadre de l'analyse des résidus, nous avons utilisé une autre instance du package `statsmodels` pour la régression logistique à consulter [ici](#).

```
# load dataset
import pandas as pd
donnee = pd.read_csv('deseases.txt', sep=',', header=0, index_col=0)
# codage 0/1
donnee["famhist"] = donnee["famhist"].astype('category').cat.codes
x = donnee.drop(['chd'], axis=1) # suppression
y = donnee['chd']
# Modélisation sur le train set
from sklearn.model_selection import train_test_split

XTrain, XTest, yTrain, yTest = train_test_split(x, y, test_size=.3,
                                                random_state=2021)

# Importation des classes de calcul
import statsmodels.api as sm
from statsmodels.genmod.generalized_linear_model import GLM
from statsmodels.genmod import families
# entraînement
trainmodel = GLM(yTrain, sm.add_constant(XTrain), family=families.Binomial())
trainresult = trainmodel.fit(attach_wls=True, atol=1e-10)
print(trainresult.summary2())

Results: Generalized linear model
=====
Model:          GLM      AIC:        346.8525
Link Function: logit    BIC:       -1481.5527
Dependent Variable: chd   Log-Likelihood: -163.43
Date: 2021-10-26 16:34 LL-Null:     -213.10
No. Observations: 323    Deviance:    326.85
Df Model:         9     Pearson chi2: 320.
Df Residuals:    313    Scale:       1.0000
```

Method: IRLS

	Coef.	Std.Err.	z	P> z	[0.025	0.975]
const	-5.6349	1.5618	-3.6080	0.0003	-8.6959	-2.5739
sbp	0.0055	0.0071	0.7727	0.4397	-0.0084	0.0193
tobacco	0.0777	0.0321	2.4172	0.0156	0.0147	0.1406
ldl	0.1789	0.0719	2.4880	0.0128	0.0380	0.3197
adiposity	0.0157	0.0360	0.4364	0.6626	-0.0549	0.0863
famhist	0.9652	0.2741	3.5209	0.0004	0.4279	1.5025
typea	0.0277	0.0155	1.7826	0.0746	-0.0028	0.0581
obesity	-0.0573	0.0510	-1.1222	0.2618	-0.1573	0.0428
alcohol	-0.0040	0.0051	-0.7778	0.4367	-0.0139	0.0060
age	0.0532	0.0145	3.6705	0.0002	0.0248	0.0816

Les résultats sont identiques.

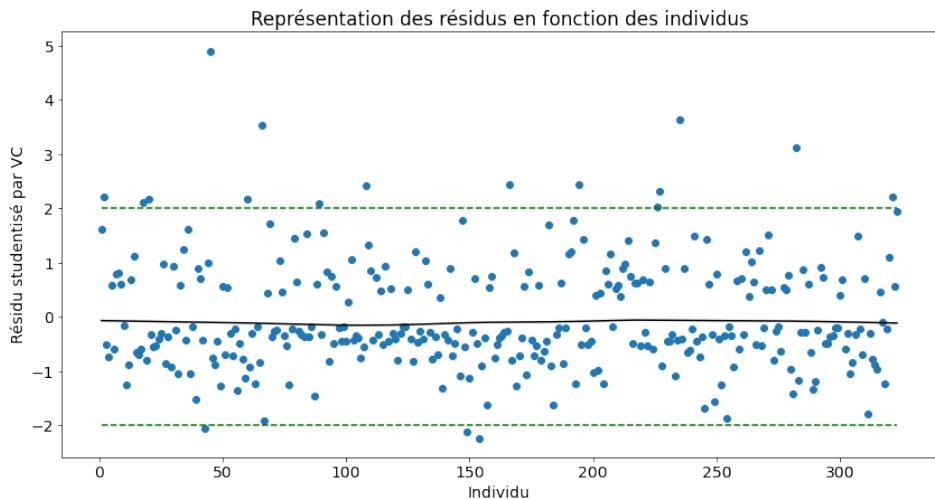
Valeurs aberrantes

Sur le graphique suivant sont représentés en ordonnée les résidus studentisés pour chaque individu. Les lignes en pointillés vertes représentent le seuil d'une loi de Student à un niveau de confiance de 95%.

```
import numpy as np
from matplotlib import pyplot as plt
plt.rc("figure", figsize=(16, 8))
plt.rc("font", size=14)

n = len(yTrain)

from statsmodels.nonparametric.smoothers_lowess import lowess
infl = trainresult.get_influence(observed=False)
x = np.linspace(1,n,n)
y = np.repeat(2,n)
fig, ax = plt.subplots(figsize = (14,7))
ax.scatter(x,infl.resid_studentized)
ax.plot(x,y,linestyle='dashed',color='green')
ax.plot(x,-y,linestyle='dashed',color='green')
filtered = lowess(infl.resid_studentized,x)
ax.plot(filtered[:,0],filtered[:,1],color='black')
ax.set_xlabel("Individu")
ax.set_ylabel("Résidu studentisé par VC")
ax.set_title("Représentation des résidus en fonction des individus")
plt.show()
```



Il est normal que certaines valeurs dépassent ce seuil ponctuellement. En fait, en moyenne 5% des individus sont à l'extérieur de la bande matérialisée par les lignes en pointillés vertes.

La courbe en noire est un lissage des valeurs des résidus studentisés. On observe une légère courbure au tour du 100^{ème} individu, mais rien de dramatique.

Les observations suspectes correspondent aux patients 21, 133, 261 et 448 et il faut regarder plus précisément ces observations pour comprendre pourquoi ces points sont aberrants. Une valeur aberrante n'est pas nécessairement une observation que l'on doit écarter de l'échantillon. C'est surtout le cas s'il s'agit d'un point levier.

```
#individus aberrants
print(XTrain.iloc[np.where(np.abs(infl.resid_studentized) > 3)])

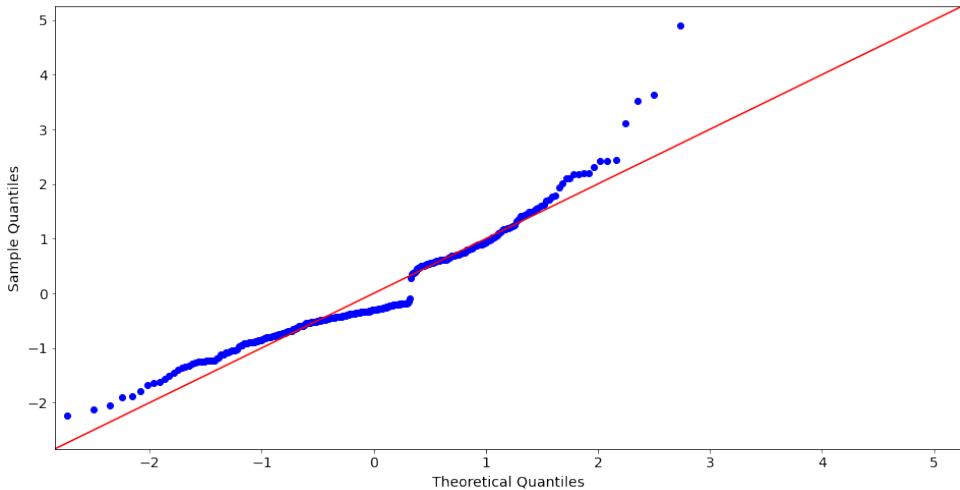
      sbp  tobacco    ldl  adiposity  famhist  typea  obesity  alcohol \
row.names
261     118      0.00   2.39      12.13       0     49    18.46     0.26
133     134      1.50   3.73      21.53       0     41    24.70    11.11
21      106      1.61   1.74      12.32       0     74    20.92    13.37
448     142      0.00   4.32      25.22       0     47    28.92     6.53

      age
row.names
261     17
133     30
21      20
448     34
```

Analyse de la normalité

Ci-dessous est représenté le graphe quantile-quantile des quantiles empiriques en ordonné par rapport aux quantiles théorique d'une loi $\mathcal{N}(0,1)$. Le choix de la version studentisé des résidus est nécessaire pour que chaque valeur soit normalisée et que la comparaison avec les quantiles empiriques fassent sens.

```
# qq-plot
import statsmodels.api as sm
fig = sm.qqplot(infl.resid_studentized,line='45')
```

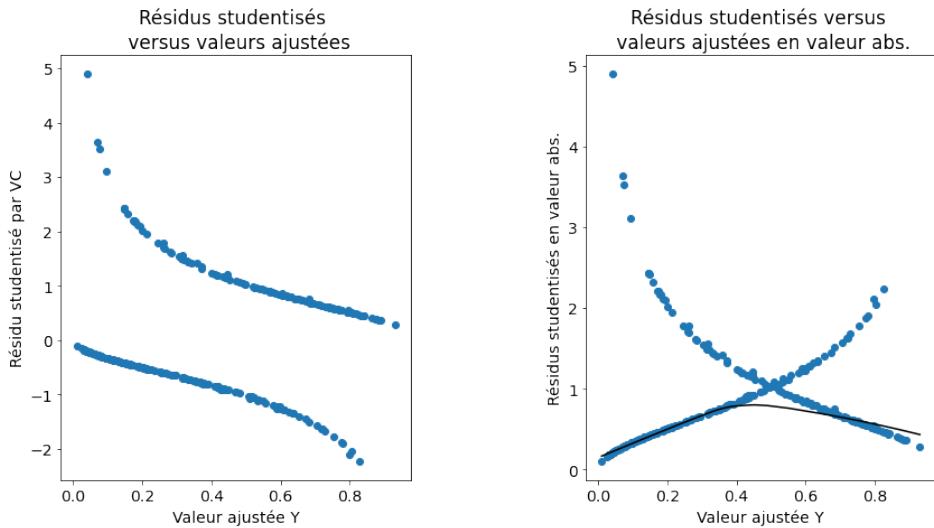


Homoscédasticité

```
fig, (ax1,ax2) = plt.subplots(1,2, figsize = (14,7))
fig.subplots_adjust(wspace=0.5)
filtered = lowess(abs(infl.resid_studentized),trainresult.fittedvalues)
ax1.scatter(trainresult.fittedvalues,infl.resid_studentized)
ax2.scatter(trainresult.fittedvalues,np.abs(infl.resid_studentized))
ax2.plot(filtered[:,0],filtered[:,1],color = 'black')

ax1.set_xlabel("Valeur ajustée Y")
ax1.set_ylabel("Résidu studentisé par VC")
ax1.set_title("Résidus studentisés \n versus valeurs ajustées")

ax2.set_xlabel("Valeur ajustée Y")
ax2.set_ylabel("Résidus studentisés en valeur abs.")
ax2.set_title("Résidus studentisés versus \n valeurs ajustées en valeur abs.")
plt.show()
```



Analyse de la structure des résidus

Ici, on affine l'analyse des résidus en représentant le graphique des résidus studentisés en fonction de chaque variable explicative (quantitative)

```
# analyse de la structure des résidus (residus versus variables explicatives)
fig, ((ax1,ax2),(ax3,ax4),(ax5,ax6),(ax7,ax8)) = plt.subplots(4,2,figsize=(14,13))
fig.subplots_adjust(wspace=1,hspace=1)

# residus vs sbp
filtered_sbp = lowess(infl.resid_studentized,XTrain['sbp'])
ax1.scatter(XTrain['sbp'],infl.resid_studentized)
ax1.plot(filtered_sbp[:,0],filtered_sbp[:,1],color = 'black')
ax1.set_xlabel("sbp")
ax1.set_ylabel("Résidus stud.")
ax1.set_title("Résidus versus sbp")

# residus vs tobacco
filtered_tobacco = lowess(infl.resid_studentized,XTrain['tobacco'])
ax2.scatter(XTrain['tobacco'],infl.resid_studentized)
ax2.plot(filtered_tobacco[:,0],filtered_tobacco[:,1],color = 'black')
ax2.set_xlabel("tobacco")
ax2.set_ylabel("Résidus stud.")
ax2.set_title("Résidus versus tobacco")

# residus versus ldl
filtered_ldl = lowess(infl.resid_studentized,XTrain['ldl'])
ax3.scatter(XTrain['ldl'],infl.resid_studentized)
ax3.plot(filtered_ldl[:,0],filtered_ldl[:,1],color = 'black')
ax3.set_xlabel("ldl")
ax3.set_ylabel("Résidus stud.")
```

```

ax3.set_title("Résidus versus ldl")

# residus versus adiposity
filtered_adiposity = lowess(infl.resid_studentized,XTrain['adiposity'])
ax4.scatter(XTrain['adiposity'],infl.resid_studentized)
ax4.plot(filtered_adiposity[:,0],filtered_adiposity[:,1],color = 'black')
ax4.set_xlabel("adiposity")
ax4.set_ylabel("Résidus stud.")
ax4.set_title("Résidus versus adiposity")

# residus versus typea
filtered_typea = lowess(infl.resid_studentized,XTrain['typea'])
ax5.scatter(XTrain['typea'],infl.resid_studentized)
ax5.plot(filtered_typea[:,0],filtered_typea[:,1],color = 'black')
ax5.set_xlabel("typea")
ax5.set_ylabel("Résidus stud.")
ax5.set_title("Résidus versus typea")

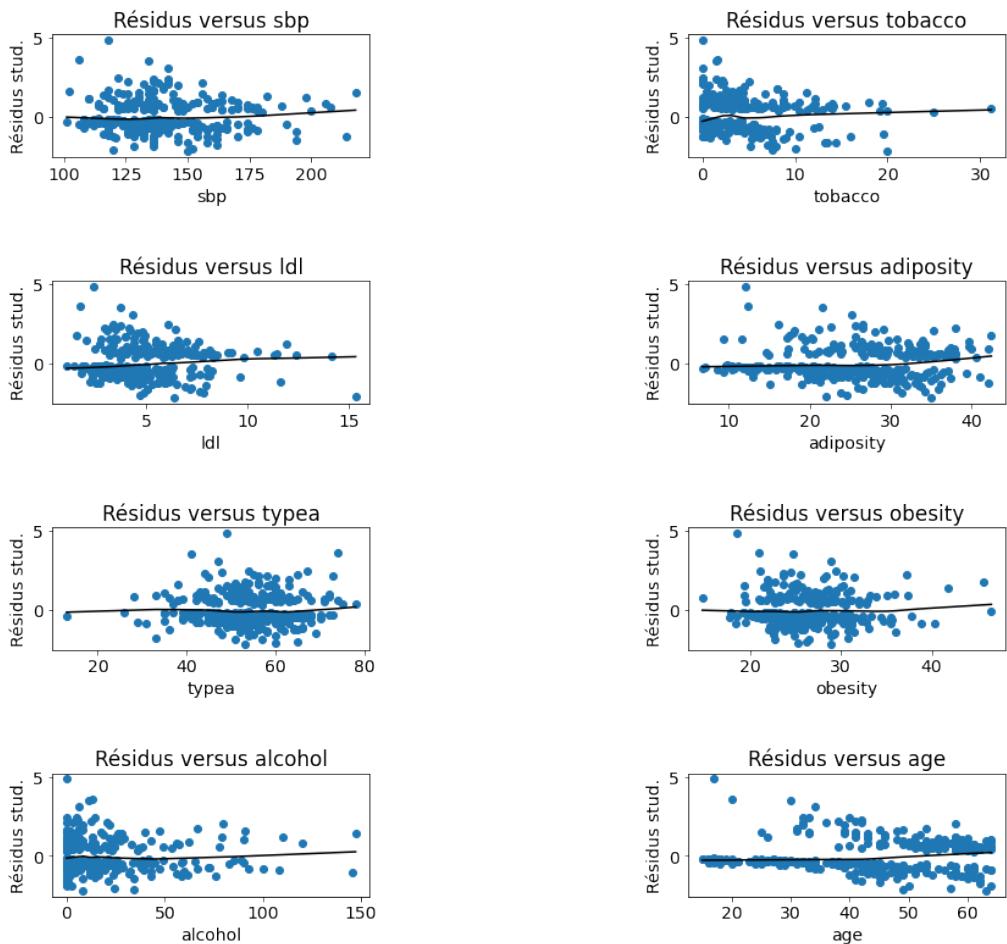
# residus versus obesity
filtered_obesity = lowess(infl.resid_studentized,XTrain['obesity'])
ax6.scatter(XTrain['obesity'],infl.resid_studentized)
ax6.plot(filtered_obesity[:,0],filtered_obesity[:,1],color = 'black')
ax6.set_xlabel("obesity")
ax6.set_ylabel("Résidus stud.")
ax6.set_title("Résidus versus obesity")

# residus versus alcohol
filtered_alcohol = lowess(infl.resid_studentized,XTrain['alcohol'])
ax7.scatter(XTrain['alcohol'],infl.resid_studentized)
ax7.plot(filtered_alcohol[:,0],filtered_alcohol[:,1],color = 'black')
ax7.set_xlabel("alcohol")
ax7.set_ylabel("Résidus stud.")
ax7.set_title("Résidus versus alcohol")

# residus versus age
filtered_age = lowess(infl.resid_studentized,XTrain['age'])
ax8.scatter(XTrain['age'],infl.resid_studentized)
ax8.plot(filtered_age[:,0],filtered_age[:,1],color = 'black')
ax8.set_xlabel("age")
ax8.set_ylabel("Résidus stud.")
ax8.set_title("Résidus versus age")

# Affichage
plt.show()

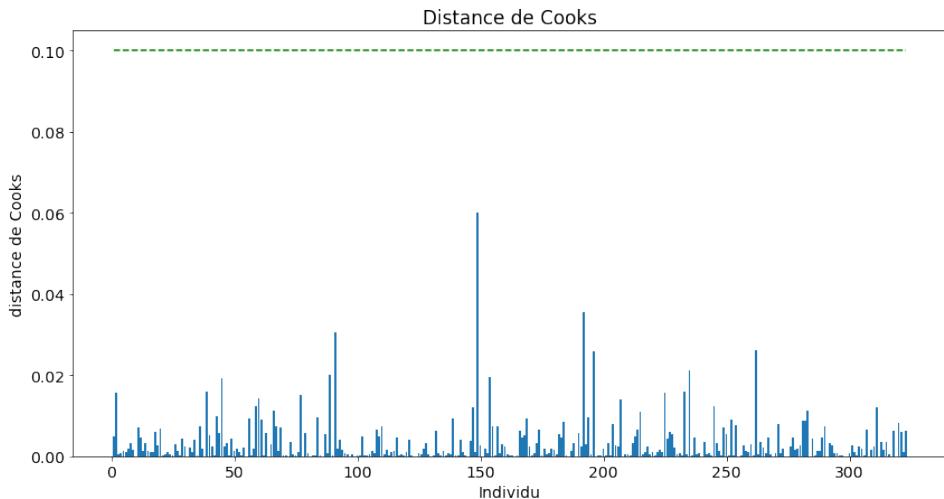
```



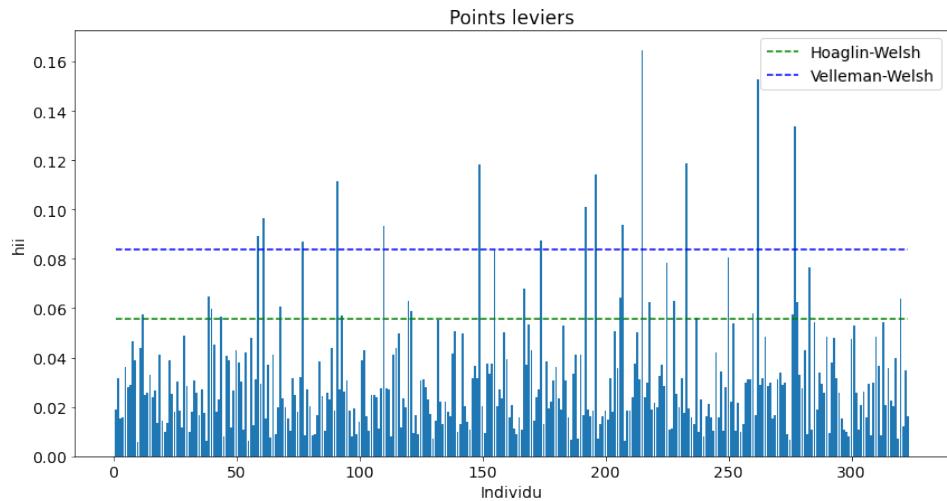
Distance de Cooks et points leviers

```
# Distance de Cooks
yok = np.repeat(0.1,n)
cd, pval = infl.cooks_distance

fig, ax = plt.subplots(figsize = (14,7))
ax.bar(x,cd)
ax.plot(x,yok,linestyle='dashed',color='green')
ax.set_xlabel("Individu")
ax.set_ylabel("distance de Cooks")
ax.set_title("Distance de Cooks")
plt.show()
```



```
# Points leviers
hii = infl.hat_matrix_diag
p = len(trainresult.params)-1
yhw = np.repeat(2*p/n,n) # Hoaglin-Welsh
yvw = np.repeat(3*p/n,n) # Velleman-Welsh
yh = np.repeat(0.5,n) # Huber
fig, ax = plt.subplots(figsize = (14,7))
ax.bar(x,hii)
ax.plot(x,yhw,linestyle='dashed',color='green',label='Hoaglin-Welsh')
ax.plot(x,yvw,linestyle='dashed',color='blue',label='Velleman-Welsh')
#ax.plot(x,yh,linestyle='dashed',color='black',label='Huber')
ax.set_xlabel("Individu")
ax.set_ylabel("hii")
ax.set_title("Points leviers")
ax.legend()
plt.show()
```



On affiche les résultats d'influence.

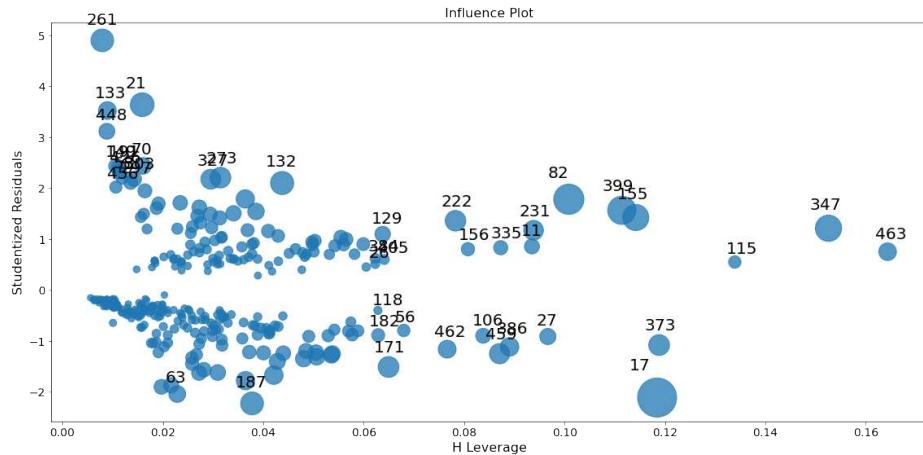
```
# summary
summ_df = infl.summary_frame()
print(summ_df.sort_values("cooks_d", ascending=False)[:10])

      dfb_const   dfb_sbp   dfb_tobacco   dfb_ldl   dfb_adiposity \
row.names
17     0.072290  0.129685 -0.039611 -0.717787      0.289022
82    -0.214095 -0.096765  0.103749 -0.095908     -0.186385
399   -0.028041  0.365545  0.105363 -0.127321      0.110318
347   -0.084766  0.257222 -0.060312  0.313865     -0.033029
155    0.043392 -0.105584 -0.079381  0.044940     -0.043468
21     0.068669 -0.101682  0.052298 -0.131683     -0.034371
132    0.073822 -0.159078 -0.163700 -0.133881      0.042875
187    0.071710 -0.017453 -0.363036 -0.031311     -0.020718
261    0.310887 -0.031793 -0.022564 -0.081358      0.046030
171    0.213503 -0.287989 -0.012839 -0.037395     -0.105883

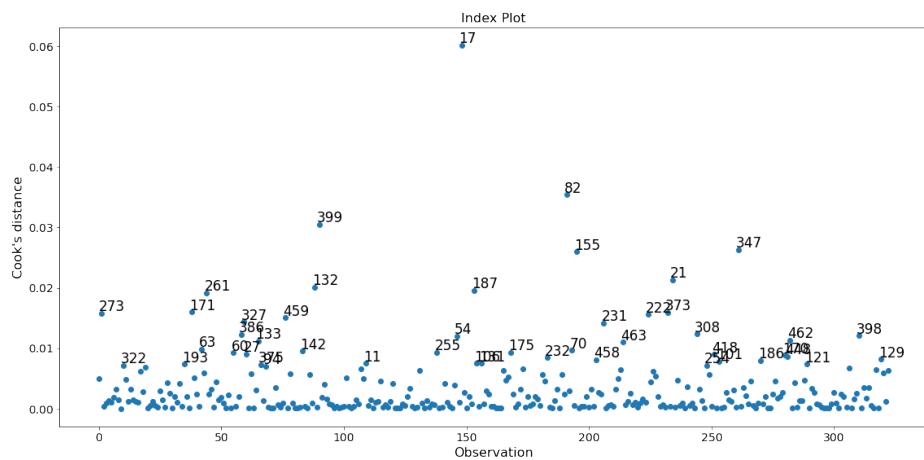
      dfb_famhist   dfb_typea   dfb_obesity   dfb_alcohol   dfb_age \
row.names
17      0.186708 -0.059878  -0.096688  -0.118838 -0.161393
82     -0.123148 -0.044434  0.490913  -0.023429  0.058671
399   -0.090018 -0.136534 -0.116016  0.199490 -0.176263
347    0.059193 -0.091215 -0.008597  0.183365 -0.182809
155    0.062390  0.038849 -0.017766  0.469643  0.035320
21     -0.108552  0.240162 -0.016776 -0.013831 -0.114338
132   -0.144122 -0.132438  0.090624  0.292713  0.100920
187    0.099992 -0.059953 -0.003059  0.105782  0.002777
261   -0.109672 -0.120338 -0.126019 -0.071480 -0.229861
171   -0.108934 -0.182098  0.091622  0.117034  0.133662
```

row.names	cooks_d	standard_resid	hat_diag	dffits_internal
17	0.060166	-2.116195	0.118439	-0.775669
82	0.035443	1.777644	0.100850	0.595342
399	0.030528	1.560428	0.111406	0.552517
347	0.026282	1.208221	0.152570	0.512659
155	0.025949	1.418739	0.114197	0.509402
21	0.021308	3.633976	0.015879	0.461604
132	0.020101	2.095969	0.043754	0.448343
187	0.019526	-2.230777	0.037756	-0.441882
261	0.019167	4.897379	0.007928	0.437799
171	0.015966	-1.516106	0.064949	-0.399574

```
# Plot influence
fig = infl.plot_influence()
fig.tight_layout(pad=1.0)
```

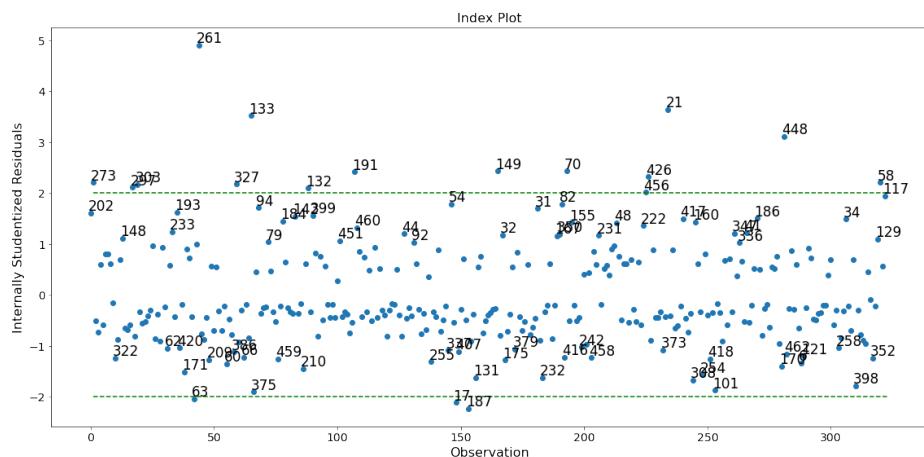


```
# Cooks distance
fig = infl.plot_index(y_var="cooks", threshold=2 * infl.cooks_distance[0].mean())
fig.tight_layout(pad=1.0)
```



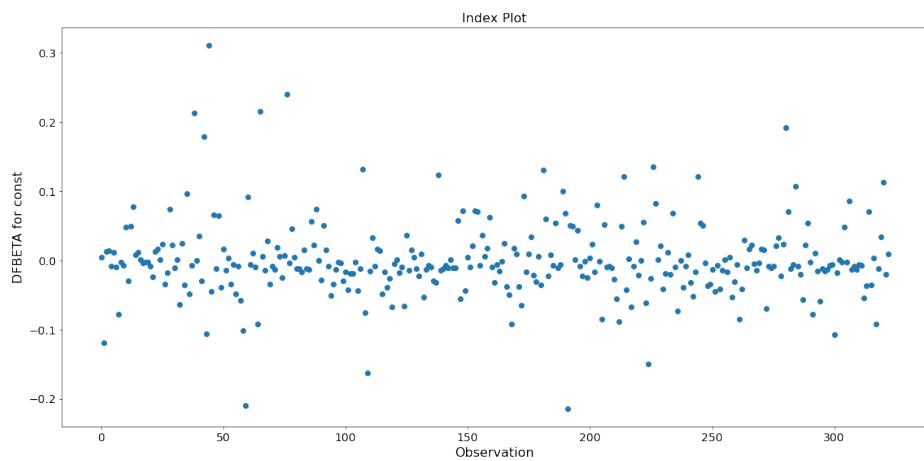
Internally studentized residuals

```
fig = infl.plot_index(y_var="resid", threshold=1)
fig.tight_layout(pad=1.0)
plt.plot(x,y,linestyle='dashed',color='green')
plt.plot(x,-y,linestyle='dashed',color='green')
plt.show()
```

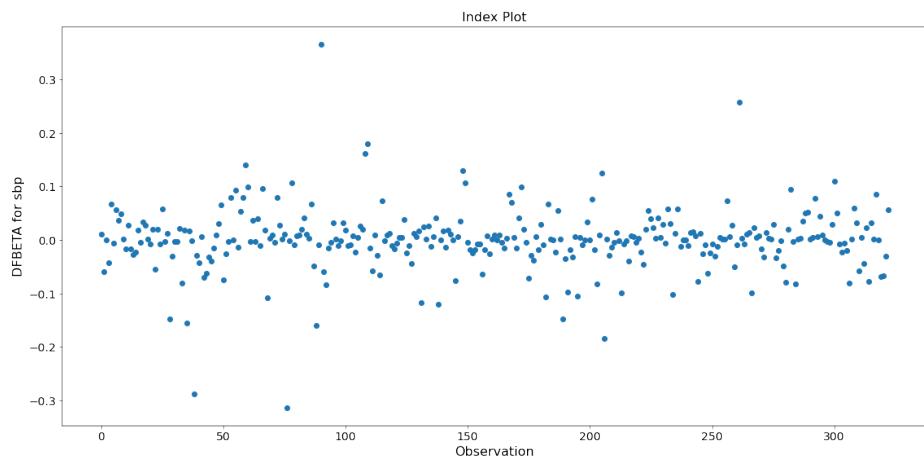


DFBETA for constant

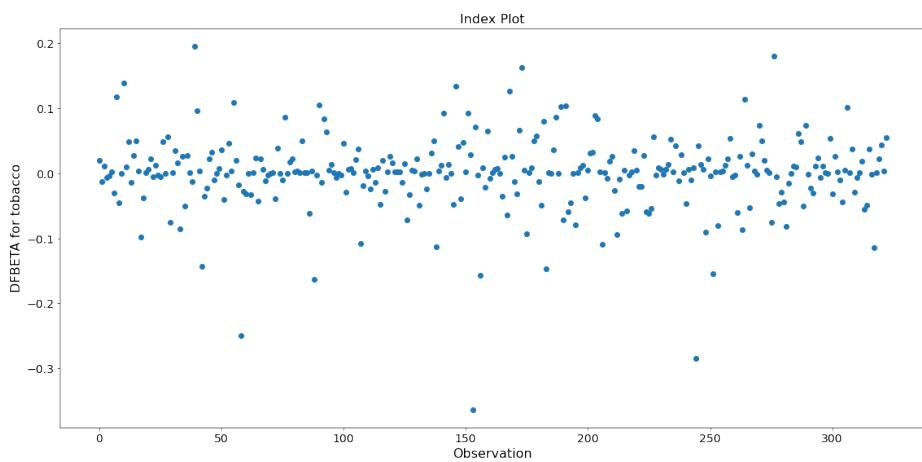
```
fig = infl.plot_index(y_var="dfbeta", idx=0, threshold=0.5)
fig.tight_layout(pad=1.0)
```



```
# DFBETA for sbp
fig = infl.plot_index(y_var="dfbeta", idx=1, threshold=0.5)
fig.tight_layout(pad=1.0)
```



```
# DFBETA for tobacco
fig = infl.plot_index(y_var="dfbeta", idx=2, threshold=0.5)
fig.tight_layout(pad=1.0)
```



CHAPITRE 15

ÉVALUATION DE LA RÉGRESSION LOGISTIQUE BINAIRE

Sommaire

15.1 Matrice de confusion	344
15.2 Quelques tests statistiques	354
15.3 Quelques courbes d'évaluation	360

Dans ce chapitre, nous nous consacrons à ce que l'on appelleraient des méthodes d'évaluation externes basées sur les prédictions $\hat{Y}(\omega)$ et/ou les probabilités a posteriori $\hat{p}(\omega)$ fournies par le classifieur. Nos indicateurs calculés dans le cadre de ce chapitre sont issus des résultats du modèle entraîné sur les données du train set.

15.1 Matrice de confusion

15.1.1 Construction de la matrice de confusion

Pour évaluer la capacité à bien classer du modèle, il est judicieux de construire ce que l'on appelle une **matrice de confusion**. Elle confronte toujours les *valeurs observées* de la variable dépendante avec les *variables predictes*, puis comptabilise les bonnes et les mauvaises prédictions. Son intérêt est qu'elle permet à la fois d'appréhender la quantité d'erreur (le taux d'erreur) et de rendre compte de la structure de l'erreur (la manière de se tromper du modèle). Le tableau 15.1 présente un exemple de matrice de confusion :

\hat{Y}	$\hat{-}$	$\hat{+}$	Total
Y	a	b	a+b
-	c	d	c+d
Total	a + c	b + d	n=a+b+c+d

TABLE 15.1 – Matrice de confusion - Forme générique

Sous Python, l'attribut `fittedvalues` correspond aux valeurs du logit calculées sur les données d'apprentissage.

```
# valeurs estimées par la régression en resubstitution
print(trainresult.fittedvalues)

row.names
202    -0.925220
273    -1.547276
445    -1.330772
275    -0.619453
404    1.087211
...
110    -3.089419
129    -0.112835
58     -1.570233
343    1.180721
117    -1.312119
Length: 323, dtype: float64
```

Nous pouvons reproduire ces valeurs à partir des coefficients de la régression et de la matrice des descripteurs.

```
# coefficient de la régression
print(trainresult.params)

const      -5.634891
sbp        0.005465
tobacco    0.077652
ldl        0.178851
adiposity   0.015717
famhist    0.965176
typea      0.027668
obesity    -0.057263
alcohol    -0.003959
age         0.053227
dtype: float64
```

On appelle la formule :

$$\text{logit}p_{\hat{\beta}}(X) = X \cdot \hat{\beta}$$

```
# valeurs estimées du logit
trainfittedvalues = pd.DataFrame(np.dot(sm.add_constant(XTrain), trainresult.params),
                                    columns=['fittedvalues'], index=XTrain.index)
print(trainfittedvalues)

fittedvalues
row.names
202      -0.925220
273      -1.547276
```

```

445      -1.330772
275      -0.619453
404      1.087211
...
110      ...
129      -0.112835
58       -1.570233
343      1.180721
117      -1.312119

```

[323 rows x 1 columns]

Nous allons à présent calculer la valeur du logit sur l'échantillon test en appliquant la même formule que celle sur l'échantillon d'apprentissage :

```

# valeurs prédictes sur le test set
testfittedvalues = np.dot(sm.add_constant(XTest), trainresult.params)
# Affichage des 10 premières
print(testfittedvalues[:10])

[ 6.19403834e-01 -7.81845961e-01 -3.50038418e-01 -3.06574912e+00
 -1.47713014e-04  7.61248898e-01 -3.22471209e+00 -2.67254981e+00
  2.06931421e+00 -6.07888465e-01]

```

Nous pouvons déduire du logit la prédiction en utilisant la règle d'affectation simple.

$$\hat{Y} = \begin{cases} 1 & \text{si } \text{logit} p_{\beta}(X) > 0 \\ 0 & \text{si } \text{logit} p_{\beta}(X) < 0 \end{cases}$$

```

# valeurs chd prédictes en resubstitution
chdpred = np.where(testfittedvalues > 0, 1, 0)
print(chdpred)

[1 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0
 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 1 0
 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0
 0 0 1 0 1 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 1 0 1 1 0]

```

Via un tableau croisé entre les classes observées et prédictes, nous obtenons la matrice de confusion suivante :

```

# Matrice de confusion
print(pd.crosstab(yTest, chdpred))

col_0    0    1
chd
0        77   22
1        21   19

```

la fonction `pred_table()` de l'objet de régression renvoie la matrice de confusion construite sur l'échantillon d'apprentissage

```

# matrice de confusion fourni par l'outil - train set
print(result.pred_table())

[[167.  36.]
 [ 47.  73.]]

```

15.1.2 Les indicateurs associés à la matrice de confusion

Dans un problème de classification binaire de type (+ vs -), à partir de la forme générique de la matrice de confusion (tableau 15.1), plusieurs indicateurs peuvent être déduits pour rendre compte de la concordance entre les valeurs observées et les valeurs prédictes.

Nous sauvegardons dans l'objet `matconf` notre matrice de confusion construite sur l'échantillon test. Elle nous servira pour le calcul des indicateurs ci-dessous

```
# Sauvegarde de la matrice de confusion
matconf = pd.crosstab(yTest, chdpred)
print(matconf)

col_0    0    1
chd
0        77   22
1        21   19
```

Les vrais positifs

Ce sont les observations qui ont été classées positives et qui le sont réellement. Sa valeur dans le tableau 15.1 est d .

```
# Vrais positifs
vrai_positif = matconf.values[1,1]
print('Vrais positifs : %.i' % (vrai_positif))

Vrais positifs : 19
```

Les faux positifs

Ce sont les individus classés positifs et qui sont en réalité des négatifs. Sa valeur dans le tableau 15.1 est b .

```
# Faux positifs
faux_positif = matconf.values[0,1]
print('Faux positifs : %.i' % (faux_positif))

Faux positifs : 22
```

Les faux négatifs

Ce sont les individus classés négatifs et qui sont en réalité des positifs. Sa valeur dans le tableau 15.1 est c .

```
# Faux négatifs
faux_negatif = matconf.values[1,0]
print('Faux négatifs : %.i' % (faux_negatif))

Faux négatifs : 21
```

Les vrais négatifs

Ce sont les individus qui ont été classés négatifs et qui le sont réellement. Sa valeur dans le tableau 15.1 est a .

```
# Vrai négatif
vrai_negatif = matconf.values[0,0]
print('Vrais négatifs : %.i' % (vrai_negatif))

Vrais négatifs : 77
```

Remarque :

Ces termes sont peu utilisés en pratique car les positifs et les négatifs n'ont pas le même statut dans la majorité des études.

Le taux d'erreur

C'est égal au nombre de mauvais classement rapporté à l'effectif total c'est-à-dire

$$\begin{aligned}\varepsilon &= \frac{b+c}{n} = 1 - \frac{a+b}{n} \\ &= \frac{\text{faux positifs} + \text{faux négatifs}}{n}\end{aligned}$$

Il estime la probabilité de mauvais classement du modèle.

```
# Taux d'erreur
error_rate = (faux_positif + faux_negatif)/len(yTest)
print('Error rate : %.2f' % (error_rate))

Error rate : 0.31
```

Le taux de succès

Il correspond à la probabilité de bon classement. C'est le complémentaire à 1 du taux d'erreur :

$$\theta = \frac{a+d}{n} = 1 - \varepsilon$$
$$= \frac{\text{vrais positifs} + \text{vrais négatifs}}{n}$$

```
# Taux de succès
accuracy = (vrai_positif + vrai_negatif)/len(yTest)
print('Accuracy : %.2f' %(accuracy))

Accuracy : 0.69
```

La sensibilité

Encore appelée *rappel* ou *taux de vrais positifs* (*TVP*), la sensibilité indique la capacité du modèle à retrouver les positifs.

$$\text{Sensibilité} = \frac{d}{c+d} = \frac{\text{vrais positifs}}{\text{faux négatifs} + \text{vrais positifs}}$$

```
# Sensibilité
sensibilite = vrai_positif/(faux_negatif+vrai_positif)
print('Sensibilité : %.2f' % (sensibilite))

Sensibilité : 0.47
```

La précision

Elle indique la proportion de vrais positifs parmi les individus qui ont été classés positifs

$$\text{précision} = \frac{d}{b+d} = \frac{\text{vrais positifs}}{\text{faux positifs} + \text{vrais positifs}}$$

Elle estime la probabilité d'un individu d'être réellement positif lorsque le modèle le classe comme tel. Dans certains domaines, on parle de **valeur prédictive positive (VPP)**.

```
# Précision
precision = vrai_positif/(faux_positif+vrai_positif)
print('Précision : %.2f' % (precision))
```

Précision : 0.46

La spécificité

A l'inverse de la sensibilité, la spécificité indique la proportion de négatifs détectés

$$\text{Spécificité} = \frac{a}{a+b} = \frac{\text{vrais négatifs}}{\text{vrais négatifs} + \text{faux positifs}}$$

```
# Spécificité
specificite = vrai_negatif/(vrai_negatif+faux_positif)
print('Spécificité : %.2f' % (specificite))
```

Spécificité : 0.78

Le taux de faux positifs (TFP)

Il correspond à la proportion de négatifs qui ont été classés positifs, c'est-à-dire

$$\begin{aligned} \text{TFP} &= \frac{b}{a+b} = \frac{\text{faux positifs}}{\text{vrais négatifs} + \text{faux positifs}} \\ &= 1 - \text{Spécificité} \end{aligned}$$

```
# Taux de faux positifs (TFP)
taux_faux_positif = faux_positif/(vrai_negatif + faux_positif)
print('Taux de faux positifs : %.2f' % (taux_faux_positif))
```

Taux de faux positifs : 0.22

la F-Mesure

Indicateur très utilisé en recherche d'information, la F-Mesure synthétise (moyennes harmoniques) le rappel et la précision. L'importance accordée à l'une ou à l'autre est paramétrable avec β .

$$F_{\beta} = \frac{(1+\beta)^2 \times \text{rappel} \times \text{précision}}{\beta^2 \times \text{précision} + \text{rappel}}$$

Lorsque

- $\beta = 1$, on accorde la même importance au rappel et à la précision, la F-Mesure devient :

$$F_{\beta=1} = \frac{2 \times \text{rappel} \times \text{précision}}{\text{précision} + \text{rappel}}$$

- $\beta < 1$, on accorde plus d'importance à la précision par rapport au rappel. Une valeur fréquemment utilisée est $\beta = 0.5$, on accorde deux fois plus d'importance à la précision.
- $\beta > 1$, on accorde plus d'importance au rappel par rapport à la précision. Une valeur fréquemment rencontrée est $\beta = 2$.

Remarque :

La F-Mesure est une moyenne harmonique entre le rappel et la précision. En effet, nous pouvons l'écrire sous la forme suivante :

$$F = \frac{1}{\alpha \frac{1}{\text{précision}} + (1 - \alpha) \frac{1}{\text{rappel}}} \quad \text{où} \quad \beta^2 = \frac{1 - \alpha}{\alpha}$$

```
# F-mesure avec beta=1
f_mesure = 2*sensibilite*precision/(precision+sensibilite)
print('F-Mesure : %.2f' % (f_mesure))
```

F-Mesure : 0.47

Indice de Youden

L'indice de Youden est bien connu en biostatistique, moins en apprentissage supervisé. Il s'écrit :

$$IY = \text{Sensibilité} + \text{Spécificité} - 1$$

Son mérite est de caractériser le classifieur selon la sensibilité et la spécificité. Il prend la valeur maximum 1 lorsque le modèle est parfait. En effet, dans ce cas Sensibilité = 1 et Spécificité = 1. Il peut être utilisé pour comparer les performances de plusieurs modèles.

Son interprétation n'est pas très évidente en revanche. C'est le principal frein à son utilisation.

```
# Indice de Youden
indiceYouden = sensibilite + specificite - 1
print('Indice de Youden : %.2f' % (indiceYouden))
```

Indice de Youden : 0.25

Rapport de vraisemblance

Le rapport de vraisemblance décrit le surcroît de chances des positifs (par rapport aux négatifs) d'être classés positifs. Sa formule est la suivante :

$$L = \frac{\text{Sensibilité}}{1 - \text{Spécificité}}$$

Le rapport de vraisemblance ne dépend pas de la proportion des positifs. Il donne donc des indications valables même si l'échantillon n'est pas représentatif. Plus grande est sa valeur, meilleur est le modèle.

```
# Rapport de vraisemblance
rapport_vraisemblance = sensibilite/(1-specificite)
print('Rapport de vraisemblance : %.2f' % (rapport_vraisemblance))
```

Rapport de vraisemblance : 2.14

Sous Python, le package `sklearn` propose des outils pour le calcul de certains de ces indicateurs.

```
# Matrice de confusion
from sklearn.metrics import confusion_matrix
print(confusion_matrix(yTest, chdpred))
```

```
[[77 22]
 [21 19]]
```

```
# taux de succès
from sklearn.metrics import accuracy_score
print(round(accuracy_score(yTest, chdpred),2))
```

0.69

```
# Sensibilité ou rappel
from sklearn.metrics import recall_score
print(round(recall_score(yTest, chdpred),2))
```

0.48

```
# Précision
from sklearn.metrics import precision_score
print(round(precision_score(yTest, chdpred),2))
```

0.46

```
# F-mesure
from sklearn.metrics import f1_score
print(round(f1_score(yTest, chdpred),2))
```

0.47

15.1.3 Inconvénients de la matrice de confusion

Aussi intéressante qu'elle soit, la matrice de confusion présente une faiblesse importante : elle repose essentiellement sur les prédictions (seuil de séparation fixé sur la probabilité estimée d'avoir $Y = 1$). Également, la matrice de confusion et le taux d'erreur sont sensibles à l'importance relative des groupes, c'est-à-dire que le classement dans le groupe le plus important pèse davantage.

15.1.4 Diagramme de fiabilité

Principe de construction

Voici les étapes de la construction du diagramme de fiabilité :

1. Appliquer le classifieur sur les données pour obtenir le score $\hat{p}(\omega)$;
2. Trier le fichier selon le score croissant ;
3. Sur la base du score, subdiviser les données en intervalles (ex. $0.0 - 0.2, 0.2 - 0.4$, etc.) ;
4. Dans chaque intervalle, calculer la proportion de positifs ;
5. Dans le même temps, toujours dans chaque intervalle, calculer la moyenne des scores ;
6. Si les chiffres concordent dans chaque intervalle, les scores sont bien calibrés, le classifieur est de bonne qualité ;
7. Nous pouvons résumer l'information dans un graphique *nuage de points* appelé **diagramme de fiabilité**, avec en abscisse la moyenne des scores, en ordonnée la proportion de positifs ;
8. **Si les scores sont bien calibrés, les points devraient être alignés sur une droite la première bissectrice ;**
9. Les points s'écartant sensiblement de la première bissectrice doivent attirer notre attention.

Application

L'évaluation basée sur les scores analyse dans quelle mesure les probabilités d'affectation à la modalité cible (`chd=1`) fournis par le modèle, que l'on nommera « score » dans ce qui suit, sont de qualité satisfaisante. Nous pouvons les calculer en appliquant la fonction logistique `.cdf` sur le logit et affichons les valeurs pour les 10 premières observations.

```
# scores fournis par la régression
score = model.cdf(testfittedvalues)
# Affichage pour les 10 premiers individus
print(score[:10])

[0.65008295 0.31392218 0.4133731 0.04454239 0.49996307 0.68162482
 0.03824628 0.06461269 0.88788471 0.35254102]
```

Ces valeurs obtenues correspondent à :

$$p_{\hat{\beta}}(x) = \frac{1}{1 + e^{-x \cdot \hat{\beta}}}$$

```
# Vérification
phat = 1.0/(1.0 + np.exp(-1.0 * testfittedvalues))
print(phat[:10])

[0.65008295 0.31392218 0.4133731 0.04454239 0.49996307 0.68162482
 0.03824628 0.06461269 0.88788471 0.35254102]
```

On crée un dataframe incorporant la variable `score` et la variable cible `chd`. Puis on applique l'étape 3.

```
# dataframe temporaire
frame = pd.DataFrame({'chd':yTest,'score':score}, index=yTest.index)
# 5 intervalles de largeur égales
bins = pd.cut(frame.score, bins=5, include_lowest=True)
# intégrer dans le frame
frame['bins'] = bins
print(frame)

      chd     score           bins
row.names
248      0  0.650083  (0.56, 0.738]
125      0  0.313922  (0.202, 0.381]
35       0  0.413373  (0.381, 0.56]
299      0  0.044542  (0.02159999999999998, 0.202]
180      0  0.499963  (0.381, 0.56]
...
365      0  0.538456  (0.381, 0.56]
181      0  0.274695  (0.202, 0.381]
15       0  0.583368  (0.56, 0.738]
356      0  0.619243  (0.56, 0.738]
272      1  0.471582  (0.381, 0.56]

[139 rows x 3 columns]
```

A partir de ce dataset, nous pouvons calculer la moyenne des scores estimés dans chaque groupe délimité par les individus (étape 5) :

```
# moyenne des scores par groupe
mean_score = frame.pivot_table(index='bins', values='score', aggfunc='mean')
print(mean_score)

           score
bins
(0.02159999999999998, 0.202]  0.102009
(0.202, 0.381]                0.288593
(0.381, 0.56]                 0.469020
(0.56, 0.738]                 0.657792
(0.738, 0.917]                0.844265
```

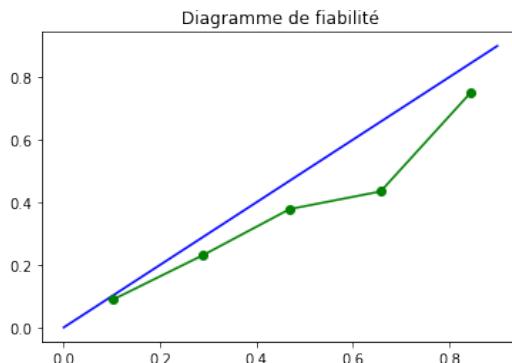
Puis la proportion des observations (scores observés) positives dans les mêmes groupes. La variable cible `chd` étant binaire (0/1), la moyenne fait très bien l'affaire.

```
# Moyenne des chd - qui équivaut à une proportion puisque
frame.chd = frame.chd.astype('float') # float
mean_chd = frame.pivot_table(index='bins', values='chd', aggfunc='mean')
print(mean_chd)
```

bins	chd
(0.02159999999999998, 0.202]	0.088889
(0.202, 0.381]	0.230769
(0.381, 0.56]	0.378378
(0.56, 0.738]	0.434783
(0.738, 0.917]	0.750000

Le diagramme de fiabilité est un graphique nuage de points opposant les scores estimés et observés. S'ils forment une droite, nous pouvons considérer que la modélisation est pertinente car le modèle arrive à approcher de manière satisfaisante l'appartenance aux classes des individus.

```
# Diagramme de fiabilité
plt.plot(np.arange(0,1,.1), np.arange(0,1,.1), 'b')
plt.plot(mean_score, mean_chd, 'go-')
plt.title('Diagramme de fiabilité')
plt.show()
```



15.2 Quelques tests statistiques

15.2.1 Test de Hosmer-Lemeshow

Construction du test

Concrètement, nous procédons de la manière suivante [voir [LSHJ13]] :

1. Appliquer le classifieur sur les données pour obtenir les estimations $\hat{p}(\omega)$;
2. Trier les données selon le score croissant ;

3. Subdividez les données en K groupes en se basant sur les quantiles (les auteurs proposent prioritairement les déciles ($K=10$)) (ex. les quantiles d'ordre 4 correspondent aux quartiles, les quantiles d'ordre 10 aux déciles, etc.);
4. Dans chaque groupe k ($k = 1, \dots, K$) d'effectif n_k , nous devons calculer plusieurs quantités :
 - n_{k1} : le nombre de positifs observés ;
 - n_{k0} : le nombre de négatifs observés ;
 - $\hat{n}_{k1} = \sum_{\omega} \hat{p}(\omega)$: la somme des scores des observations situées dans le groupe k . On la désigne comme la fréquence théorique des positifs dans le groupe ;
 - $\hat{p}_{k1} = \frac{\hat{n}_{k1}}{n_k}$: la moyenne des scores observés dans le groupe k ;
 - $\hat{n}_{k0} = n_k - \hat{n}_{k1}$: la fréquence théorique des négatifs.

5. Nous calculons alors la statistique de Hosmer et Lemeshow en utilisant une des formules suivantes :

$$\begin{aligned}\chi_K^2 &= \sum_k \left[\frac{(n_{k1} - \hat{n}_{k1})^2}{\hat{n}_{k1}} + \frac{(n_{k0} - \hat{n}_{k0})^2}{\hat{n}_{k0}} \right] \\ &= \sum_k \frac{n_k(n_{k1} - \hat{n}_{k1})^2}{\hat{n}_{k1}(n_k - \hat{n}_{k1})} \\ &= \sum_k \frac{(n_{k1} - \hat{n}_{k1})^2}{\hat{n}_{k1}(1 - \hat{p}_{k1})}\end{aligned}$$

Lorsque le modèle est correct (H_0), la statistique χ_K^2 suit approximativement une loi de χ^2 à $K - 2$ degrés de liberté.

Règle de décision :

Lorsque la probabilité critique du test (p-value) est plus grande que le risque choisi, le modèle issu de la régression logistique est accepté.

```
# Data frame
frame2 = pd.DataFrame({'chd':yTest, 'score':score})
# 10 intervalles de fréquences égales
bins2 = pd.qcut(frame2.score, q=10)
# Intégrer dans le frame2
frame2['bins'] = bins2
print(frame2)
```

	chd	score	bins
row.names			
248	0	0.650083	(0.595, 0.694]
125	0	0.313922	(0.275, 0.363]
35	0	0.413373	(0.363, 0.443]
299	0	0.044542	(0.0225, 0.0506]
180	0	0.499963	(0.499, 0.595]
...

```

365      0  0.538456  (0.499, 0.595]
181      0  0.274695  (0.195, 0.275]
15       0  0.583368  (0.499, 0.595]
356      0  0.619243  (0.595, 0.694]
272      1  0.471582  (0.443, 0.499]

[139 rows x 3 columns]

# Effectifs par groupe
eff_total = frame2.pivot_table(index='bins', values='chd',
                                 aggfunc='count').values[:,0]
print(eff_total)
[14 14 14 14 14 13 14 14 14 14 14]

# Somme des scores par groupe
sum_score = frame2.pivot_table(index='bins', values='score',
                                 aggfunc = 'sum').values[:,0]
print(sum_score)
[ 0.51546279  1.16957962  2.31060622  3.16059627  4.55784269  5.34697728
  6.62824134  7.51248542  9.12028315 11.00882421]

# Nombre de positifs par groupe
eff_positif = frame2.pivot_table(index='bins', values='chd',
                                   aggfunc='sum').values[:,0]
print(eff_positif)
[ 0  2  2  2  3  8  4  4  5 10]

# Nombre de négatif par groupe
eff_negatif = eff_total - eff_positif
print(eff_negatif)
[14 12 12 12 11  5 10 10  9  4]

# bloc 1
chi1 = np.sum((eff_positif-sum_score)**2/sum_score)
print(chi1)
8.060126762971196

# bloc 2
chi2 = np.sum((eff_negatif - (eff_total-sum_score))**2/((eff_total-sum_score)))
print(chi2)
8.040767450794648

# Statistique de Hosmer-Lemeshow
HL_stat = chi1 + chi2
print('Hosmer-Lemeshow statistic : %.4f' % (HL_stat))
Hosmer-Lemeshow statistic : 16.1009

# pvalue
import scipy.stats as stats
pvalue = 1.0 - stats.chi2.cdf(HL_stat,8)
print('p-value associé : %.4f' % (pvalue))

p-value associé : 0.0410

```

15.2.2 Test de Mann-Whitney

Le test de Mann-Whitney est de test non paramétrique de comparaison de populations [voir [TM87]].

Pourquoi un test de comparaison de populations ?

La discrimination sera d'autant meilleure que les positifs ont un score élevé et les négatifs un score faible. Dans les tableaux où l'on trie les observations selon un score croissant, les négatifs seraient agglutinés en haut, les positifs en bas. On peut illustrer ce point de vue en comparant les distributions des scores conditionnellement aux classes d'appartenance. Lorsque le modèle est de bonne qualité, les distributions conditionnelles des scores sont bien différenciées ; dans le cas contraire, elles sont confondues.

Il faut pouvoir quantifier cette impression visuelle. Pour ce faire, un test de comparaison de populations semble approprié. L'objectif est de répondre à la question : **est-ce que les positifs ont des scores (significativement) plus élevés que les négatifs ?** Le test non paramétrique de Mann-Whitney est celui que l'on retient le plus souvent dans la littérature. Dans le cadre de l'apprentissage supervisé, il convient surtout parce qu'il est en relation avec le critère AUC (Area Under Curve) associé à la courbe ROC. Les formules associées à ce test sont les suivantes :

1. A partir des scores, nous calculons le rang $r(\omega)$ des individus dans l'ensemble de l'échantillon, sans distinction de classes ;
2. Nous calculons alors les scores conditionnelles de rang :

- Pour les positifs :

$$r_+ = \sum_{\omega:y(\omega)=1} r(\omega)$$

- Pour les négatifs :

$$r_- = \sum_{\omega:y(\omega)=0} r(\omega)$$

3. Nous en déduisons les statistiques :

$$U_+ = r_+ - \frac{n_+(n_+ + 1)}{2} \quad \text{et} \quad U_- = r_- - \frac{n_-(n_- + 1)}{2}$$

4. La statistique de Mann-Whitney correspond au minimum de ces deux quantités, soit

$$U = \min(U_+, U_-)$$

5. Sous H_0 , les distributions sont confondues, la statistique centrée et réduite Z suit une loi normale $\mathcal{N}(0,1)$

$$Z = \frac{U - \frac{n_+ n_-}{2}}{\sqrt{\frac{1}{12} (n_+ + n_- + 1) n_+ n_-}}$$

Remarque :

- Il s'agit usuellement d'un **test bilatéral**. Mais en réalité on imagine mal que les positifs puissent présenter des scores significativement plus faibles que les négatifs. Ou alors, il faudrait prendre le complémentaire à 1 des valeurs produites par le classifieur.
- Deux types de corrections** peuvent être introduites pour préciser les résultats dans certaines circonstances :
 - Une correction de **continuité** lorsque les effectifs sont faibles ;
 - Une correction du **dénominateur de la statistique centrée et réduite** lorsqu'il y a des ex-aequo, on utilise habituellement les rangs moyens.

```
# effectif par groupe
eff = yTest.value_counts()
n_moins, n_plus = eff[0], eff[1]
# Affichage
print(eff)
print(n_moins)
print(n_plus)

0    99
1    40
Name: chd, dtype: int64
99
40

# frame
frame3 = pd.DataFrame({'chd':yTest, 'score':score}, index=XTest.index)
print(frame3)

      chd      score
row.names
248        0  0.650083
125        0  0.313922
35         0  0.413373
299        0  0.044542
180        0  0.499963
...
       ...
365        0  0.538456
181        0  0.274695
15         0  0.583368
356        0  0.619243
272        1  0.471582

[139 rows x 2 columns]

# trie croissante
sort_score = frame3.sort_values(by='score', ascending=True)
sort_score.loc[:, 'rang'] = np.arange(1, len(yTest)+1)
print(sort_score)
```

```

      chd      score   rang
row.names
292       0  0.023481     1
332       0  0.024204     2
14        0  0.028989     3
49        0  0.030610     4
437       0  0.032354     5
...
       ...    ...
130       0  0.828835   135
162       1  0.883848   136
47        1  0.887885   137
112       1  0.893165   138
126       1  0.916979   139

[139 rows x 3 columns]

# somme des rangs de chaque classe
rang_sum = sort_score.groupby('chd')['rang'].sum()
print(rang_sum)

chd
0      5992
1      3738
Name: rang, dtype: int32

# Effectif par groupe
r_moins,r_plus = rang_sum[0], rang_sum[1]
print(r_moins)
print(r_plus)

5992
3738

# Statistiques
U_moins = r_moins - (n_moins*(n_moins+1)/2)
print('U_ : %.2f' % (U_moins))
U_plus = r_plus - (n_plus*(n_plus+1)/2)
print('U+ : %.2f' % (U_plus))
U = min(U_moins, U_plus)
print('U : %.2f' % (U))

U_ : 1042.00
U+ : 2918.00
U : 1042.00

# Statistique de Mann-Whitney
num_mn = U - (n_plus*n_moins)/2
deno_mn = np.sqrt((1/12)*(n_moins+n_plus+1)*n_moins*n_plus)
# Statistique de Mann-Whitney
MN_stat = num_mn/deno_mn
print('Mann-Whitney statistic : %.4f' % (MN_stat))

```

```

# p-value
pvalue = 1.0-stats.norm.cdf(MN_stat)
print('p-value associée : %.4f' % (pvalue))

Mann-Whitney statistic : -4.3640
p-value associée : 1.0000

```

15.3 Quelques courbes d'évaluation

15.3.1 La courbe ROC

La courbe ROC est un outil d'évaluation et de comparaison des modèles, dans le cas où la variable d'intérêt Y est qualitative binaire. Cette courbe permet de savoir si un modèle M1 sera toujours meilleur qu'un modèle M2 quelle que soit la matrice de coût. La courbe ROC est opérationnelle même dans le cas des distributions très déséquilibrées. C'est un outil graphique qui permet de visualiser les performances. Un seul coup d'œil doit permettre de voir le(s) modèle(s) susceptible(s) de nous intéresser. Un indicateur synthétique associé = **aire sous la courbe ROC**.

Principe de la courbe ROC

$P(Y = 1|X = x) \geq P(Y = 0|X = x)$ équivaut à une règle d'affectation $P(Y = 1|X = x) \geq 0.5$ (seuil = 0.5). Cette règle d'affectation fournit une matrice de confusion $MC_{0.5}$, et donc 2 indicateurs $TVP_{0.5}$ et $TFP_{0.5}$. Si nous choisissons un autre seuil (0.6 par exemple), nous obtiendrons une matrice de confusion $MC_{0.6}$ et donc $TVP_{0.6}$ et $TFP_{0.6}$...etc.

L'idée de la courbe ROC est de faire varier le seuil de 1 à 0 et, pour chaque cas, calculer le TVP et le TFP que l'on reporte dans un graphique : en abscisse le TFP, en ordonnée le TVP.

Modèle au hasard = diagonale Modèle parfait = passe par les points (0,0), (0,1) et (1,1) car les scores des individus avec $Y = 1$ sont tous supérieurs aux scores des individus avec $Y = 0$.

Dans de nombreuses applications, la courbe ROC fournit des informations plus intéressantes sur la qualité de l'apprentissage que le simple taux d'erreur. C'est surtout vrai lorsque les classes sont très déséquilibrées, et lorsque le coût de mauvaise affectation est susceptible de modifications. Il faut néanmoins que l'on ait une classe cible (par exemple $Y = 1$) clairement identifiée et que la méthode d'apprentissage puisse fournir un **score** proportionnel à $P(Y = 1|X)$.

Il existe plusieurs façons de faire la courbe ROC sous python. On peut utiliser la fonction **plot_roc_curve** qui se trouve dans le module **metrics** de scikit-learn ou faire des manipulations avec la librairie **matplotlib**. C'est cette deuxième approche qui sera utilisée.

Construction de la courbe ROC

La courbe ROC est un peu lourde à mettre en place. Dans la pratique, il n'est pas nécessaire de construire explicitement la matrice de confusion, nous procédons de la manière suivante :

1. Calculer le score de chaque individu à l'aide du modèle de prédiction ;
2. Trier le fichier selon un **score décroissant** ;

3. Considérons qu'il n'y a pas d'ex-aquo. Chaque valeur du score être potentiellement un seuil s . Pour toutes les observations dont le score est supérieur ou égal à s , les individus dans la partie haute du tableau, nous pouvons comptabiliser le nombre de positifs $n_+(s)$ et le nombre de négatifs $n_-(s)$. Nous en déduisons $TVP = \frac{n_+(s)}{n_+}$ et $TFP = \frac{n_-(s)}{n_-}$
4. La courbe ROC correspond au graphique nuage des points qui relie les couples (TVP, TFP). Le premier point est forcément (0,0). Le dernier est (1,1).

Remarque :

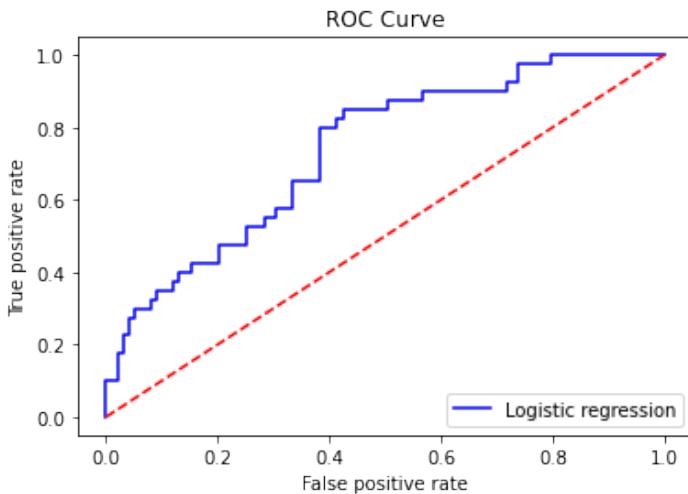
Deux situations extrêmes peuvent survenir :

- La discrimination est **parfaite**. Tous les positifs sont situés devant les négatifs, la courbe ROC est collée aux extrémités Ouest et Nord du repère.
- Les scores sont totalement inopérants, le classifieur attribue des valeurs au hasard, dans ce cas les positifs et les négatifs sont mélangés. La courbe ROC se confond avec la première bissectrice.

```
# False positive rate - True positive rate
from sklearn.metrics import roc_curve
fpr, tpr, threshold1 = roc_curve(yTest, testfittedvalues, pos_label=1)
# Affichage
print(fpr[:10])
print(tpr[:10])

[0.          0.          0.          0.02020202 0.02020202 0.03030303
 0.03030303 0.04040404 0.04040404 0.05050505]
[0.      0.025 0.1    0.1    0.175 0.175 0.225 0.225 0.275 0.275]

# Courbe roc
plt.plot(fpr,tpr,color='b', label='Logistic regression')
plt.plot([0,1],[0,1],'r--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC Curve')
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.legend(loc='lower right')
plt.show()
```



Le critère AUC

Il est possible de caractériser numériquement la courbe ROC en calculant la surface située sous la courbe : c'est le **critère AUC**. Elle exprime la probabilité de placer un individu positif devant un négatif. Ainsi, dans le cas d'une discrimination parfaite, les positifs sont sûrs d'être placés devant les négatifs, nous avons $AUC = 1$. A contrario, si le classifieur attribue des scores au hasard, il y a autant de chances de placer un positif devant un négatif que l'inverse, la courbe ROC se confond avec la première bissectrice, nous avons $AUC = 0.5$. C'est la situation de référence, notre classifieur doit faire mieux. On propose généralement différents paliers pour donner un ordre d'idée sur la qualité de la discrimination.

Elle indique la probabilité pour que la fonction SCORE place un positif devant un négatif (dans le meilleur des cas $AUC = 1$ pour le modèle parfait). Si SCORE classe au hasard les individus (c'est-à-dire le modèle de prédiction ne sert à rien), $AUC = 0.5$ symbolisé par la diagonale principale dans le graphique.

Dans scikit-learn, on peut utiliser la fonction **auc** du module **metrics** et y introduire le taux de faux positif et le taux de vrai positif.

```
# Aire sous la courbe
from sklearn.metrics import auc
area = auc(fpr,tpr)
print('ROC AUC score : %.2f' % (area))

ROC AUC score : 0.74
```

ou encore en utilisant la fonction **roc_auc_score** en y introduisant l'échantillon test et les probabilités estimées.

```
# Aire sous la courbe
from sklearn.metrics import roc_auc_score
area2 = roc_auc_score(yTest, testfittedvalues)
print('ROC AUC score : %.2f' % (area2))

ROC AUC score : 0.74
```

Critère AUC et statistique de Mann-Withney

Il existe une relation entre la statistique U_+ de Mann-Withney et le critère AUC. La meilleure justification est certainement du côté de l'interprétation de ces quantités sous l'angle des comparaisons par paires. La relation est la suivante :

$$AUC = \frac{U_+}{n_- \times n_+} \quad (15.1)$$

```
# Aire sous la courbe
area3 = U_plus/(n_moins*n_plus)
print('ROC AUC score : %.2f' % (area3))

ROC AUC score : 0.74
```

15.3.2 La courbe LIFT

Principe de la courbe LIFT

Encore appelée **courbe de gain**, cette courbe représente la proportion de vrais positifs en fonction des individus sélectionnés, lorsqu'on fait varier le seuil. Sa forme dépend du taux de positif a priori. Elle a même ordonnée que la courbe ROC, mais une abscisse généralement plus grande. La courbe LIFT est généralement sous la courbe ROC.

On peut, soit utiliser la fonction **plot_cumulative_gain** de la librairie **scikitplot** de python afin de pouvoir avoir une représentation graphique de la courbe LIFT, soit construire la courbe à l'aide des librairies numpy, pandas et matplotlib. C'est la deuxième approche qui sera employée. L'inconvénient de cette approche est qu'elle est longue et nécessite plusieurs manipulations.

Tout d'abord, on transforme en 0 et 1 la cible de l'échantillon test et on récupère les éléments de la deuxième colonne qui correspond à l'indice 1.

```
# Transformation en 0 et 1 # Eléments de la 2ème colonne
pos = pd.get_dummies(yTest).values[:,1]
print(pos)

[0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 0 1 0 1 0 1 1 0 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 0 1 0 1 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

On calcule le nombre total de positif.

```
# Nombre total de positif
n_pos = np.sum(pos)
print(n_pos)
```

40

On tri selon le score croissant puis on inverse selon le score décroissant.

```
# Index pour tri selon le score croissant
index = np.argsort(score)[::-1]
print(index[:10])
```

```
[ 48  52    8   16   55  113   96  132   86   63]
```

On tri les individus.

```
# Trie des individus
```

```
sort_pos = pos[index]
```

```
print(sort_pos[:10])
```

```
[1 1 1 1 0 0 1 1 1 0]
```

On fait la somme cumulée

```
# Somme cumulée
```

```
cum_pos = np.cumsum(sort_pos)
```

```
print(cum_pos[:10])
```

```
[1 2 3 4 4 4 5 6 7 7]
```

On calcule le rappel

```
# Rappel
```

```
rappel = cum_pos/n_pos
```

```
print(rappel[:10])
```

```
[0.025 0.05 0.075 0.1 0.1 0.1 0.125 0.15 0.175 0.175]
```

On définit la taille de l'échantillon test et celle de la cible.

```
# Taille d'échantillon test
```

```
n = yTest.shape[0]
```

```
# Taille de cible - sequence de valeurs
```

```
taille = np.arange(start = 1, stop = n+1, step = 1)
```

On fait passer le rappel en proportion

```
# Passer en proportion
```

```
taille = taille/n
```

```
print(taille[:10])
```

```
[0.00719424 0.01438849 0.02158273 0.02877698 0.03597122 0.04316547
```

```
0.05035971 0.05755396 0.0647482 0.07194245]
```

pour finalement avoir le graphique suivant :

```
# Graphique
```

```
plt.plot(taille, rappel, 'b', label='Logistic Regression')
```

```
plt.plot([0, 1], [0, 1], 'r--')
```

```
plt.xlim([-0.05, 1.05])
```

```
plt.ylim([-0.05, 1.05])
```

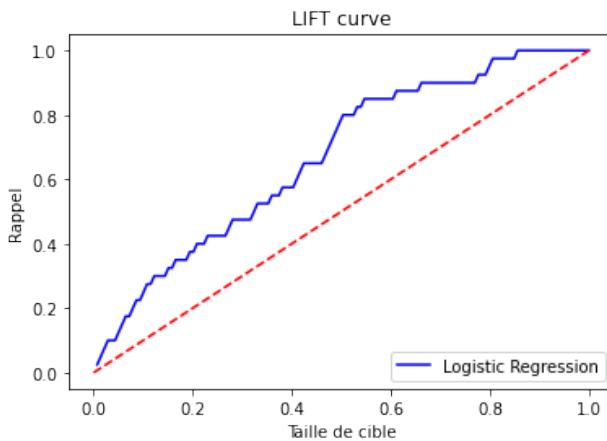
```
plt.title ("LIFT curve")
```

```
plt.xlabel("Taille de cible")
```

```
plt.ylabel("Rappel")
```

```
plt.legend(loc='lower right')
```

```
plt.show()
```



Aire sous la courbe LIFT

Il existe un lien entre les aires sous les courbes LIFT et ROC. L'aire sous la courbe LIFT, notée AUL, s'exprime simplement à priori par :

$$AUL = \frac{p}{2} + (1-p)AUC$$

où $p = \text{Proba}(G1)$ = probabilité a priori de l'événement $Y = 1$ dans la population.

```
# proportion
prop = np.sum(yTest)/yTest.shape[0]
# Aire AUL
area_lift = (prop/2)+(1-prop)*area
print('LIFT AUL score : %.2f' % (area_lift))

LIFT AUL score : 0.67
```

15.3.3 La courbe rappel-prévision

Principe de la courbe

La courbe rappel-précision est très utilisée en recherche d'information (en anglais, *information retrieval*). Suite à une requête, nous obtenons un ensemble d'individus que nous appellerons la « cible », nous sommes face à deux exigences contradictoires : nous aimerais retrouver une fraction élevée des positifs potentiels (**rappel**) ; nous aimerais que la cible ne contienne que des positifs (**précision**). La courbe traduit l'arbitrage entre ces deux critères lorsque l'on fait varier le seuil d'affectation s .

Elle est conceptuellement proche de la courbe ROC. Pour chaque valeur de s , nous formons (virtuellement) la matrice de confusion et nous calculons les deux indicateurs. Il y a quand même une différence très importante. La précision étant un « profil-colonne » de la matrice de confusion, il faut donc travailler sur un échantillon représentatif (la proportion des positifs $\frac{n_+}{n}$ doit être le reflet de la probabilité d'être positif p) pour pouvoir l'exploiter convenablement. Si cette

condition est respectée, elle paraît plus adaptée que la courbe ROC lorsque les classes sont très déséquilibrées (la proportion des positifs est très faible), notamment pour différencier le comportement des algorithmes d'apprentissage supervisé. Pour élaborer la **courbe rappel-précision**, nous procérons comme suit :

1. Calculer le score de chaque individu ;
2. Trier les données selon un score décroissant ;
3. Admettons qu'il n'y a pas d'ex-aquo, chaque valeur du score est un outil potentiel s . Pour les individus situés dans la partie haute du tableau c'est-à-dire dont le score est supérieur ou égal à s , il s'agit de la cible, nous comptabilisons le nombre de positifs $n_+(s)$ et le nombre total d'observations $n(s)$.
4. Nous en déduisons le $rappel(s) = \frac{n_+(s)}{n_+}$ et la $precision(s) = \frac{n_+(s)}{n(s)}$

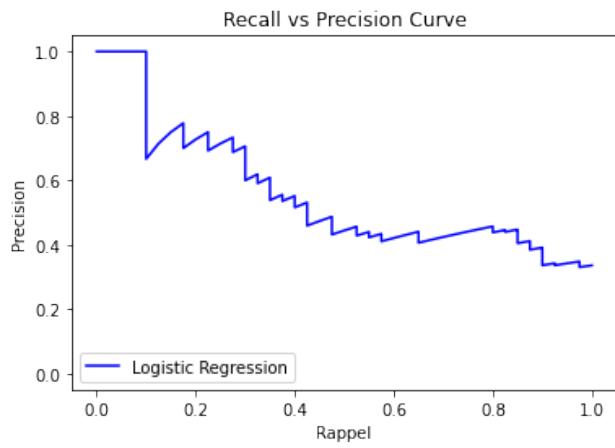
Remarque

Dans les parties hautes du tableau, lorsque le seuil est élevé, la taille de la cible sera réduite. La précision sera forte, dans la cible ne seront présents que des positifs ; mais le rappel sera faible, une faible fraction de l'ensemble des positifs y sont inclus. A mesure que s diminue, la taille de la cible augmente, elle sera de plus en plus polluée (la précision diminue) mais intégrera une plus grande fraction des positifs (le rappel augmente). La courbe est donc globalement décroissante, mais elle n'est pas forcément monotone.

```
# Precision - recall curve
from sklearn.metrics import precision_recall_curve
precision, recall, threshold2 = precision_recall_curve(yTest,testfittedvalues)
```

On construit le graphique suivant :

```
# Graphique
plt.plot(recall, precision, 'blue', label='Logistic Regression')
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.title('Recall vs Precision Curve')
plt.xlabel('Rappel')
plt.ylabel('Precision')
plt.legend(loc='lower left')
plt.show()
```



CHAPITRE 16

RÉGRESSION LOGISTIQUE MULTINOMIALE

Sommaire

16.1 Le modèle polytomique nominal	368
16.2 Évaluation des classifieurs	376
16.3 Test sur les coefficients de la régression multinomiale	381
16.4 Modèle multinomial et régresseurs catégorielles	390

16.1 Le modèle polytomique nominal

16.1.1 Modèle polytomique nominal

C'est le modèle utilisé par défaut lorsque les différentes modalités de Y n'ont pas de hiérarchie entre elles. On s'inspire du modèle logistique binaire :

$$\begin{cases} \mathbb{P}(Y = 0/X) = \frac{1}{1 + e^{\beta X}} \\ \mathbb{P}(Y = 1/X) = \frac{e^{\beta X}}{1 + e^{\beta X}} \end{cases}$$

d'où l'on tire :

$$\mathbb{P}(Y = 1/X) = p(X) = e^{\beta X} \mathbb{P}(Y = 0/X)$$

On peut étendre ce modèle au cas où Y est à valeurs dans $\{0, 1, \dots, K\}$ en prenant (par exemple) $Y = 0$ comme modalité de référence. On aboutit au modèle spécifié par :

$$\mathbb{P}(Y = k/X) = p_{\beta}^{(k)}(X) = e^{\beta^{(k)} X} \mathbb{P}(Y = 0/X)$$

où $\beta^{(k)}$ est le vecteur des $p + 1$ coefficients des régresseurs pour la modalité $k \geq 1$ de l'output Y. Comme la somme des probabilités de sortie vaut 1, on trouve :

$$\left\{ \begin{array}{l} \mathbb{P}(Y = 0/X) = p_{\beta}^{(0)}(X) = \frac{1}{\sum\limits_{r=0}^K e^{\beta^{(r)} X}} \\ \mathbb{P}(Y = k/X) = p_{\beta}^{(k)}(X) = \frac{e^{\beta^{(k)} X}}{\sum\limits_{r=0}^K e^{\beta^{(r)} X}} \end{array} \right.$$

Remarque :

- $K = 1$ ramène au modèle logistique
- La première équation se ramène à la seconde en posant $\beta^{(0)} = 0$ (vecteur nul). En d'autres termes, $Y = 0$ a été pris comme modalité de référence.

On en déduit que ce choix modifie les valeurs des paramètres du modèle, mais pas la nature du modèle en lui-même. En particulier, les valeurs estimées de $\mathbb{P}(Y = k/X)$ ne dépendent pas du choix de la modalité de référence.

Remarquons que pour deux modalités j et k avec $j \neq k$, on déduit l'écriture du modèle que :

$$\frac{\mathbb{P}(Y = k/X)}{\mathbb{P}(Y = j/X)} = e^{(\beta^{(k)} - \beta^{(j)})X}$$

Il y a $\frac{p+1}{K}$ paramètres à estimer dans ce modèle, ce qui peut vite faire beaucoup.

16.1.2 Données

Dans ce chapitre, on utilisera une base de données qui contient des données de 4177 individus pour lesquels on souhaite prédire le sexe de l'individu. La base est disponible [ici](#).

```
# load dataset
import pandas as pd
abalone = pd.read_csv('abalone.txt', sep=',')
print(abalone.head())

```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7

```
# informations
print(abalone.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 Column      Non-Null Count   Dtype  
--- 
0  Sex          4177 non-null    object  
1  Length       4177 non-null    float64 
2  Diameter     4177 non-null    float64 
3  Height       4177 non-null    float64 
4  Whole weight 4177 non-null    float64 
5  Shucked weight 4177 non-null    float64 
6  Viscera weight 4177 non-null    float64 
7  Shell weight 4177 non-null    float64 
8  Rings        4177 non-null    int64  
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
None
```

Notre variable cible `Sex` est de type `object`. Nous la transformons en catégorielle.

```
# change to category
abalone['Sex'] = abalone['Sex'].astype('category')
print(abalone.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 Column      Non-Null Count   Dtype    
--- 
0  Sex          4177 non-null    category 
1  Length       4177 non-null    float64  
2  Diameter     4177 non-null    float64  
3  Height       4177 non-null    float64  
4  Whole weight 4177 non-null    float64  
5  Shucked weight 4177 non-null    float64  
6  Viscera weight 4177 non-null    float64  
7  Shell weight 4177 non-null    float64  
8  Rings        4177 non-null    int64  
dtypes: category(1), float64(7), int64(1)
memory usage: 265.4 KB
None
```

16.1.3 Estimation et interprétations des paramètres

Les paramètres $\beta^k, k = 1, \dots, K - 1$ sont estimés par maximum de vraisemblance. La log-vraisemblance s'écrit :

$$\ln L(Y, \beta) = \sum_{i=1}^n \sum_{k=0}^K 1_{\{Y_i=k\}} \left(\beta^{(r)} X_i - \ln \sum_{r=0}^K e^{\beta^{(r)} X_i} \right)$$

On en déduit l'EMV des $\beta_j^{(k)}$ par des méthodes numériques. L'EMV a les mêmes propriétés que le modèle logistique (Intervalle de confiance, test de significativité et adéquation analogues).

Application 1 : Régresseurs quantitatifs

Nous entraînons notre modèle sur l'ensemble de nos données.

```
# Importation des classes de calcul
import statsmodels.api as sm
fullmodel=sm.MNLogit(endog=abalone.Sex,
                      exog=sm.add_constant(abalone[abalone.columns[1:]]))
fullresult = fullmodel.fit()
# Summary
print(fullresult.summary())
```

Optimization terminated successfully.

Current function value: 0.854590
Iterations 8

MNLogit Regression Results

Dep. Variable:	Sex	No. Observations:	4177			
Model:	MNLogit	Df Residuals:	4159			
Method:	MLE	Df Model:	16			
Date:	Wed, 03 Nov 2021	Pseudo R-squ.:	0.2204			
Time:	07:02:47	Log-Likelihood:	-3569.6			
converged:	True	LL-Null:	-4578.9			
Covariance Type:	nonrobust	LLR p-value:	0.000			
Sex=I	coef	std err	z	P> z	[0.025	0.975]
const	2.8410	0.505	5.627	0.000	1.851	3.831
Length	17.6817	2.864	6.174	0.000	12.069	23.295
Diameter	-13.0489	3.526	-3.700	0.000	-19.960	-6.137
Height	-8.1076	3.447	-2.352	0.019	-14.864	-1.351
Whole weight	-6.3995	1.728	-3.703	0.000	-9.787	-3.012
Shucked weight	5.2464	1.972	2.661	0.008	1.382	9.111
Viscera weight	-13.2179	2.653	-4.983	0.000	-18.417	-8.019
Shell weight	5.5951	2.506	2.233	0.026	0.684	10.506
Rings	-0.1967	0.026	-7.498	0.000	-0.248	-0.145
Sex=M	coef	std err	z	P> z	[0.025	0.975]
const	2.5212	0.432	5.838	0.000	1.675	3.368
Length	-1.0157	1.953	-0.520	0.603	-4.844	2.813
Diameter	-4.9630	2.355	-2.107	0.035	-9.579	-0.347
Height	-3.1768	1.964	-1.617	0.106	-7.026	0.673

Whole weight	-0.1314	0.737	-0.178	0.858	-1.576	1.313
Shucked weight	3.0484	0.884	3.450	0.001	1.317	4.780
Viscera weight	-2.1665	1.283	-1.689	0.091	-4.680	0.347
Shell weight	0.4747	1.131	0.420	0.675	-1.741	2.691
Rings	0.0059	0.016	0.379	0.705	-0.025	0.037

Remarque :

Par défaut, Python prend comme modalité de référence la première lettre suivant l'ordre alphabétique (ou la plus petite valeur suivant l'ordre numérique). Par conséquent, la modalité $Y = F$ est prise comme modalité de référence. Notre modèle à estimer s'écrit :

$$\begin{cases} \ln \left(\frac{\mathbb{P}(Y = I/X = x)}{\mathbb{P}(Y = F/X = x)} \right) = \beta_0^I + \beta_1^I \times Length + \cdots + \beta_8^I \times Rings \\ \ln \left(\frac{\mathbb{P}(Y = M/X = x)}{\mathbb{P}(Y = F/X = x)} \right) = \beta_0^M + \beta_1^M \times Length + \cdots + \beta_8^M \times Rings \end{cases}$$

Nous avons les modèles (logit) estimés suivants :

$$\begin{cases} \ln \left(\frac{\mathbb{P}(Y = I/X = x)}{\mathbb{P}(Y = F/X = x)} \right) = 2.8410 + 17.6817 \times Length + \cdots - 0.1967 \times Rings \\ \ln \left(\frac{\mathbb{P}(Y = M/X = x)}{\mathbb{P}(Y = F/X = x)} \right) = 2.5212 - 1.0157 \times Length + \cdots + 0.0059 \times Rings \end{cases}$$

Pour vérifier la concordance de nos résultats, nous affichons les résultats obtenus avec le logiciel R et la librairie `nnet`.

```
# Multinomial logistic regression -----
library(nnet)
mydata$Sex <- relevel(mydata$Sex, ref = 'F') # modalité de ref
mymodel <- multinom(formula = "Sex~.", data=mydata)
print(summary(mymodel))

Call:
multinom(formula = "Sex~.", data = mydata)

Coefficients:
              (Intercept) Length Diameter Height Whole.weight Shucked.weight
I            2.840238 17.677247 -13.040889 -8.105880   -6.4105702      5.256506
M            2.520820 -1.014589 -4.962426 -3.179209   -0.1321433      3.049042
               Viscera.weight Shell.weight      Rings
I            -13.204185  5.6052962 -0.196676428
M            -2.165372  0.4750846  0.005941886

Std. Errors:
              (Intercept) Length Diameter Height Whole.weight Shucked.weight
I            0.5048827 2.863819 3.526356 3.447535    1.7285912      1.9718969
M            0.4318683 1.953324 2.355216 1.964657    0.7368581      0.8836341
               Viscera.weight Shell.weight      Rings
I            2.652651   2.506042 0.02623068
```

```
M      1.282568    1.130567 0.01569697
```

```
Residual Deviance: 7139.243
AIC: 7175.243
```

On calcule la valeur de z et la p-value associée :

```
# 2-tailed Z-test -----
z <- summary(mymodel)$coefficients/summary(mymodel)$standard.errors
p <- (1-pnorm(abs(z),0,1))*2
print(round(p,3))

(Intercept) Length Diameter Height Whole.weight Shucked.weight Viscera.weight
I          0 0.000    0.000  0.019       0.000      0.008      0.000
M          0 0.603    0.035  0.106       0.858      0.001     0.091
Shell.weight Rings
I          0.025 0.000
M          0.674 0.705
```

Les résultats sont identiques.

16.1.4 Odds et odds ratios

Les odds ratios n'apparaissent généralement pas dans les sorties logiciels pour le modèle multinomial : il faut donc les calculer à la main en prenant garde du codage particulier des variables explicatives qualitatives.

1. Odds

On rappelle que l'odds d'un évènement $Y = k$ sachant $X = x$ est égal au rapport $\frac{\mathbb{P}(Y = k/X = x)}{\mathbb{P}(Y \neq k/X = x)}$

Dans le cas du multinomial, on définit l'odds d'un évènement $Y = k$ contre un évènement $Y = l$ par :

$$\text{odds}(x, Y = k \text{ vs } Y = l) = \frac{\mathbb{P}(Y = k/X = x)}{\mathbb{P}(Y = l/X = x)} = \exp((\beta^k - \beta^l)' x)$$

Dans le cas où $Y = l$ est défini comme modalité de référence ($l = 0$), on définit l'odds de la $Y = k$ sachant $X = x$ par :

$$\text{odds}(x, Y = k \text{ vs } Y = 0) = \frac{\mathbb{P}(Y = k/X = x)}{\mathbb{P}(Y = 0/X = x)} = \exp(\beta^k x)$$

Notons que cet odds est défini relativement à la modalité de référence et il s'agit en fait d'un rapport de probabilités, et non à proprement parler d'une cote (odds). Il faudra donc être extrêmement vigilant quant aux interprétations données aux résultats.

2. Odds ratios

Pour deux observations x_1 et x_2 , on définit alors l'odds ratio par :

$$\text{OR}(x_1, x_2, Y = k \text{ vs } Y = l) = \frac{\frac{\mathbb{P}_\beta(Y = k/X = x_1)}{\mathbb{P}_\beta(Y = l/X = x_1)}}{\frac{\mathbb{P}_\beta(Y = k/X = x_2)}{\mathbb{P}_\beta(Y = l/X = x_2)}} = \exp((\beta^k - \beta^l)'(x_1 - x_2))$$

Ainsi, l'odds-ratio de $Y = k$ par rapport à $Y = 0$ pour deux valeurs possibles x_1 et x_2 des régresseurs est :

$$\text{OR}(x_1, x_2, Y = k \text{ vs } Y = 0) = \frac{\frac{\mathbb{P}(Y = k/X = x_1)}{\mathbb{P}(Y = 0/X = x_1)}}{\frac{\mathbb{P}(Y = k/X = x_2)}{\mathbb{P}(Y = 0/X = x_2)}} = \exp(\beta^k(x_1 - x_2))$$

Cette valeur dépend de la modalité de référence. Son interprétation est assez délicate, notamment quand il n'y a pas de modalité de référence « naturelle ».

On extrait les coefficients d'estimation

```
# coefficients d'estimation
coef = fullresult.params.round(4)
coef.columns = ['I', 'M']
print(coef)
```

	I	M
const	2.8410	2.5212
Length	17.6817	-1.0157
Diameter	-13.0489	-4.9630
Height	-8.1076	-3.1768
Whole weight	-6.3995	-0.1314
Shucked weight	5.2464	3.0484
Viscera weight	-13.2179	-2.1665
Shell weight	5.5951	0.4747
Rings	-0.1967	0.0059

Ensuite on applique l'exponentielle pour obtenir les odds-ratios.

```
# odds ratio
import numpy as np
odds_ratio = np.exp(coef)
print(odds_ratio)
```

	I	M
const	1.713289e+01	12.443520
Length	4.776005e+07	0.362149
Diameter	2.152458e-06	0.006992
Height	3.012410e-04	0.041719
Whole weight	1.662388e-03	0.876867

Shucked weight	1.898815e+02	21.081587
Viscera weight	1.817770e-06	0.114578
Shell weight	2.691046e+02	1.607532
Rings	8.214370e-01	1.005917

16.1.5 Quelques statistiques de test

1. Pseudo- R^2 de McFadden

Notons par LL_M la vraisemblance du modèle étudié, le pseudo- R^2 de McFadden est défini de la même manière que pour la régression binaire, à savoir :

$$R_{MF}^2 = 1 - \frac{LL_M}{LL_0}$$

le pseudo- R^2 de McFadden varie entre 0 (modèle pas meilleur que le trivial) et 1 (modèle parfait).

```
#R2 de McFadden
r2Mcfadden = 1 - fullresult.llf / fullresult.llnull
print("R2 de McFadden : %.2f" % (r2Mcfadden))

R2 de McFadden : 0.22

Statsmodels retourne la valeur du pseudo- $R^2$  de McFadden :!

#qui est fourni directement par l'outil
print("R2 de McFadden : %.2f" % (fullresult.prsquared))

R2 de McFadden : 0.22
```

2. Test du rapport de vraisemblance

Le test du rapport de vraisemblance consiste à comparer 2 déviances. Pour l'évaluation globale il s'agit de confronter celles du modèle étudié et du modèle trivial. La statistique du test s'écrit :

$$LR = D_0 - D_M$$

Cette statistique suit une loi de χ^2 à $ddl = (K - 1) \times p$. La région critique du test au risque α correspond aux grandes valeurs de la statistique de test, c'est-à-dire :

$$LR \geq \chi^2_{1-\alpha}(ddl)$$

Nous pouvons aussi décider via la **p-value**. Si elle est plus petite que α , le modèle est globalement significatif.

```
# Déviance du modèle étudié
DM = -2*fullresult.llf
# Deviance du modèle trivial
D0 = -2*fullresult.llnull
# Test du rapport
LR = D0 - DM
#
```

```

import numpy as np
import scipy.stats as stats
K = len(np.unique(abalone.Sex))
p = len(fullresult.params)-1
# degré de liberté
ddl = (K-1)*p
# p-value associée
pvalue = 1.0-stats.chi.cdf(LR,ddl)
print(f'Déviance du modèle étudié : %.2f' % (DM))
print(f'Déviance du modèle trivial : %.2f' % (DO))
print(f'Statistique du rapport de vraisemblance : %.2f' % (LR))
print(f'p-value associée : %.4f' % (pvalue))

Déviance du modèle étudié : 7139.24
Déviance du modèle trivial : 9157.82
Statistique du rapport de vraisemblance : 2018.57
p-value associée : 0.0000

```

La p-value est nulle. Donc le modèle est globalement très significatif.

16.2 Évaluation des classifieurs

16.2.1 Matrice de confusion et taux d'erreur

La matrice de confusion confronte les valeurs observées de Y sur l'échantillon et les valeurs prédictes par le modèle. Nous avons le tableau de contingence suivant :

$\begin{array}{c} \hat{Y} \\ \diagdown \\ Y \end{array}$	$\hat{Y} = 1$	\dots	$\hat{Y} = k$	\dots	$\hat{Y} = K$	Total
$Y = 1$	n_{11}	\dots	n_{1k}	\dots	n_{1K}	$n_{1\cdot} = n_1$
\dots						\dots
$Y = k$	n_{k1}	\dots	n_{kk}	\dots	n_{kK}	$n_{k\cdot} = n_k$
\dots						\dots
$Y = K$	n_{K1}	\dots	n_{Kk}	\dots	n_{KK}	$n_{K\cdot} = n_K$
Total	$n_{\cdot 1}$		$n_{\cdot k}$		$n_{\cdot K}$	n

TABLE 16.1 – Matrice de confusion pour un apprentissage multi-classes ($K > 2$)

L'effectif de la case (k,j) est égal au nombre d'individus appartenant à la catégorie $Y = k$ qui ont été affectés à $Y = j$, c'est-à-dire :

$$n_{kj} = \text{card} \left\{ \omega \in \Omega, Y(\omega) = k \text{ et } \hat{Y}(\omega) = j \right\}$$

Application :

On calcule les scores prédits par le modèle :

```

# fitted values
import numpy as np
score = pd.DataFrame(fullresult.predict(), columns = ['F', 'I', 'M'],

```

```

        index = abalone.index)
print(score)

      F          I          M
0    0.288123  0.234859  0.477019
1    0.075959  0.753620  0.170421
2    0.311295  0.374541  0.314163
3    0.287757  0.307598  0.404646
4    0.076392  0.736766  0.186841
...
4172  0.498735  0.064642  0.436623
4173  0.366880  0.167148  0.465973
4174  0.505440  0.024515  0.470045
4175  0.432085  0.084507  0.483408
4176  0.366939  0.001493  0.631568

[4177 rows x 3 columns]

```

On vérifie que la somme des probabilités pour chaque individu est bien égale à 1.

```

# Vérification
print(score.sum(axis=1))

0      1.0
1      1.0
2      1.0
3      1.0
4      1.0
...
4172   1.0
4173   1.0
4174   1.0
4175   1.0
4176   1.0
Length: 4177, dtype: float64

```

La règle de prédiction consiste à affecter un individu à la classe pour laquelle il possède la plus grande probabilité. Si on note $\hat{\pi}_i^k$ ($k = F, I, M$) la probabilité d'appartenir à la classe k pour l'individu i . Pour le premier individu, on a les probabilités d'appartenance aux classes suivantes : $\hat{\pi}_1^F = 0.29$, $\hat{\pi}_1^I = 0.23$ et $\hat{\pi}_1^M = 0.48$. On constate que $\hat{\pi}_1^M > \hat{\pi}_1^F > \hat{\pi}_1^I$. On affecte l'individu 1 à la classe M . On fait le même raisonnement pour les autres individus.

```

# Prédiction
pred = []
for i in range(abalone.shape[0]):
    idx = score.loc[i,] [score.loc[i,]==np.max(score.loc[i,])].index[0]
    pred.append(idx)
# 10 premiers observations
print(pred[:10])

['M', 'I', 'I', 'M', 'I', 'I', 'M', 'M', 'I', 'M']

```

On construit la matrice de confusion :

```
# Confusion matrix
confmat = pd.crosstab(abalone.Sex,np.array(pred))
confmat.index = ['F','I','M']
confmat.columns = ['F_hat', 'I_hat', 'M_hat']
confmat.loc[:, 'Total'] = confmat.sum(axis=1)
confmat.loc['Total', :] = confmat.sum(axis=0)
print(confmat)
```

	F_hat	I_hat	M_hat	Total
F	449.0	215.0	643.0	1307.0
I	69.0	1108.0	165.0	1342.0
M	385.0	351.0	792.0	1528.0
Total	903.0	1674.0	1600.0	4177.0

Statsmodels retourne la matrice de confusion calculée sur l'échantillon d'apprentissage

```
# confusion matrix
conf_mat = pd.DataFrame(fullresult.pred_table(),
                        columns = ['F_hat', 'I_hat', 'M_hat'],
                        index = ['F', 'I', 'M']).astype('int')
conf_mat.index.name = "Sex"
conf_mat.loc[:, 'Total'] = conf_mat.sum(axis=1)
conf_mat.loc['Total', :] = conf_mat.sum(axis=0)
print(conf_mat)
```

	F_hat	I_hat	M_hat	Total
Sex				
F	449.0	215.0	643.0	1307.0
I	69.0	1108.0	165.0	1342.0
M	385.0	351.0	792.0	1528.0
Total	903.0	1674.0	1600.0	4177.0

Le taux d'erreur est l'estimation de la probabilité de mal classer, il correspond au rapport entre le nombre total d'observations mal classées et l'effectif total dans le jeu de données :

$$\varepsilon = \frac{\sum_k \sum_{j \neq k} n_{kj}}{n} = 1 - \frac{\sum_k n_{kk}}{n}$$

```
# Taux d'erreur (en resubstitution)
error_rate = 1 - np.trace(fullresult.pred_table()) / abalone.shape[0]
print(f"error rate (en resubstitution) : %.2f" % (error_rate))

error rate (en resubstitution) : 0.44
```

Le taux de succès θ est toujours le complément à 1 du taux d'erreur. Il indique la probabilité de bien classer :

$$\theta = 1 - \varepsilon = \frac{\sum_k n_{kk}}{n}$$

```

# Accuracy
accuracy = 1 - error_rate
print(f'Accuracy : %.2f' % (accuracy))

Accuracy : 0.56

```

16.2.2 Indicateurs synthétiques pour le rappel et la précision

1. Rappel précision par catégorie

Le rappel et la précision sont des indicateurs très populaires car leurs interprétations sont simples à appréhender. Le premier indique la capacité du modèle à retrouver les positifs, le second, la capacité à les prédire (désigner) avec justesse. Nous pouvons les associer aux catégories dans le cadre multi-classes, pour le rappel de $Y = k$:

$$\text{rappel}_k = \frac{n_{kk}}{n_k}$$

et la précision de $Y = k$:

$$\text{précision}_k = \frac{n_{kk}}{n_k}$$

```

# rappel et précision
mat = fullresult.pred_table()
rappel = np.diagonal(mat)/np.sum(mat, axis=1)
precision = np.diagonal(mat)/np.sum(mat, axis=0)
frame = pd.DataFrame(np.transpose([rappel, precision]),
                      columns = ['rappel', 'precision'],
                      index = ['F', 'I', 'M'])

print(frame)

      rappel  precision
F  0.343535    0.497231
I  0.825633    0.661888
M  0.518325    0.495000

```

2. Microaveraging et macroaveraging

La **microaveraging** (micro-moyenne) est une moyenne pondérée où les catégories pèsent selon leur effectif dans le tableau de contingence. On accorde le même poids aux observations. Il est produit directement via la matrice de confusion (Tableau 16.1).

La **macroaveraging** (macro-moyenne) est une moyenne non-pondérée où l'on accorde le même poids aux catégories. Nous pouvons le produire directement via les rappels et précisions obtenues pour les catégories. Lorsque les prévalences des modalités de la variable dépendante sont très différentes, ces deux ratios peuvent diverger assez fortement. A nous de choisir le bon selon les objectifs de l'étude.

La micro-moyenne met l'accent sur les modalités fréquentes, la macro-moyenne accorde plus d'importance à celles qui sont peu fréquentes.

	Microaveraging	Macroaveraging
Rappel	$\mu_{\text{rappel}} = \frac{\sum_{k=1}^K n_{kk}}{\sum_{k=1}^K n_{k.}}$	$\rho_{\text{rappel}} = \frac{\sum_{k=1}^K \text{rappel}_k}{K}$
Précision	$\mu_{\text{précision}} = \frac{\sum_{k=1}^K n_{kk}}{\sum_{k=1}^K n_{.k}}$	$\rho_{\text{rappel}} = \frac{\sum_{k=1}^K \text{précision}_k}{K}$

TABLE 16.2 – Microaveraging et macroaveraging

On rappelle que $\sum_{k=1}^K n_{k.} = \sum_{k=1}^K n_{.k} = n$, ce qui implique $\mu_{\text{rappel}} = \mu_{\text{précision}} = \text{taux de succès}$.

```
# Microaveraging et macroaveraging
mu_rappel = mu_precision = np.trace(mat)/abalone.shape[0]
rho_rappel = np.sum(rappel)/len(np.unique(abalone.Sex))
rho_precision = np.sum(precision)/len(np.unique(abalone.Sex))
mat2 = np.array([[mu_rappel,rho_rappel],[mu_precision, rho_precision]])
frame2 = pd.DataFrame(mat2,columns = ["microaveraging","macroaveraging"]),
           index = ['rappel','précision'])
print(frame2)

      microaveraging  macroaveraging
rappel          0.562365        0.562498
précision       0.562365        0.551373
```

3. Taux d'erreur et échantillon non représentatif

Lorsque l'échantillon de test n'est pas représentatif, le taux d'erreur n'est pas transposable à la population. Il ne correspond pas à la probabilité de mauvais classement du modèle. il nous faut le corriger en utilisant les "vraies" prévalences p_k des catégories dans la population. La formulation est simple :

$$\varepsilon = \sum_{k=1}^K p_k \times (1 - \text{rappel}_k)$$

```
# Taux d'erreur corrigé
pk = abalone.Sex.value_counts(normalize=True).values
error_rate2 = sum(pk*(1-rappel))
print(f"error rate modified : {%.2f} % (error_rate2)")

error rate modified : 0.45
```

Remarque :

Lorsque l'échantillon est représentatif, nous pouvons estimer p_k par $\hat{p}_k = \frac{n_k}{n}$, ce qui revient à :

$$\begin{aligned}\varepsilon &= \sum_{k=1}^K \hat{p}_k \times (1 - \text{rappel}_k) = \sum_{k=1}^K \frac{n_k}{n} \times \left(1 - \frac{n_{kk}}{n_k}\right) \\ &= \sum_{k=1}^K \left(\frac{n_k}{n} - \frac{n_{kk}}{n}\right) = \sum_{k=1}^K \frac{n_k}{n} - \sum_{k=1}^K \frac{n_{kk}}{n} \\ &= 1 - \sum_{k=1}^K \frac{n_{kk}}{n}\end{aligned}$$

16.3 Test sur les coefficients de la régression multinomiale

Comme pour la régression binaire, nous disposons de deux outils pour réaliser les tests : la statistique du rapport de vraisemblance et la statistique de Wald.

16.3.1 Estimation de la matrice de variance covariance

La matrice de variance covariance est une pièce essentielle de la statistique inférentielle. Concernant la régression logistique, elle nous permettra de mettre en place les tests de Wald. Nous pourrions en tirer parti également pour la production des intervalles de confiance des coefficients et des prédictions.

La matrice de variance covariance $\hat{\Sigma}$ correspond à l'inverse de la matrice hessienne. Elle est aussi symétrique par blocs. Il faut bien faire attention pour discerner les informations importantes qu'elles comportent : nous avons la variance des coefficients pour chaque équation logit, les covariances entre coefficients de la même équation logit, et les covariances des coefficients relatives à des équations logit différentes. On peut s'y perdre rapidement.

```
# Matrice de variances-covariances des coefficients
cov_mat = fullresult.cov_params()
print(cov_mat)
```

Sex		I					\
		const	Length	Diameter	Height	Whole weight	
Sex	I	0.254927	-0.529581	-0.110802	-0.349865	0.082166	
const	Length	-0.529581	8.201781	-8.534279	-0.540185	-0.091478	
Diameter		-0.110802	-8.534279	12.435372	-1.320351	-0.045916	
Height		-0.349865	-0.540185	-1.320351	11.884649	-0.004658	
Whole weight		0.082166	-0.091478	-0.045916	-0.004658	2.986971	
Shucked weight		0.024581	-0.321563	-0.135486	0.137877	-2.990224	
Viscera weight		0.130280	-0.712939	0.336533	-0.806651	-2.891650	
Shell weight		0.237190	0.274672	-0.996110	-1.128953	-3.370821	
Rings		-0.002348	0.000009	-0.006020	-0.012814	-0.006168	
M	const	0.142859	-0.241806	-0.092133	-0.154807	0.027843	
Length		-0.239450	2.269590	-2.166260	0.081608	-0.016157	

Diameter	-0.083415	-2.167804	3.241323	-0.255114	-0.021635
Height	-0.136465	0.052562	-0.255210	2.251893	-0.002978
Whole weight	0.025628	-0.013053	-0.020723	-0.006058	0.310918
Shucked weight	0.015218	-0.085949	-0.032064	-0.021439	-0.309348
Viscera weight	0.048628	-0.190692	0.050596	-0.143211	-0.319135
Shell weight	0.071130	-0.014944	-0.175714	-0.205617	-0.344593
Rings	-0.000949	0.000515	-0.000832	-0.001763	-0.001601

Sex

Shucked weight Viscera weight Shell weight Rings

Sex

I	const	0.024581	0.130280	0.237190	-0.002348
	Length	-0.321563	-0.712939	0.274672	0.000009
	Diameter	-0.135486	0.336533	-0.996110	-0.006020
	Height	0.137877	-0.806651	-1.128953	-0.012814
	Whole weight	-2.990224	-2.891650	-3.370821	-0.006168
	Shucked weight	3.887073	2.048690	3.065174	0.014356
	Viscera weight	2.048690	7.036323	2.343530	0.006830
	Shell weight	3.065174	2.343530	6.279126	-0.005693
	Rings	0.014356	0.006830	-0.005693	0.000688
M	const	0.018433	0.052971	0.088041	-0.000982
	Length	-0.079341	-0.208569	-0.041583	0.000676
	Diameter	-0.038849	0.051553	-0.187573	-0.000935
	Height	-0.025194	-0.139549	-0.217858	-0.001852
	Whole weight	-0.309479	-0.319263	-0.342260	-0.001565
	Shucked weight	0.455620	0.189888	0.315289	0.003021
	Viscera weight	0.180440	0.923311	0.285680	0.001483
	Shell weight	0.313389	0.278724	0.724370	-0.001110
	Rings	0.002982	0.001667	-0.001065	0.000143

Sex

M
const Length Diameter Height Whole weight

Sex

I	const	0.142859	-0.239450	-0.083415	-0.136465	0.025628
	Length	-0.241806	2.269590	-2.167804	0.052562	-0.013053
	Diameter	-0.092133	-2.166260	3.241323	-0.255210	-0.020723
	Height	-0.154807	0.081608	-0.255114	2.251893	-0.006058
	Whole weight	0.027843	-0.016157	-0.021635	-0.002978	0.310918
	Shucked weight	0.018433	-0.079341	-0.038849	-0.025194	-0.309479
	Viscera weight	0.052971	-0.208569	0.051553	-0.139549	-0.319263
	Shell weight	0.088041	-0.041583	-0.187573	-0.217858	-0.342260
	Rings	-0.000982	0.000676	-0.000935	-0.001852	-0.001565
M	const	0.186524	-0.335189	-0.086796	-0.182704	0.033987
	Length	-0.335189	3.815551	-3.837523	-0.003274	-0.008956
	Diameter	-0.086796	-3.837523	5.547099	-0.487781	-0.024585
	Height	-0.182704	-0.003274	-0.487781	3.857651	0.002056
	Whole weight	0.033987	-0.008956	-0.024585	0.002056	0.542947
	Shucked weight	0.020031	-0.135117	-0.048294	-0.030072	-0.536469
	Viscera weight	0.069246	-0.299511	0.093446	-0.227325	-0.573334

Shell weight	0.099078	0.011218	-0.279837	-0.317059	-0.617266
Rings	-0.001290	0.000599	-0.002092	-0.003584	-0.002508
Sex					
Sex		Shucked weight	Viscera weight	Shell weight	Rings
I	const	0.015218	0.048628	0.071130	-0.000949
	Length	-0.085949	-0.190692	-0.014944	0.000515
	Diameter	-0.032064	0.050596	-0.175714	-0.000832
	Height	-0.021439	-0.143211	-0.205617	-0.001763
	Whole weight	-0.309348	-0.319135	-0.344593	-0.001601
	Shucked weight	0.455620	0.180440	0.313389	0.002982
	Viscera weight	0.189888	0.923311	0.278724	0.001667
	Shell weight	0.315289	0.285680	0.724370	-0.001065
	Rings	0.003021	0.001483	-0.001110	0.000143
M	const	0.020031	0.069246	0.099078	-0.001290
	Length	-0.135117	-0.299511	0.011218	0.000599
	Diameter	-0.048294	0.093446	-0.279837	-0.002092
	Height	-0.030072	-0.227325	-0.317059	-0.003584
	Whole weight	-0.536469	-0.573334	-0.617266	-0.002508
	Shucked weight	0.780791	0.324532	0.542414	0.005140
	Viscera weight	0.324532	1.644989	0.479873	0.002691
	Shell weight	0.542414	0.479873	1.278147	-0.001916
	Rings	0.005140	0.002691	-0.001916	0.000246

C'est une matrice de taille $[(K - 1) \times (p + 1), (K - 1) \times (p + 1)]$, soit 18×18 . Afin d'obtenir les écarts types des coefficients tel que décrit par le modèle, nous devons diviser notre matrice en $K - 1$ matrices de taille $(p + 1) \times (p + 1)$. Nous avons la matrice 1 relative au logit $\ln\left(\frac{\mathbb{P}(Y = I/X)}{\mathbb{P}(Y = F/X)}\right)$

```
# Bloc 1
cov_mat1= cov_mat.iloc[:9,:9]
print(cov_mat1)

Sex                                I
                                         const      Length   Diameter   Height  Whole weight
Sex
I  const      0.254927 -0.529581 -0.110802 -0.349865  0.082166
  Length     -0.529581  8.201781 -8.534279 -0.540185 -0.091478
  Diameter    -0.110802 -8.534279 12.435372 -1.320351 -0.045916
  Height      -0.349865 -0.540185 -1.320351 11.884649 -0.004658
  Whole weight 0.082166 -0.091478 -0.045916 -0.004658  2.986971
  Shucked weight 0.024581 -0.321563 -0.135486 0.137877 -2.990224
  Viscera weight 0.130280 -0.712939  0.336533 -0.806651 -2.891650
  Shell weight   0.237190  0.274672 -0.996110 -1.128953 -3.370821
  Rings        -0.002348  0.000009 -0.006020 -0.012814 -0.006168

Sex
                                         Shucked weight   Viscera weight   Shell weight   Rings
Sex
```

I	const	0.024581	0.130280	0.237190	-0.002348
	Length	-0.321563	-0.712939	0.274672	0.000009
	Diameter	-0.135486	0.336533	-0.996110	-0.006020
	Height	0.137877	-0.806651	-1.128953	-0.012814
	Whole weight	-2.990224	-2.891650	-3.370821	-0.006168
	Shucked weight	3.887073	2.048690	3.065174	0.014356
	Viscera weight	2.048690	7.036323	2.343530	0.006830
	Shell weight	3.065174	2.343530	6.279126	-0.005693
	Rings	0.014356	0.006830	-0.005693	0.000688

et la matrice 2 relative au logit $\ln \left(\frac{\mathbb{P}(Y = M/X)}{\mathbb{P}(Y = F/X)} \right)$

Bloc 2

Sex		I				
		const	Length	Diameter	Height	Whole weight
Sex						
I	const	0.254927	-0.529581	-0.110802	-0.349865	0.082166
	Length	-0.529581	8.201781	-8.534279	-0.540185	-0.091478
	Diameter	-0.110802	-8.534279	12.435372	-1.320351	-0.045916
	Height	-0.349865	-0.540185	-1.320351	11.884649	-0.004658
	Whole weight	0.082166	-0.091478	-0.045916	-0.004658	2.986971
	Shucked weight	0.024581	-0.321563	-0.135486	0.137877	-2.990224
	Viscera weight	0.130280	-0.712939	0.336533	-0.806651	-2.891650
	Shell weight	0.237190	0.274672	-0.996110	-1.128953	-3.370821
	Rings	-0.002348	0.000009	-0.006020	-0.012814	-0.006168
Sex						
		Shucked weight Viscera weight Shell weight Rings				
Sex						
I	const	0.024581	0.130280	0.237190	-0.002348	
	Length	-0.321563	-0.712939	0.274672	0.000009	
	Diameter	-0.135486	0.336533	-0.996110	-0.006020	
	Height	0.137877	-0.806651	-1.128953	-0.012814	
	Whole weight	-2.990224	-2.891650	-3.370821	-0.006168	
	Shucked weight	3.887073	2.048690	3.065174	0.014356	
	Viscera weight	2.048690	7.036323	2.343530	0.006830	
	Shell weight	3.065174	2.343530	6.279126	-0.005693	
	Rings	0.014356	0.006830	-0.005693	0.000688	

Ensuite on applique la racine carré sur les éléments diagonaux de chacune d'elles.

```
# Standard Errors
std_cov_mat1 = np.sqrt(np.diagonal(cov_mat1))
std_cov_mat2 = np.sqrt(np.diagonal(cov_mat2))
sigma_coef = pd.DataFrame(np.transpose([std_cov_mat1, std_cov_mat2]),
                           index = coef.index,
                           columns = ['I', 'M'])
sigma_coef.index.name = 'Std. Errors'
print(sigma_coef)
```

	I	M
Std. Errors		
const	0.504903	0.431884
Length	2.863875	1.953343
Diameter	3.526382	2.355228
Height	3.447412	1.964090
Whole weight	1.728285	0.736850
Shucked weight	1.971566	0.883624
Viscera weight	2.652607	1.282571
Shell weight	2.505818	1.130552
Rings	0.026230	0.015697

`Statsmodels` renvoie l'écart-type des coefficients estimés grâce à l'outil `bse`

```
# standard errors
std_error_coef = fullresult.bse
std_error_coef.columns = ['I','M']
print(std_error_coef)
```

	I	M
const	0.504903	0.431884
Length	2.863875	1.953343
Diameter	3.526382	2.355228
Height	3.447412	1.964090
Whole weight	1.728285	0.736850
Shucked weight	1.971566	0.883624
Viscera weight	2.652607	1.282571
Shell weight	2.505818	1.130552
Rings	0.026230	0.015697

16.3.2 Significativité d'un coefficient dans un logit

L'hypothèse nulle de ce est s'écrit :

$$H_0 : \beta_p^k = 0$$

Un coefficient dans un des logit est-il significatif? Si la réponse est non, il ne l'est pas alors nous pouvons supprimer la variable associée dans le logit concerné. Cependant, nous ne pouvons rien conclure concernant les autres logit : Nous ne pouvons donc pas exclure la variable de l'étude.

1. Test du rapport de vraisemblance

Pour ce test, il s'agit d'optimiser la vraisemblance en forçant $\beta_p^k = 0$. Nous obtenons le modèle contraint (modèle sous H_0), d'en extraire la déviance D_{H_0} que l'on comparera à celle du modèle complet D_M . La statistique de test s'écrit :

$$LR = D_{H_0} - D_M$$

La statistique LR suit une χ^2 à 1 degré de liberté sous H_0 .

2. Test de Wald

La statistique de Wald est formé par le rapport entre le carré du coefficient et sa variance.

$$W_p^k = \left(\frac{\hat{\beta}_p^k}{\hat{\sigma}_{\hat{\beta}_p^k}} \right)^2$$

La statistique W_p^k suit une χ^2 à 1 degré de liberté sous H_0 .

```
# Wald test
W_lenght_M = (coef.iloc[1,1]/sigma_coef.iloc[1,1])**2
# p-value
import scipy.stats as stats
pvalue = 1 - stats.chi2.cdf(W_lenght_M,1)
print(f'Wald test : %.4f' % (W_lenght_M))
print(f'p-value associé : %.4f' %(pvalue))

Wald test : 0.2704
p-value associé : 0.6031
```

16.3.3 Significativité d'un coefficient dans tous les logit

L'hypothèse nulle du test s'écrit :

$$H_0: \beta_p^k = 0 \quad \forall k$$

Il va plus loin que le précédent. Il cherche à savoir si les coefficients d'une variable explicative sont simultanément nuls dans l'ensemble des logit. Si les données sont compatibles avec H_0 , nous pouvons la retirer du modèle.

1. Test du rapport de vraisemblance

Le principe est toujours le même, nous calculons la déviance du modèle contraint et nous la comparons à celle du modèle complet. La statistique de test est :

$$LR = -2 \times (D_{H_0} - D_M)$$

Cette statistique suit une loi du χ^2 à $K - 1$ degrés de liberté sous H_0 .

Pour notre jeu de données, nous lançons les calculs en excluant la variable `length`.

```
# Importation des classes de calcul
X_new = abalone[abalone.columns[2:]]
model_without_length = sm.MNLogit(endog=abalone.Sex,
                                    exog=sm.add_constant(X_new)).fit()
# Summary
print(model_without_length.summary2())

Optimization terminated successfully.
    Current function value: 0.860906
    Iterations 8
    Results: MNLogit
=====
```

```

Model: MNLogit          Pseudo R-squared: 0.215
Dependent Variable: Sex           AIC:    7224.0110
Date: 2021-11-04 14:32  BIC:    7325.4086
No. Observations: 4177          Log-Likelihood: -3596.0
Df Model: 14                LL-Null: -4578.9
Df Residuals: 4161            LLR p-value: 0.0000
Converged: 1.0000            Scale: 1.0000
No. Iterations: 8.0000
-----
```

Sex = 0	Coef.	Std.Err.	t	P> t	[0.025	0.975]
const	3.8473	0.4781	8.0471	0.0000	2.9102	4.7843
Diameter	5.7964	1.9064	3.0405	0.0024	2.0599	9.5328
Height	-6.6853	3.3780	-1.9791	0.0478	-13.3060	-0.0646
Whole weight	-6.2885	1.6926	-3.7152	0.0002	-9.6060	-2.9710
Shucked weight	6.0332	1.9289	3.1278	0.0018	2.2526	9.8139
Viscera weight	-11.8605	2.5995	-4.5626	0.0000	-16.9555	-6.7656
Shell weight	4.8824	2.4498	1.9930	0.0463	0.0808	9.6840
Rings	-0.1969	0.0260	-7.5723	0.0000	-0.2479	-0.1460
Sex = 1	Coef.	Std.Err.	t	P> t	[0.025	0.975]
const	2.5438	0.4038	6.3002	0.0000	1.7525	3.3352
Diameter	-6.2968	1.3179	-4.7778	0.0000	-8.8799	-3.7137
Height	-3.3305	2.0004	-1.6649	0.0959	-7.2513	0.5904
Whole weight	-0.1122	0.7380	-0.1520	0.8792	-1.5586	1.3343
Shucked weight	3.0283	0.8824	3.4319	0.0006	1.2988	4.7577
Viscera weight	-2.2066	1.2750	-1.7306	0.0835	-4.7056	0.2924
Shell weight	0.5343	1.1326	0.4717	0.6371	-1.6855	2.7540
Rings	0.0058	0.0157	0.3710	0.7107	-0.0250	0.0366

On calcule la statistique de test :

```

# LR test
LR_test = -2*(model_without_length.llf - fullresult.llf)
# Degre de liberté
ddl = model_without_length.df_resid - fullresult.df_resid
# p-value
pvalue = 1.0 - stats.chi2.cdf(LR_test, ddl)
print(f'Likelihood ratio test : %.4f' % (LR_test))
print(f'p-value : %.4f' % (pvalue))
```

Likelihood ratio test : 52.7684
p-value : 0

2. Test de Wald

La statistique de test s'écrit :

$$W_p = \hat{\beta}_p' \hat{\Sigma}_j^{-1} \hat{\beta}_p$$

Sous H_0 , cette statistique suit une loi du χ^2 à $K - 1$ degrés de liberté. $\hat{\beta}_p$ est le vecteur des coefficients à évaluer, de dimension $(K - 1) \times 1$, $\hat{\Sigma}_j$ est leur matrice de variance covariance¹.

Pour notre jeu de données, nous récupérons la sous-partie de la matrice de variance covariance qui nous concerne :

```
# indice des coefficients concernés
index = [1,10]
# Sous-matrice
cov_length_coef = np.zeros(shape=(2,2))
for i in range(len(index)):
    for j in range(len(index)):
        cov_length_coef[i,j] = cov_mat.values[index[i],index[j]]
# Vérification
print(cov_length_coef)

[[8.20178062 2.26959014]
 [2.26959014 3.81555069]]
```

Nous inversons cette sous-matrice :

```
# Inversion de cette matrice
inv_cov_length_coef = np.linalg.inv(cov_length_coef)
print(inv_cov_length_coef)

[[ 0.14594772 -0.08681355]
 [-0.08681355  0.31372436]]
```

Nous récupérons également le sous-vecteur des coefficients estimés.

```
# Coefficients à tester
length_coef = coef.iloc[1,:]
print(length_coef)

I      17.6817
M     -1.0157
Name: Length, dtype: float64
```

Nous calculons la forme quadratique correspondant à la statistique de test ainsi que le p-value associée

```
# Statistique de test (Forme quadratique)
wald_stat = np.dot(length_coef, np.dot(inv_cov_length_coef,length_coef))
print('Statistique de test de Wald : %.4f' % (wald_stat))
# p-value avec ddl=len(index)
pvalue = 1.0 - stats.chi2.cdf(wald_stat, len(index))
print('p-value associé : %.4f' % (pvalue))

Statistique de test de Wald : 49.0713
p-value associé : 0.0000
```

`Statsmodels` propose l'outil `wald_test()` pour réaliser les tests généralisés. L'enjeu réside alors dans la définition de la matrice `r_matrix` permettant de spécifier les coefficients à tester.

1. Tout l'enjeu est de savoir lire correctement la matrice de variance covariance globale $\hat{\Sigma}$ et y « piocher » les valeurs de $\hat{\Sigma}_j$.

16.3.4 Test d'égalité d'un coefficient dans tous les logit

Nous souhaitons savoir si les coefficients d'une variable X_p ont identiques d'un logit à l'autre. L'hypothèse nulle s'écrit :

$$H_0 : \beta_p^1 = \dots = \beta_p^{K-1}$$

Lorsqu'elle est compatible avec les données, cela veut dire que la variable a le même impact dans tous les logit. Il n'est pas question en revanche de la supprimer de la régression si elle est par ailleurs significative : son impact est le même, mais il n'est pas nul.

1. Test de rapport de vraisemblance

Définir le modèle contraint dans les logiciels de statistique n'est pas très facile.

2. Test de Wald - Calcul direct

Nous illustrons cette approche à travers notre exemple. L'hypothèse nulle du test est :

$$H_0 : \beta_1^I = \beta_1^M$$

qui peut écrire

$$H_0 : \delta_1 = \beta_1^I - \beta_1^M = 0$$

La statistique de test est :

$$\hat{\delta}_1 = \hat{\beta}_1^I - \hat{\beta}_1^M$$

Sous H_0 , cette statistique est d'espérance nulle et de variance (estimée) :

$$\hat{V}(\hat{\delta}_1) = \hat{V}(\hat{\beta}_1^I) + \hat{V}(\hat{\beta}_1^M) - 2 \times \text{Cov}(\hat{\beta}_1^I, \hat{\beta}_1^M)$$

Sous H_0 , la statistique de test

$$\frac{\delta_1^2}{\hat{V}(\hat{\delta}_1)}$$

suit une loi de χ^2 à 1 degré de liberté.

```
# Différence des coefficients
delta_length = coef.iloc[1,0]-coef.iloc[1,1]
# Variance
var_delta_length = np.sum(np.diagonal(cov_length_coef))-2*cov_length_coef[1,0]
# Statistique de Wald
wald_test = delta_length**2/var_delta_length
# p-value
pvalue = 1.0 - stats.chi2.cdf(wald_test,1)
print(f'Wald test : %.4f' % (wald_test))
print(f'p-value : %.4f' % (pvalue))

Wald test : 46.7486
p-value : 0.0000
```

3. Test de Wald - Calcul générique

Lorsque le nombre d'équations logit est supérieur à 2, l'affaire devient plus compliquée. Il paraît plus judicieux de passer par l'écriture générique des tests en écrivant correctement la matrice \mathbf{R} . Elle est de format m lignes et $l = (K - 1) \times (p + 1)$ colonnes. m ($m \leq p + 1$) est le nombre de variables (y compris la constante) à tester. Si on pose $\hat{\alpha} = R\hat{\beta}$, la statistique de test s'écrit :

$$W_{(R)} = \hat{\alpha}' [R\hat{\Sigma}R]^{-1} \hat{\alpha}$$

Cette statistique suit une χ^2 à m degrés de liberté.

Pour notre exemple, grâce à la librairie [Numpy](#), nous créons une matrice nulle de format (1,18). Ensuite nous remplaçons les valeurs des indices 1 et 10^2 par 1 et -1 respectivement.

```
# Matrice R
R = np.zeros(shape=(1,18))
R[:,1]=1
R[:,10]=-1
print(R)

[[ 0.  1.  0.  0.  0.  0.  0.  0.  0. -1.  0.  0.  0.  0.  0.  0.]]
```

On peut à présent effectuer les calculs de la statistique du test de Wald.

```
# coefficient beta_hat
beta = np.append(coef.iloc[:,0],coef.iloc[:,1], axis=0)
# alpha
alpha = np.dot(R,beta)
#
V = np.dot(R,np.dot(cov_mat,R.T))
# Wald_test
wald_test = alpha*np.linalg.inv(V)*alpha
print(f'Wald test : %.4f' % (wald_test))

Wald test : 46.7486
```

Nous retrouvons exactement la même valeur qu'avec l'approche directe.

16.4 Modèle multinomial et régresseurs catégorielles

Cette section sera illustrée à travers un exemple.

16.4.1 Problématique : Données et modèle

On suppose que nos données sont les suivantes :

- Y : variable à expliquer avec plusieurs catégories (1,2,3)
- X : une variable catégorielle à trois modalités (A,B,C)

2. Rappel : Python débute le comptage à partir de 0.

- Z : une variable catégorielle à deux modalités (E,F)

On veut construire le modèle suivant :

$$\ln \left(\frac{\mathbb{P}(Y = k)}{\mathbb{P}(Y = \text{ref})} \right) = \beta_0^k + \beta_A^k 1_{Z=A} + \beta_B^k 1_{Z=B} + \beta_C^k 1_{Z=C} + \beta_E^k 1_{X=E} + \beta_F^k 1_{X=F}$$

et une contrainte sur les coefficients β_A^k , β_B^k et β_C^k et sur les coefficients β_E^k et β_F^k pour que le modèle soit identifiable, soit $\forall k$:

$$\begin{cases} \beta_A^k + \beta_B^k + \beta_C^k = 0 \\ \beta_E^k + \beta_F^k = 0 \end{cases}$$

Notons que la base de données est en général sous la forme :

individu	X	Z	Y
ind 1	E	A	1
ind 2	F	B	2
ind 3	F	C	1
ind 4	E	C	3
...			

16.4.2 Application

On a posé à 195 étudiants la question : si vous trouvez un portefeuille dans la rue contenant de l'argent et des papiers :

- vous gardez tout (réponse 1) ;
- vous gardez l'argent et rendez le portefeuille (réponse 2) ;
- vous rendez tout (réponse 3).

On construit alors la variable **WALLET** telle que :

- **WALLET**=1 si l'étudiant répond 1 ;
- **WALLET**=2 si l'étudiant répond 2 ;
- **WALLET**=3 si l'étudiant répond 3 ;

Ppour chaque étudiant, on note :

- Le sexe (variable **MALE**=1 si homme, 0 si femme) ;
- La nature des études suivies (variable **BUSINESS**=1 pour les écoles de commerce, 0 pour les autres écoles) ;
- L'existence de punitions passées (variable **PUNISH**=1 si puni seulement à l'école primaire, 2 si puni seulement à l'école primaire et secondaire et 3 si puni seulement à l'école primaire, secondaire et supérieur) ;
- L'explication ou pas par les parents des punitions reçues dans l'enfance (variable **EXPLAIN**=1 si les parents expliquaient, 0 sinon).

On cherche à expliquer la variable `WALLET` par les autres variables.

```
# load dataset
portefeuille = pd.read_sas('portefeuille.sas7bdat')
print(portefeuille.head())

   wallet  male  business  punish  explain
0      2.0    0.0       0.0     2.0      0.0
1      2.0    0.0       0.0     2.0      1.0
2      3.0    0.0       0.0     1.0      1.0
3      3.0    0.0       0.0     2.0      0.0
4      1.0    1.0       0.0     1.0      1.0
```

On voit nos données possèdent des virgules, ce qui traduit qu'ils sont de type `float`.

```
# structure
print(portefeuille.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 5 columns):
 Column    Non-Null Count  Dtype  
---      
0   wallet    195 non-null   float64
1   male      195 non-null   float64
2   business  195 non-null   float64
3   punish    195 non-null   float64
4   explain   195 non-null   float64
dtypes: float64(5)
memory usage: 7.7 KB
None
```

On effectue une double conversion de données : d'abord sous forme de `int` afin d'annuler l'effet `float` et ensuite sous forme de variable catégorielle.

```
# conversion
portefeuille = portefeuille.astype('int').astype('category')
print(portefeuille.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 5 columns):
 Column    Non-Null Count  Dtype  
---      
0   wallet    195 non-null   category
1   male      195 non-null   category
2   business  195 non-null   category
3   punish    195 non-null   category
4   explain   195 non-null   category
dtypes: category(5)
memory usage: 1.7 KB
None
```

On calcule quelques statistiques sur ces données.

```
# Statistiques
print(portefeuille.describe(include='all'))
```

	wallet	male	business	punish	explain
count	195	195	195	195	195
unique	3	2	2	3	2
top	3	1	0	1	1
freq	121	98	150	138	137

On fait quelques croisements de nos variables explicatives avec la variable cible.

Croisement wallet vs. male

```
# croisement wallet vs male
crosstab1 = pd.crosstab(portefeuille.wallet, portefeuille.male)
print(crosstab1)
```

male	0	1
wallet		
1	8	16
2	16	34
3	73	48

Croisement wallet vs. business

```
# croisement wallet vs business
crosstab2 = pd.crosstab(portefeuille.wallet, portefeuille.business)
print(crosstab2)
```

business	0	1
wallet		
1	13	11
2	36	14
3	101	20

Croisement wallet vs. punish

```
# croisement wallet vs punish
crosstab3 = pd.crosstab(portefeuille.wallet, portefeuille.punish)
print(crosstab3)
```

punish	1	2	3
wallet			
1	8	7	9
2	35	10	5
3	95	18	8

Croisement wallet vs. explain

```
# croisement wallet vs explain
crosstab4 = pd.crosstab(portefeuille.wallet, portefeuille.explain)
print(crosstab4)

explain    0    1
wallet
1         15   9
2         18  32
3         25  96
```

Le modèle polytomique multinomial permettant d'expliquer la variable **WALLET** par les autres variables (en prenant comme modalité de référence $Y = 1$) :

$$\ln \left(\frac{p_{\beta}^k(x)}{p_{\beta}^1(x)} \right) = \beta_0^k + \beta_1^k 1_{\{x_1=1\}} + \beta_2^k 1_{\{x_2=1\}} + \beta_3^k 1_{\{x_3=2\}} + \beta_4^k 1_{\{x_3=3\}} + \beta_5^k 1_{\{x_4=1\}} \quad \text{avec } k = 2, 3$$

On entraîne notre modèle :

```
# Importation de la classe de calcul
import statsmodels.formula.api as smf

portefeuille.wallet = portefeuille.wallet.astype('int')
formula = 'wallet~male+business+punish+explain'
cat_model = smf.mnlogit(formula=formula, data = portefeuille).fit()
print(cat_model.summary())

Optimization terminated successfully.
      Current function value: 0.773232
      Iterations 7
      MNLogit Regression Results
=====
Dep. Variable:           wallet      No. Observations:             195
Model:                 MNLogit      Df Residuals:                  183
Method:                MLE        Df Model:                      10
Date: Fri, 05 Nov 2021   Pseudo R-squ.:            0.1436
Time: 08:18:36          Log-Likelihood:          -150.78
converged:                    True        LL-Null:                  -176.07
Covariance Type:    nonrobust     LLR p-value:       2.088e-07
=====
              coef      std err      z      P>|z|      [0.025      0.975]
-----  

Intercept      1.2994      0.663     1.961      0.050      0.001      2.598
male[T.1]      -0.0956      0.584    -0.164      0.870     -1.240      1.049
business[T.1]   -0.7635      0.562    -1.358      0.174     -1.865      0.338
punish[T.2]     -0.8959      0.656    -1.366      0.172     -2.182      0.390
punish[T.3]     -1.7880      0.713    -2.506      0.012     -3.186     -0.390
explain[T.1]     0.7957      0.565     1.409      0.159     -0.311      1.902
```

wallet=3	coef	std err	z	P> z	[0.025	0.975]
Intercept	2.4062	0.624	3.854	0.000	1.183	3.630
male[T.1]	-1.2672	0.554	-2.287	0.022	-2.353	-0.181
business[T.1]	-1.1791	0.549	-2.149	0.032	-2.254	-0.104
punish[T.2]	-1.1451	0.633	-1.809	0.070	-2.386	0.096
punish[T.3]	-2.1412	0.681	-3.144	0.002	-3.476	-0.806
explain[T.1]	1.5935	0.549	2.902	0.004	0.517	2.670

On affiche les coefficients d'estimation des différents logit :

```
# coefficients de régression
cat_coef = cat_model.params
cat_coef.columns = ['2','3']
print(cat_coef)

          2      3
Intercept  1.299392  2.406210
male[T.1]   -0.095584 -1.267203
business[T.1] -0.763541 -1.179112
punish[T.2]   -0.895925 -1.145095
punish[T.3]   -1.787998 -2.141171
explain[T.1]   0.795714  1.593536
```

Tout comme pour les autres variables, on peut tester la nullité d'un sous ensemble de coefficients à l'aide des statistiques de Wald, du rapport de vraisemblance et du score. Par exemple, on obtient la probabilité critique du test du rapport de vraisemblance pour le test :

$$\begin{cases} H_0 : \beta_1^k = \dots = \beta_5^k = 0 \\ H_1 : \exists (k,l) \in \{2,3\} \times \{1, \dots, 5\} \end{cases}$$

avec les commandes

```
# Modèle null
cat_nullmodel = smf.mnlogit(formula='wallet~1', data = portefeuille).fit()
print(cat_nullmodel.summary())
```

Optimization terminated successfully.

```
    Current function value: 0.902922
    Iterations 5
```

MNLogit Regression Results

Dep. Variable:	wallet	No. Observations:	195
Model:	MNLogit	Df Residuals:	193
Method:	MLE	Df Model:	0
Date:	Fri, 05 Nov 2021	Pseudo R-squ.:	7.814e-11
Time:	23:26:01	Log-Likelihood:	-176.07
converged:	True	LL-Null:	-176.07
Covariance Type:	nonrobust	LLR p-value:	nan

wallet=2	coef	std err	z	P> z	[0.025	0.975]
<hr/>						
Intercept	0.7340	0.248	2.956	0.003	0.247	1.221
<hr/>						
wallet=3	coef	std err	z	P> z	[0.025	0.975]
<hr/>						
Intercept	1.6177	0.223	7.240	0.000	1.180	2.056
<hr/>						

```
# Test du rapport de vraisemblance
statRV = -2*(cat_nullmodel.llf - cat_model.llf)
# degré de liberté
ddl = cat_nullmodel.df_resid - cat_model.df_resid
# p-value
pvalue = 1.0 - stats.chi2.cdf(statRV,ddl)
print(f'Test du rapport de vraisemblance : %.4f' % (statRV))
print(f'pvalue : %.4f' % (pvalue))
```

Test du rapport de vraisemblance : 50.5793
pvalue : 0.0000

On veut calculer l'odds ratio de la variable punish pour les modalités 2 et 3 OR(2,3,Y = 2 vs Y = 3). On calcule d'abord odds(2,Y = 2 vs Y = 3) et odds(3,Y = 2 vs Y = 3)

```
# coefficients
coef = cat_model.params.values
# odd de 2
odd_223 = np.exp(coef[0,0]+coef[3,0]-coef[0,1]-coef[3,1])
# odds de 3
odd_323 = np.exp(coef[0,0]+coef[4,0]-coef[0,1]-coef[4,1])
# odds ratio
OR_23 = odd_223/odd_323
print(f'OR(2,3,Y=2 vs Y=3) : %.2f' % (OR_23))
```

OR(2,3,Y=2 vs Y=3) : 0.90

Remarque sur le modèle polytomique nominal avec régresseurs catégoriels

Certains packages proposent de changer la forme du tableau tel que présenté à la section 16.4.1. Il est alors recommandé d'avoir la base de données sous la forme :

X	Z	frequence(Y=1)	frequence(Y=2)	frequence(Y=3)
E	A	3	2	à
E	B	0	3	15
E	C	1	12	7
F	A	0	0	2
...				

où l'on calcule le nombre de fois que l'on observe $Y = 1$, $Y = 2$ et $Y = 3$ pour chaque combinaison possible de X et Z. Cette nouvelle base contient au maximum autant de lignes que de combinaisons

possibles de X et Z : $2 \times 3 = 6$ lignes maximum pour le jeu de données de la table de la section 16.4.1. Le modèle s'écrit :

$$\ln \left(\frac{p_{\beta}^k(x)}{p_{\beta}^1(x)} \right) = \beta_0^k + \beta_1^k 1_{\{x_1=0\}} + \beta_2^k 1_{\{x_1=1\}} + \beta_3^k 1_{\{x_2=0\}} + \beta_4^k 1_{\{x_2=1\}} + \beta_5^k 1_{\{X_3=1\}} + \beta_6^k 1_{\{X_3=2\}} + \beta_7^k 1_{\{X_3=3\}} + \beta_8^k 1_{\{X_4=0\}} + \beta_9^k 1_{\{X_4=1\}} \quad k = 2,3$$

Et on pose les contraintes d'identifiabilité du modèle :

$$\beta_1^k + \beta_2^k = 0, \quad \beta_3^k + \beta_4^k = 0, \quad \beta_5^k + \beta_6^k + \beta_7^k = 0, \quad \beta_8^k + \beta_9^k = 0 \quad \text{avec } k = 2,3$$

Nous illustrons cela sous R avec le package VGAM. On transforme notre tableau initial.

```
##load dataset
library(haven)
portefeuille <- read_sas("portefeuille.sas7bdat")
# transformation
library(tidyverse)
dat.freq1 <- portefeuille %>%
  group_by(male, business, punish, explain) %>%
  summarize(freq.wallet.1 = sum(wallet == 1),
            freq.wallet.2 = sum(wallet == 2),
            freq.wallet.3 = sum(wallet == 3))
dat.freq <- dat.freq1 %>%
  ungroup(male, business, punish, explain) %>%
  mutate(male = as.factor(male),
         business = as.factor(business),
         punish = as.factor(punish),
         explain = as.factor(explain))
# Affichage
print(dat.freq)

# A tibble: 23 x 7
   male  business  punish  explain freq.wallet.1 freq.wallet.2 freq.wallet.3
   <fct> <fct>    <fct>    <fct>      <int>       <int>       <int>
 1 0     0          1          0           1           3           8
 2 0     0          1          1           0           5          45
 3 0     0          2          0           0           2           5
 4 0     0          2          1           0           2           5
 5 0     0          3          0           3           1           1
 6 0     0          3          1           0           0           3
 7 0     1          1          0           0           0           1
 8 0     1          1          1           1           2           2
 9 0     1          2          0           1           0           0
10 0    1          2          1           0           1           2
# ... with 13 more rows
```

On entraîne le modèle :

```

### Entraînement du modèle
library(VGAM)
model <- vglm(cbind(dat.freq$freq.wallet.1,dat.freq$freq.wallet.2,
                     dat.freq$freq.wallet.3) ~ male + business +
                     punish + explain, family = multinomial, data = dat.freq)
# Affichage
print(summary(model))

Call:
vglm(formula = cbind(dat.freq$freq.wallet.1, dat.freq$freq.wallet.2,
                     dat.freq$freq.wallet.3) ~ male + business + punish + explain,
      family = multinomial, data = dat.freq)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept):1 -2.4062    0.6243 -3.854 0.000116 ***
(Intercept):2 -1.1068    0.4282 -2.585 0.009750 **
male1:1        1.2672    0.5542  2.287 0.022216 *
male1:2        1.1716    0.3717  3.152 0.001623 **
business1:1    1.1791    0.5486  2.149 0.031623 *
business1:2    0.4156    0.4234  0.981 0.326389
punish2:1       1.1451    0.6330  1.809 0.070473 .
punish2:2       0.2492    0.4823  0.517 0.605448
punish3:1       2.1412    0.6811  3.144 0.001669 **
punish3:2       0.3532    0.6398  0.552 0.580964
explain1:1     -1.5935    0.5490 -2.902 0.003703 **
explain1:2     -0.7978    0.4060 -1.965 0.049421 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Names of linear predictors: log(mu[,1]/mu[,3]), log(mu[,2]/mu[,3])

Residual deviance: 31.9197 on 34 degrees of freedom

Log-likelihood: -46.3233 on 34 degrees of freedom

Number of Fisher scoring iterations: 4

No Hauck-Donner effect found in any of the estimates

Reference group is level 3 of the response

```

La fonction `vglm` de ce package prend la classe $Y = 3$ comme référence.

CHAPITRE 17

RÉGRESSION LOGISTIQUE POLYTOMIQUE ORDINALE

Sommaire

17.1 Le modèle à catégories adjacentes	399
17.2 Le modèle à odds proportionnels	400
17.3 Exemple d'application	402
17.4 Modèle Log-linéaire : Régression de Poisson	408

Dans ce chapitre, on suppose que les modalités de Y sont hiérarchisées (i.e ordonnées). Dans ce cas, il y a une notion intuitive entre ces modalités. En particulier, les questions qui se posent naturellement sont donc plus du type $Y = k$ versus $Y = k + 1$ ou $Y = k - 1$, ou $Y \leq k$ versus $Y > k$ que de comparer toutes les modalités à une modalité de référence. Il existe beaucoup de modèle qui répondent à ce type de problématique, on en voit deux ici.

17.1 Le modèle à catégories adjacentes

Dans l'écriture du modèle de cette partie, on a toujours écrit implicitement (et génériquement) $X = (1, X_1, \dots, X_p)$ pour le vecteur des régresseurs, et $\beta = (\beta_0, \beta_1, \dots, \beta_p)$ pour le vecteur des coefficients à estimer. Dans le même cadre, on notera génériquement $X^* = (X_1, \dots, X_p)$ et $\beta^* = (\beta_1, \dots, \beta_p)$ (on a donc exclu le régresseur constant).

En reprenant les notations du modèle nominal, le modèle à catégories adjacentes cherche à expliquer le passage d'une catégorie à la catégorie suivante selon une équation du type :

$$\frac{\mathbb{P}(Y = k+1|X)}{\mathbb{P}(Y = k|X)} = e^{\beta_0^{(k+1)} + \beta^* X^*} \quad 0 \leq k \leq K-1$$

où (ce qui est l'essentiel), β^* ne dépend pas de la classe k .

Ce modèle est un cas particulier du modèle nominal ; en particulier, l'hypothèse selon laquelle β^* ne dépend pas de la classe k peut (et doit !) être testée dans ce cadre.

Un des intérêts de ce modèle est qu'on a sensiblement réduit le nombre de paramètres à estimer : il y en a désormais $K + p$. Par ailleurs, il peut aussi constituer un outil de backtesting des classes en Y (amenant à aménager ou concaténer ces classes si c'est possible pour pouvoir valablement utiliser ce modèle).

17.2 Le modèle à odds proportionnels

17.2.1 Écriture du modèle

C'est le modèle le plus utilisé. Le principe du modèle à odds proportionnels est de construire un modèle pseudo-logique où la sortie d'intérêt serait l'évènement $Y \leq k$. Mathématiquement, un tel modèle s'écrit :

$$\frac{\mathbb{P}(Y \leq k|X)}{\mathbb{P}(Y > k|X)} = \exp(\beta_0^k + \beta^* X^*) \quad 0 \leq k \leq K - 1$$

ou encore

$$\text{logit}(\mathbb{P}(Y \leq k|X)) = \beta_0^k + \beta^* X^* \quad 0 \leq k \leq K - 1$$

Remarque :

- Il n'est pas évident de prime abord que ces équations soient compatibles...
- ...mais on peut montrer qu'elles le sont.
- Si $K = 1$ on retrouve un modèle logistique inversé (en permutant les rôles de $Y = 0$ et $Y = 1$).
- Il y a $K + p$ paramètres à estimer dans ce modèle.
- C'est le modèle le plus utilisé...
- ...mais certains logiciels le spécifient différemment (se méfier...)
- Les procédures habituelles (IC, tests...) s'adaptent bien.

17.2.2 Pourquoi « odds proportionnels » ?

L'écriture

$$\frac{\mathbb{P}(Y \leq k|X)}{\mathbb{P}(Y > k|X)} = \exp(\beta_0^k + \beta^* X^*) \quad 0 \leq k \leq K - 1$$

dit que la cote (l'odds) de $Y \leq k$ sachant X vaut $\exp(\beta_0^k + \beta^* X^*)$. Du coup, une expression extrêmement simple de l'odds ratio correspond à :

$$\text{OR}(x_1, x_2, Y \leq X \text{ vs } Y > k) = \exp(\beta^*(x_1^* - x_2^*))$$

Il est évidemment remarquable que cet odds -ratio est indépendant du niveau k choisi, le nom du modèle (odds proportions est tiré de cette propriété).

17.2.3 Égalité des pentes

L'écriture

$$\text{logit}(\mathbb{P}(Y \leq k|X)) = \beta_0^k + \beta^* X^* \quad 0 \leq k \leq K - 1$$

dit que les droites représentant la fonction $x_j \mapsto \text{logit}(\mathbb{P}(Y \leq k|X = x_j))$ sont parallèles (de pente β_j) suivant les modalités k .

Supposons pour simplifier que l'on dispose d'une seule variable explicative X et considérons le modèle suivant :

$$\text{logit}(\mathbb{P}(Y \leq k|X)) = \beta^k + \beta^* X^*$$

On constate que seule la constante diffère selon k , c'est pourquoi on parle d'égalité des pentes. Graphiquement, on a :

```
# Graphique
import numpy as np
from matplotlib import pyplot as plt

plt.plot([0,10],[0,.5], color = 'blue')
plt.plot([0,10],[0.15,0.65],color = 'blue')
plt.plot([0,10],[0.3,0.8],color = 'blue')
plt.annotate('k=1', xy =(10.1,.5),xytext =(10.1,.5))
plt.annotate('k=2', xy =(10.1,.65),xytext =(10.1,.65))
plt.annotate('k=3', xy =(10.1,.8),xytext =(10.1,.8))
plt.ylabel('$p_k(x)$')
plt.xlabel('x')
plt.ylim([0,1])
plt.xlim([0,11])
plt.show()
```

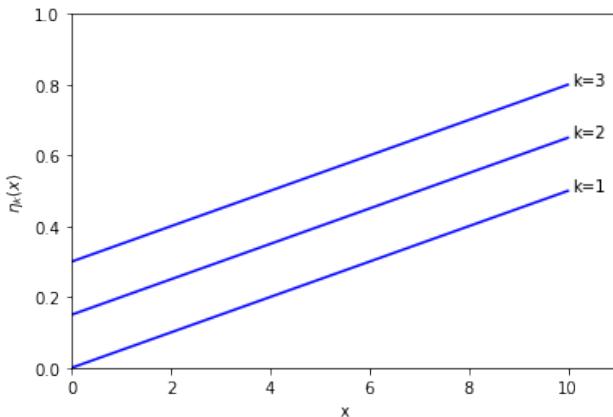


FIGURE 17.1 – Représentation du modèle $p_\beta^k(x) = \beta^k - \beta^* x = \eta_k(x)$

Si on envisage des valeurs différentes pour les paramètres de pentes $\beta_k (k = 1, K - 1)$ alors les droites de la figure 17.1 vont se couper. Il est facile de voir que ceci remet en cause le caractère ordonné des modalités de la variable expliquée. En effet, supposons qu'au delà d'une valeur x_0 , la première droite ($k = 1$) se situe au-dessus de la seconde ($k = 2$), on a alors :

$$\forall x > x_0 \quad \mathbb{P}_\beta(Y \leq 1|X = x) > \mathbb{P}_\beta(Y \leq 2|X = x)$$

Un tel résultat est évidemment gênant : il remet en cause la modélisation retenue, qui distribue les différentes modalités de Y sur un même axe ordonné. Il faut dans ce cas se tourner vers un modèle plus général.

Dans le cas où la variable Y est ordonnée, on définit l'odds relativement à l'évènement $\{Y \leq k\}$ par

$$\text{odds}(x_i) = \frac{\mathbb{P}_\beta(Y \leq k|X = x)}{1 - \mathbb{P}_\beta(Y \leq k|X = x)} = \exp(\beta^k + \beta^*x^*)$$

L'odds ratio relativement à l'évènement $\{Y \leq k\}$ s'écrit donc :

$$\text{OR}(x_1, x_2) = \exp((x_1 - x_2)' \beta^*)$$

Cet odds ratio ne dépend pas de la modalité k , on dit que l'hypothèse des seuils aléatoires se traduit par une « hypothèse de proportionnalité des odds ratio ».

17.2.4 Test d'égalité des pentes

On se pose la question de vérifier si la modélisation en terme de seuil aléatoire est « raisonnable » vis à vis de nos données. Nous avons vu dans la partie précédente que cette hypothèse se traduit par une hypothèse d'égalité des pentes ou de proportionnalité des odds ratio. Un moyen de vérifier si cette hypothèse est raisonnable consiste à tester l'hypothèse d'égalité des pentes. Cette approche consiste tout simplement à comparer le modèle en question (avec égalité des pentes) à un modèle où on lève l'égalité des pentes. Il s'agit de considérer les hypothèses :

$$\begin{cases} H_0 : \beta^0 = \dots = \beta^{K-1} \\ H_1 : \text{pour au moins une valeur } j \in \{1, \dots, p\}, \text{ il existe } k_1 \text{ et } k_2 \text{ tels que } \beta_j^{k_1} \neq \beta_j^{k_2} \end{cases}$$

Cette hypothèse se teste par des variantes des tests classiques (Wald, déviance, score) généralement disponibles dans les logiciels. On dit alors qu'on fait un test d'égalité des pentes. La proc logistique de SAS, par exemple, réalise un test du score dont la statistique suit asymptotiquement, sous H_0 , une loi du χ^2 à $Kp - K - p$ degrés de liberté.

17.3 Exemple d'application

17.3.1 Données de portefeuille

Nous allons utiliser le jeu de données du portefeuille. Ici nous donnons l'exemple de la fonction `vglm` du package `VGAM` qui modélise directement $\mathbb{P}(Y \leq k)$. Il s'agit de la même fonction que celle qui a été utilisée dans la partie sur la régression polytomique avec R (section 16.4.2), mais nous changeons l'argument de l'option `family` en spécifiant `cumulative(parallel=TRUE)`.

Estimation des paramètres

Le modèle polytomique à estimer s'écrit :

$$\begin{aligned}\text{logit}(\hat{\mathbb{P}}_{\beta}(Y \leq k)) &= \beta^k + \alpha_1 \times 1_{\{X_1=1\}} + \alpha_2 \times 1_{\{X_2=1\}} + \alpha_3 \times 1_{\{X_3=2\}} \\ &\quad \alpha_4 \times 1_{\{X_3=2\}} + \alpha_5 \times 1_{\{X_4=1\}} \quad k = 1,2\end{aligned}$$

On entraîne le modèle :

```
# Modèle polytomique ordinaire
require(VGAM)
# model 1
model1 <- vglm(cbind(dat.freq$freq.wallet.1, dat.freq$freq.wallet.2,
                      dat.freq$freq.wallet.3) ~ male+business+punish + explain,
                 family = cumulative(parallel = TRUE), data = dat.freq)

# Affichage
print(summary(model1))

Call:
vglm(formula = cbind(dat.freq$freq.wallet.1, dat.freq$freq.wallet.2,
                     dat.freq$freq.wallet.3) ~ male + business + punish + explain,
      family = cumulative(parallel = TRUE), data = dat.freq)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept):1 -2.5678    0.4169 -6.159 7.33e-10 ***
(Intercept):2 -0.7890    0.3675 -2.147  0.03178 *
male1          1.0598    0.3254  3.256  0.00113 **
business1      0.7389    0.3516  2.102  0.03560 *
punish2         0.6276    0.4005  1.567  0.11706
punish3         1.4031    0.4721  2.972  0.00296 **
explain1        -1.0519   0.3414 -3.081  0.00206 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Names of linear predictors: logitlink(P[Y<=1]), logitlink(P[Y<=2])

Residual deviance: 37.6943 on 39 degrees of freedom

Log-likelihood: -49.2106 on 39 degrees of freedom

Number of Fisher scoring iterations: 5

No Hauck-Donner effect found in any of the estimates

Exponentiated coefficients:
  male1 business1  punish2  punish3  explain1
2.8858456 2.0935787 1.8731872 4.0676512 0.3492847
```

Le modèle estimé peut s'écrire sous la forme :

$$\begin{cases} \text{logit}(\hat{\mathbb{P}}_\beta(Y \leq 1)) = -2.5678 + 1.0598 \times 1_{\{X_1=1\}} + 0.7389 \times 1_{\{X_2=1\}} + 0.6276 \times 1_{\{X_3=2\}} \\ \quad + 1.4031 \times 1_{\{X_3=3\}} - 1.0519 \times 1_{\{X_4=1\}} \\ \text{logit}(\hat{\mathbb{P}}_\beta(Y \leq 2)) = -0.7890 + 1.0598 \times 1_{\{X_1=1\}} + 0.7389 \times 1_{\{X_2=1\}} + 0.6276 \times 1_{\{X_3=2\}} \\ \quad + 1.4031 \times 1_{\{X_3=3\}} - 1.0519 \times 1_{\{X_4=1\}} \end{cases}$$

Les tests

Les tests de l'effet global ie les tests correspondants à :

$$H_0 : \forall k \in \{A, B, C, E, F\}, \alpha_k = 0$$

se calculent grâce à la fonction suivante :

```
## Test de l'effet global
print(lrttest_vglm(model1))

Likelihood ratio test

Model 1: cbind(dat.freq$freq.wallet.1, dat.freq$freq.wallet.2, dat.freq$freq.wallet.3) ~
  male + business + punish + explain
Model 2: cbind(dat.freq$freq.wallet.1, dat.freq$freq.wallet.2, dat.freq$freq.wallet.3) ~
  1
#Df  LogLik Df  Chisq Pr(>Chisq)
1 39 -49.211
2 44 -71.613  5 44.805   1.59e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

qui correspond au test du rapport de vraisemblance.

Pour les tests variable par variable, par exemple pour l'effet global de la variable X, ie :

$$H_0 : \alpha_E = \alpha_F$$

on utilise la fonction suivante :

```
# Test de l'effet global variable par variable
print(anova(model1, type=3))
```

Analysis of Deviance Table (Type III tests: each term added last)

Model: 'cumulative', 'VGAMordinal', 'VGAMcategorical'

Links: 'logitlink'

Response: cbind(dat.freq\$freq.wallet.1, dat.freq\$freq.wallet.2, dat.freq\$freq.wallet.3)

Df Deviance Resid. Df Resid. Dev Pr(>Chi)

```

male      1 10.9265      40    48.621 0.000948 ***
business  1 4.2667      40    41.961 0.038867 *
punish    2 9.1512      41    46.845 0.010300 *
explain   1 9.5168      40    47.211 0.002036 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

```

Remarque :

A noter que contrairement à la partie sur la régression logistique, nous faisons appel à la fonction `anova` avec un « a » minuscule et non pas majuscule. Cela vient du fait que la fonction générique `anova` a été recodé pour le package `VGAM` en mixant les fonctions `anova` et `Anova`.

Test d'égalité des pentes

Pour savoir si l'hypothèse des pentes parallèles (ou odds proportionnels) est valide, pour R, il faut construire un second modèle sans l'hypothèse des pentes parallèles (`family=cumulative(parallel=TRUE)`)

```

## Modèle sans hypothèse de pentes parallèles
model2 <- vglm(cbind(dat.freq$freq.wallet.1, dat.freq$freq.wallet.2,
                      dat.freq$freq.wallet.3) ~ male+business+punish+explain,
                family = cumulative(parallel = FALSE), data = dat.freq)

print(summary(model2))

Call:
vglm(formula = cbind(dat.freq$freq.wallet.1, dat.freq$freq.wallet.2,
                      dat.freq$freq.wallet.3) ~ male + business + punish + explain,
      family = cumulative(parallel = FALSE), data = dat.freq)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept):1 -2.7068    0.5828 -4.644 3.42e-06 ***
(Intercept):2 -0.8393    0.3840 -2.186 0.028846 *
male1:1        0.7393    0.4940  1.497 0.134446
male1:2        1.1569    0.3376  3.427 0.000609 ***
business1:1    1.0766    0.4833  2.228 0.025910 *
business1:2    0.5771    0.3758  1.535 0.124675
punish2:1       1.0562    0.5889  1.794 0.072865 .
punish2:2       0.5641    0.4271  1.321 0.186653
punish3:1       1.9764    0.6116  3.231 0.001232 **
punish3:2       0.9582    0.5191  1.846 0.064910 .
explain1:1     -1.2164    0.4965 -2.450 0.014284 *
explain1:2     -0.9524    0.3631 -2.623 0.008719 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Names of linear predictors: logitlink(P[Y<=1]), logitlink(P[Y<=2])

Residual deviance: 32.607 on 34 degrees of freedom

```

```
Log-likelihood: -46.667 on 34 degrees of freedom
```

```
Number of Fisher scoring iterations: 9
```

```
No Hauck-Donner effect found in any of the estimates
```

```
Exponentiated coefficients:
```

male1:1	male1:2	business1:1	business1:2	punish2:1	punish2:2	punish3:1	pur
2.0945603	3.1802023	2.9345938	1.7808065	2.8755075	1.7578088	7.2168244	2.6
explain1:1	explain1:2						
0.2963045	0.3858083						

et faire un test du rapport de vraisemblance entre les 2 modèles (on peut montrer qu'ils sont emboîtés et que l'on a le droit de faire ce test) :

```
# Test du rapport de vraisemblance
print(anova(model1,model2,type = 1))
```

```
Analysis of Deviance Table
```

Model 1: cbind(dat.freq\$freq.wallet.1, dat.freq\$freq.wallet.2, dat.freq\$freq.wallet.3) ~ male + business + punish + explain	Model 2: cbind(dat.freq\$freq.wallet.1, dat.freq\$freq.wallet.2, dat.freq\$freq.wallet.3) ~ male + business + punish + explain
Resid. Df Resid. Dev Df Deviance Pr(>Chi)	Resid. Df Resid. Dev Df Deviance Pr(>Chi)
1 39 37.694	2 34 32.607 5 5.0873 0.4053

Prédiction

Les prédictions peuvent se faire sur les mêmes données qui ont permis de construire le modèle ou sur des nouvelles données. Le principe des prédictions est très similaire à celui utilisé pour la régression logistique. Seulement comme il y a un nombre fini de combinaisons possibles, tous les individus avec une même combinaison des variables explicatives X et Z auront la même probabilité d'être $Y = 1$, $Y = 2$ ou $Y = 3$. Sous R, on obtient directement les probabilités d'être $Y = 1$, $Y = 3$ et $Y = 3$ pour chaque combinaison en faisant :

```
# Prédiction
pred <- data.frame(predict(model1, type = "response"))
colnames(pred) <- c('Y_hat=1', 'Y_hat=2', 'Y_hat=3')
print(pred)

Y_hat=1  Y_hat=2  Y_hat=3
1  0.07123764 0.2411462 0.68761612
2  0.02609171 0.1108572 0.86305108
3  0.12562696 0.3341212 0.54025184
4  0.04778595 0.1813451 0.77086899
5  0.23780239 0.4110658 0.35113183
6  0.09826668 0.2939986 0.60773472
```

```

7 0.13836261 0.3491093 0.51252812
8 0.05310965 0.1962574 0.75063297
9 0.23124131 0.4092549 0.35950382
10 0.09507523 0.2885120 0.61641273
11 0.39510797 0.3995023 0.20538968
12 0.18123336 0.3860609 0.43270578
13 0.07176543 0.2423287 0.68590592
14 0.29310064 0.4175326 0.28936678
15 0.12650283 0.3352206 0.53827660
16 0.47378709 0.3683061 0.15790679
17 0.23924634 0.4114310 0.34932267
18 0.31666545 0.4162950 0.26703955
19 0.13931312 0.3501452 0.51054169
20 0.46468456 0.3724876 0.16282784
21 0.23265759 0.4096671 0.35767530
22 0.65337937 0.2644158 0.08220479
23 0.39700957 0.3988952 0.20409525

```

17.3.2 Données d'études universitaires

Une étude examine les facteurs qui influencent la décision de postuler ou non aux études supérieures. On demande aux juniors universitaires s'ils sont peu probables (unlikely), assez probables (somewhat likely) ou très susceptibles (very likely) de postuler à des études supérieures. Par conséquent, notre variable de résultat comporte trois catégories. Des données sur le statut d'éducation des parents, que l'établissement de premier cycle soit public ou privé, et la moyenne cumulative actuelle (gpa) sont également collectées. Les chercheurs ont des raisons de croire que les « distances » entre ces trois points ne sont pas égales. Par exemple, la distance entre "unlikely" et "somewhat likely" peut être plus courte que la distance entre "somewhat likely" et "very likely".

```

# load dataset
import pandas as pd

ologit = pd.read_stata("https://stats.idre.ucla.edu/stat/data/ologit.dta")
print(ologit.head())

      apply  pared  public    gpa
0  very likely      0      0  3.26
1 somewhat likely    1      0  3.21
2      unlikely      1      1  3.94
3 somewhat likely    0      0  2.81
4 somewhat likely    0      0  2.53

```

Description des données

Cet ensemble de données hypothétique comporte une variable à trois niveaux appelée `apply`, avec les niveaux « unlikely », « somewhat likely » et « very likely », codés 1, 2 et 3, respectivement, que nous utiliserons comme variable de résultat. Nous avons également trois variables que nous utiliserons comme prédicteurs : `pared`, qui est une variable 0/1 indiquant si au moins un parent a un diplôme d'études supérieures ; `public`, qui est une variable 0/1 où 1 indique que l'établissement de premier cycle est public et 0 privé, et `gpa`, qui est la moyenne pondérée cumulative de l'étudiant.

Régression logistique ordinale : odds proportionnels

Ci-dessous, nous utilisons la commande `OrderedModel` commande du packages `Statsmodels` pour estimer un modèle de régression logistique ordinal¹. Nous ajustons le modèle de régression logistique ordinal suivant.

```
# Ordered model
from statsmodels.miscmodels.ordinal_model import OrderedModel

mod_log = OrderedModel.from_formula('apply~pared+public+gpa', data=ologit,
                                    distr='logit')
res_log = mod_log.fit(method='bfgs', disp=False)
print(res_log.summary())

OrderedModel Results
=====
Dep. Variable: apply Log-Likelihood: -358.51
Model: OrderedModel AIC: 727.0
Method: Maximum Likelihood BIC: 747.0
Date: Sat, 12 Mar 2022
Time: 20:20:19
No. Observations: 400
Df Residuals: 395
Df Model: 5
=====
            coef    std err      z     P>|z|    [0.025    0.975]
-----
pared        1.0476    0.266    3.942    0.000    0.527    1.569
public       -0.0586    0.298   -0.197    0.844   -0.642    0.525
gpa          0.6158    0.261    2.363    0.018    0.105    1.127
unlikely/somewhat likely  2.2035    0.780    2.827    0.005    0.676    3.731
somewhat likely/very likely  0.7398    0.080    9.236    0.000    0.583    0.897
=====
```

Le modèle estimé peut s'écrire sous la forme :

$$\begin{cases} \text{logit}(\hat{P}_\beta(Y \leq 1)) = 2.2035 - 1.0476 \times \text{pared} - (-0.0586) \times \text{public} - 0.6158 \times \text{gpa} \\ \text{logit}(\hat{P}_\beta(Y \leq 2)) = 0.7398 - 1.0476 \times \text{pared} - (-0.0586) \times \text{public} - 0.6158 \times \text{gpa} \end{cases}$$

Pour plus d'informations et d'interprétations, cliquez [ici](#).

17.4 Modèle Log-linéaire : Régression de Poisson

17.4.1 Présentation du modèle

Cadre

Il s'agit d'un modèle où on observe des données de comptage, par exemple des tables de contingence, chaque case correspondant à un croisement x de modalités de variables explicatives. Dans ce cas on s'intéresse au nombre N_x d'observations faites dans la case du tableau correspondant à ce croisement de modalités.. Le modèle Log-linéaire suppose que chaque N_x est de loi de Poisson

1. En cas des difficultés dans l'installation, cliquez [ici](#)

de paramètre $\lambda(x)$.

On peut prendre l'exemple du nombre de ventes d'un produit en ligne, à expliquer par des caractéristiques du produit et/ou des acteurs, ainsi que celui du nombres d'épilepsie entre deux visites médicales, à expliquer par âge, état initial, traitement, etc... La question d'intérêt lié à un tel modèle est souvent l'influence d'un régresseur particulier, toutes choses égales par ailleurs (existence d'une promotion sur le produit, nouveau traitement contre l'épilepsie, etc...).

Modèle

Dans ce cadre, le modèle log-linéaire postule que

$$\lambda(x) = \beta x$$

où β désigne comme d'habitude un vecteur de paramètres à estimer.

La vraisemblance associée aux observations s'écrit :

$$L(\beta) = \prod_x \exp(-\exp(\beta x)) \frac{\exp(N_x \beta x)}{N_x!}$$

et l'EMV se calcule numériquement.

On peut transposer,modulo les aménagements nécessaires, la plupart des idées et méthodes vues dans les autres modèles cadre avec la régression de Poisson. Notons que l'objet de comparaison privilégié dans ce cadre est le rate-ratio qui s'écrit :

$$\text{RR}(x_1, x_2) = \frac{\lambda(x_1)}{\lambda(x_2)} = \exp(\beta(x_1 - x_2))$$

Sur-dispersion

- Dans un modèle Poissonien, le paramètre $\lambda(x)$ est à la fois l'espérance et la variance conditionnelle de N_x .
- Il arrive souvent que les données présentent une sur-dispersion (variance > espérance)
- Corrections possibles :
 - Ajouter un paramètre de sur-dispersion ;
 - Changer de modèle (prendre une loi binomiale négative) ;
 - Modéliser en amont la cause de la sur-dispersion (par exemple un défaut d'indépendance).

17.4.2 Application aux données réelles

Nous reprenons sous [Python](#) les analyses et résultats réalisés sous [R](#) et qui sont consultables [ici](#). Nous disposons d'un jeu données présentant le nombre d'espèces de plantes dans différents quadrats (`Species`) en fonction de la biomasse totale du quadrat (`Biomasse`) et du pH codé selon trois niveaux (faible, moyen, élevé).

```
# load dataset
import pandas as pd
species = pd.read_csv('species.csv', sep=',')
print(species.head())
```

```

      pH   Biomass   Species
0  high  0.469297      30
1  high  1.730870      39
2  high  2.089778      44
3  high  3.925787      35
4  high  4.366793      25

print(species.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90 entries, 0 to 89
Data columns (total 3 columns):
 #   Column   Non-Null Count   Dtype  
--- 
 0   pH        90 non-null     object  
 1   Biomass   90 non-null     float64 
 2   Species   90 non-null     int64   
dtypes: float64(1), int64(1), object(1)
memory usage: 2.2+ KB
None

```

On transforme la variable pH en catégorielle et on réordonne les modalités.

```

# Type category
species['pH'] = species['pH'].astype('category')
# Niveau de pH dans le bon sens
species['pH'] = species['pH'].cat.reorder_categories(['low', 'mid', 'high'])
print(species.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90 entries, 0 to 89
Data columns (total 3 columns):
 #   Column   Non-Null Count   Dtype  
--- 
 0   pH        90 non-null     category 
 1   Biomass   90 non-null     float64  
 2   Species   90 non-null     int64   
dtypes: category(1), float64(1), int64(1)
memory usage: 1.7 KB
None

```

Regardons la distribution de la réponse en fonction des prédicteurs.

```

# Nuage de points
import numpy as np
import matplotlib.pyplot as plt

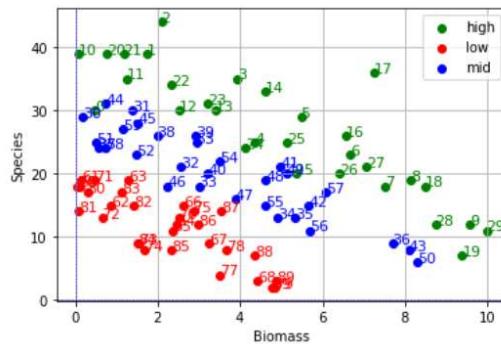
fig, axes = plt.subplots()
axes.set_xlabel("Biomass")
axes.set_ylabel("Species")
cdict = {'high': 'green', 'mid': 'blue', 'low': 'red'}

```

```

for group in np.unique(species.pH):
    idx = np.where(species.pH==group)
    axes.scatter(species.Biomass.values[idx[0]],
                 species.Species.values[idx[0]],
                 label = group, c = cdict[group])
for i in idx[0]:
    axes.text(species.Biomass.values[i],species.Species.values[i],
              s=species.index[i],c = cdict[group], fontsize = 11)
    axes.legend()
axes.grid()
plt.axhline(0, color='blue',linestyle="--", linewidth=0.5)
plt.axvline(0, color='blue',linestyle="--", linewidth=0.5)
plt.show()

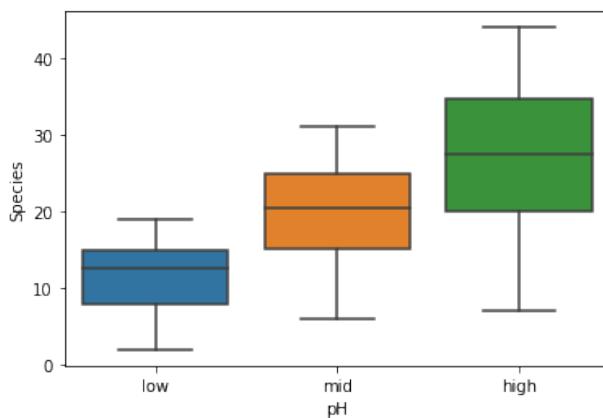
```



```

# Boxplot
import seaborn as sns
ax = sns.boxplot(x=species["pH"],y=species['Species'])
plt.show()

```



Sur ces graphiques, nous remarquons que les conditions associées à un plus grand nombre moyen d'espèces (biomasse faible, pH élevé) ont aussi une plus grande variance. Cela suggère qu'une régression de Poisson pourrait être appropriée.

Comme pour la régression logistique, la régression de Poisson utilise la fonction `glm` de `statsmodels`. Il faut spécifier la famille poisson et (optionnellement) le lien `log`.

```
# Poisson régression
from statsmodels.formula.api import glm
import statsmodels.api as sm
model = glm('Species~Biomass+pH', data=species, family=sm.families.Poisson())
result = model.fit()
print(result.summary())
```

Generalized Linear Model Regression Results

Dep. Variable:	Species	No. Observations:	90
Model:	GLM	Df Residuals:	86
Model Family:	Poisson	Df Model:	3
Link Function:	log	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-259.22
Date:	Sat, 06 Nov 2021	Deviance:	99.242
Time:	21:17:59	Pearson chi2:	95.2
No. Iterations:	4		
Covariance Type:	nonrobust		

	coef	std err	z	P> z	[0.025	0.975]
Intercept	2.7125	0.057	47.479	0.000	2.601	2.825
pH[T.mid]	0.6912	0.068	10.143	0.000	0.558	0.825
pH[T.high]	1.1364	0.067	16.910	0.000	1.005	1.268
Biomass	-0.1276	0.010	-12.579	0.000	-0.147	-0.108

On rajoute à cette sortie certaines informations nécessaires.

```
# Ajout
liste1 = (result.null_deviance,result.df_resid+result.df_model)
liste2 = (result.deviance, result.df_resid)
print(f'Null deviance : %.3f on %.i degrees of freedom' % liste1)
print(f'Residual deviance : %.3f on %.i degrees of freedom' % liste2)
print(f'AIC: %2.f' % (result.aic))
```

```
Null deviance : 452.346 on 89 degrees of freedom
Residual deviance : 99.242 on 86 degrees of freedom
AIC: 526.43
```

Avant d'interpréter les coefficients, vérifions l'ajustement du modèle avec les graphiques de diagnostics

```

# Graphique de diagnostic
import numpy as np
from statsmodels.nonparametric.smoothers_lowess import lowess # pour le lissage

infl = result.get_influence( observed=False)

fig, (ax1,ax2) = plt.subplots(1,2,figsize=(14, 5))
fig.subplots_adjust(wspace=0.5)

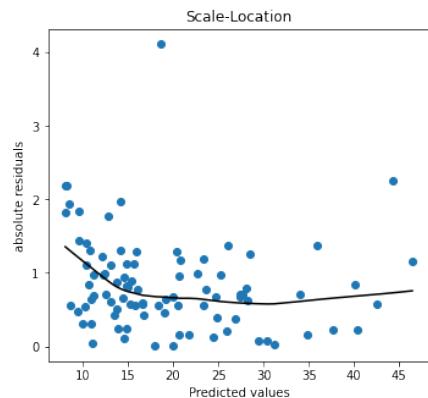
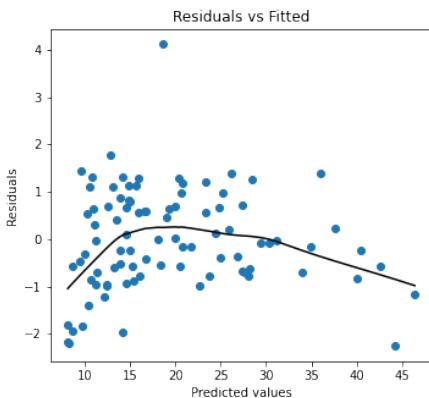
filtered = lowess(infl.resid_studentized,result.fittedvalues)
filtered1 = lowess(np.abs(infl.resid_studentized),result.fittedvalues)

ax1.scatter(result.fittedvalues,infl.resid_studentized)
ax1.plot(filtered[:,0],filtered[:,1],color = 'black')
ax2.scatter(result.fittedvalues,np.abs(infl.resid_studentized))
ax2.plot(filtered1[:,0],filtered1[:,1],color = 'black')

ax1.set_xlabel("Predicted values")
ax1.set_ylabel("Residuals")
ax1.set_title("Residuals vs Fitted")

ax2.set_xlabel("Predicted values")
ax2.set_ylabel("absolute residuals")
ax2.set_title("Scale-Location")
plt.show()

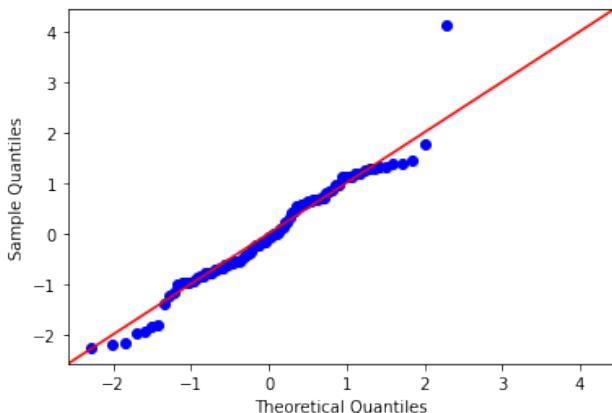
```



```

# qq-plot
import statsmodels.api as sm
fig = sm.qqplot(infl.resid_studentized,line='45')
plt.show()

```



Le premier graphique montre une tendance au niveau des résidus (résidus plus négatifs aux extrêmes et plus positifs au centre). Vu le nombre de points, cette tendance n'est probablement pas due au hasard mais représente un effet systématique qui n'est pas pris en compte dans ce modèle.

Essayons donc un modèle plus complexe où il y a interaction entre l'effet de la biomasse et du pH.

```
# interaction
model_inter = glm('Species~Biomass*pH',data=species,
                   family=sm.families.Poisson()).fit()

# informations
infl2 = model_inter.get_influence( observed=False)

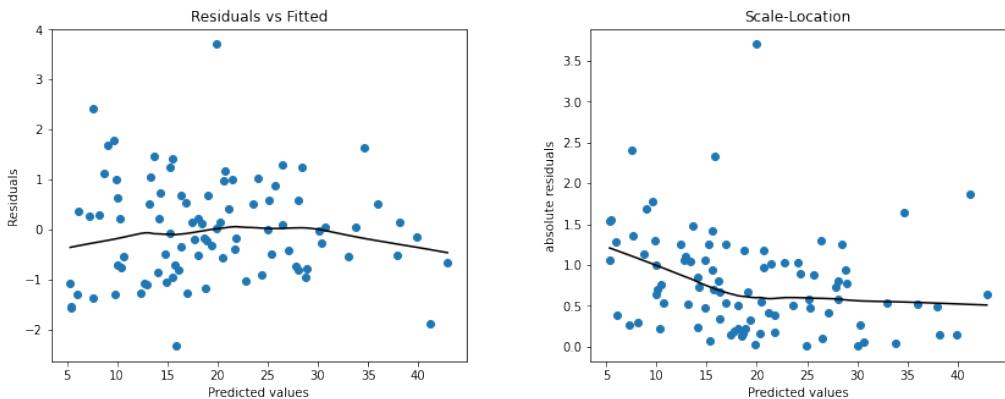
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(14, 5))
fig.subplots_adjust(wspace=0.3)

filtered2 = lowess(infl2.resid_studentized,model_inter.fittedvalues)
filtered3 = lowess(abs(infl2.resid_studentized),model_inter.fittedvalues)

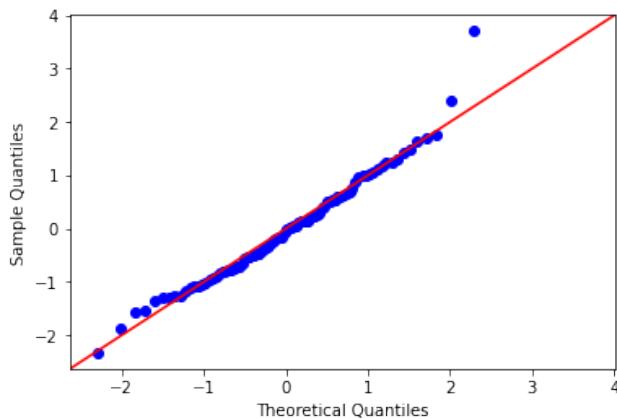
ax1.scatter(model_inter.fittedvalues,infl2.resid_studentized)
ax1.plot(filtered2[:,0],filtered2[:,1],color = 'black')
ax2.scatter(model_inter.fittedvalues,np.abs(infl2.resid_studentized))
ax2.plot(filtered3[:,0],filtered3[:,1],color = 'black')

ax1.set_xlabel("Predicted values")
ax1.set_ylabel("Residuals")
ax1.set_title("Residuals vs Fitted")

ax2.set_xlabel("Predicted values")
ax2.set_ylabel("absolute residuals")
ax2.set_title("Scale-Location")
plt.show()
```



```
# QQ-plot
fig = sm.qqplot(infl2.resid_studentized,line='45')
plt.show()
```



Sans avoir éliminé complètement la tendance, cet ajustement semble bien meilleur.

Note :

Dans ce cas-ci, ces résidus s'approchent de la normalité (d'après le diagramme quantile-quantile) car les valeurs observées de la réponse sont assez élevées (nombre moyen de 20 espèces par quadrat). Nous avions la même situation pour la régression logistique binomiale lorsque n était élevé. Toutefois, nous ne nous attendons pas à ce que le diagramme quantile-quantile montre une droite si λ est petit, même si le modèle de Poisson s'applique parfaitement. Regardons maintenant le sommaire du modèle.

```
# summary
print(model_inter.summary2())
```

```

Results: Generalized linear model
=====
Model:          GLM          AIC:      514.3913
Link Function: log          BIC:     -294.7829
Dependent Variable: Species    Log-Likelihood: -251.20
Date:        2021-11-07 09:45 LL-Null:   -435.77
No. Observations: 90          Deviance:  83.201
Df Model:       5            Pearson chi2: 83.7
Df Residuals:   84           Scale:     1.0000
Method:         IRLS

-----
```

	Coef.	Std.Err.	z	P> z	[0.025	0.975]
Intercept	2.9526	0.0824	35.8330	0.0000	2.7911	3.1140
pH[T.mid]	0.4841	0.1072	4.5147	0.0000	0.2739	0.6943
pH[T.high]	0.8156	0.1028	7.9307	0.0000	0.6140	1.0171
Biomass	-0.2622	0.0380	-6.8928	0.0000	-0.3367	-0.1876
Biomass:pH[T.mid]	0.1231	0.0427	2.8839	0.0039	0.0395	0.2068
Biomass:pH[T.high]	0.1550	0.0400	3.8726	0.0001	0.0766	0.2335

=====

```

# Ajout
liste1 = (model_inter.null_deviance,model_inter.df_resid+model_inter.df_model)
liste2 = (model_inter.deviance, model_inter.df_resid)
print(f'Null deviance : %.3f on %.i degrees of freedom' % liste1)
print(f'Residual deviance : %.3f on %.i degrees of freedom' % liste2)
print(f'AIC: %.2f' % (model_inter.aic))
```

```

Null deviance : 452.346 on 89 degrees of freedom
Residual deviance : 83.201 on 84 degrees of freedom
AIC: 514.39
```

L'AIC de ce modèle est égal à 514 comparé à 526 pour le modèle sans interaction, ce qui confirme le meilleur ajustement. Le pseudo- R^2 est aussi élevé ($1 - 83/452 = 0.82$).

Interprétation des coefficients

L'interprétation des coefficients du modèle est plus complexe avec les interactions, donc commençons avec le modèle sans interaction.

```

print(result.summary())

Generalized Linear Model Regression Results
=====
Dep. Variable:      Species      No. Observations:      90
Model:              GLM          Df Residuals:          86
Model Family:       Poisson     Df Model:             3
Link Function:      log          Scale:               1.0000
Method:             IRLS
Date:        Sat, 06 Nov 2021 Deviance:            99.242
Time:        21:34:09      Pearson chi2:          95.2
```

No. Iterations:		4				
Covariance Type:		nonrobust				
<hr/>						
	coef	std err	z	P> z	[0.025	0.975]
<hr/>						
Intercept	2.7125	0.057	47.479	0.000	2.601	2.825
pH[T.mid]	0.6912	0.068	10.143	0.000	0.558	0.825
pH[T.high]	1.1364	0.067	16.910	0.000	1.005	1.268
Biomass	-0.1276	0.010	-12.579	0.000	-0.147	-0.108
<hr/>						

Il est plus facile de déterminer d'abord l'effet de chaque coefficient du modèle sur le prédicteur linéaire : $\eta = \beta_0 + \sum_{j=1}^p \beta_j x_j$, puis déduire l'effet sur la moyenne de la réponse (λ) à partir de la fonction $\lambda = \exp(\eta)$.

Par exemple, l'ordonnée à l'origine indique que dans le cas où la biomasse est 0 et le pH est bas (niveau de référence), $\eta = 2.71$ et donc $\lambda = \exp(2.71) = 15.0$ espèces. Bien sûr, cette valeur n'est pas réaliste car il ne peut pas y avoir d'espèces sans biomasse.

Pour les autres paramètres :

- Le coefficient **Biomass** indique que η diminue de 0.13 pour chaque augmentation d'une unité de biomasse, si les autres variables (donc le pH) restent constantes. Un changement additif de -0.13 sur l'échelle logarithmique correspond à un changement multiplicatif de $\exp(-0.13) = 0.88$, soit une perte de 12% du nombre moyen d'espèces par unité de biomasse supplémentaire.
- Le coefficient de **pHmid** indique que lorsqu'on passe d'un niveau bas à moyen de pH (pour la même biomasse), η augmente de 0.69. Puisque $\exp(0.69) = 2.0$, le nombre moyen d'espèces à un pH moyen est le double de celui à un pH bas.
- De même, le coefficient de **pHhigh** indique que le nombre moyen d'espèces est multiplié par $\exp(1.14) = 3.13$ si on passe d'un pH bas à un pH élevé pour la même biomasse.

En résumé, dans le cas où il n'y a pas d'interactions, les effets sont additifs sur l'échelle du prédicteur linéaire. Le coefficient β_j de la variable x_j indique qu'une augmentation de 1 unité de x_j , en maintenant les valeurs des autres variables, résulte en une multiplication de la réponse moyenne par $\exp(\beta_j)$.

Regardons maintenant le cas de l'interaction.

```
# modèle avec interaction
print(model_inter.summary2())

Results: Generalized linear model
=====
Model:          GLM           AIC:      514.3913
Link Function: log          BIC:      -294.7829
Dependent Variable: Species   Log-Likelihood: -251.20
```

Date:	2021-11-07 09:45	LL-Null:	-435.77			
No. Observations:	90	Deviance:	83.201			
Df Model:	5	Pearson chi2:	83.7			
Df Residuals:	84	Scale:	1.0000			
Method:	IRLS					

	Coef.	Std.Err.	z	P> z	[0.025	0.975]

Intercept	2.9526	0.0824	35.8330	0.0000	2.7911	3.1140
pH[T.mid]	0.4841	0.1072	4.5147	0.0000	0.2739	0.6943
pH[T.high]	0.8156	0.1028	7.9307	0.0000	0.6140	1.0171
Biomass	-0.2622	0.0380	-6.8928	0.0000	-0.3367	-0.1876
Biomass:pH[T.mid]	0.1231	0.0427	2.8839	0.0039	0.0395	0.2068
Biomass:pH[T.high]	0.1550	0.0400	3.8726	0.0001	0.0766	0.2335
=====						

Dans le modèle avec interaction, l'effet du dH dépend de la biomasse et vice versa. Considérons donc séparément les trois niveau de pH; pour simplier, l'effet "biomasse = 0" sera indiqué même si ce n'est pas un niveau réaliset de la variable.

- A pH bas, nous avons : $\eta = 2.95 - 0.26 \times \text{biomasse}$. Donc, le nombre d'espèce moyen est de $\exp(2.95) = 19$ pour une biomasse de 0 et diminimue de 23% ($\exp(-0.26) = 0.77$) par unité de biomasse.
- A pH moyen, nous avons : $\eta = (2.95 + 0.48) + (-0.26 + 0.12) \times \text{biomasse}$, ou $\eta = 3.43 - 0.14 \times \text{biomasse}$. Donc le nombre d'espèce est de 30.9 pour une hausse de 0 et diminue de 13% par unité de biomasse.
- A pH élevé, nous avons : $\eta = (2.95 + 0.82) + (-0.26 + 0.16) \times \text{biomasse}$, équivalent à un nombre d'espèces moyen de 43.4 pour une biomasse de 0 et une diminution de 10% par unité de biomasse.

Pour une interaction entre une variable numérique et une variable catégorielle, il est plus simple de décrire les effets pour chaque niveau de la variable catégorielle, comme nous venons de faire. Pour une interaction entre deux variables numériques, nous pouvons visualiser les effets à partir des courbes de valeurs prédites, mais il devient compliqué d'interpréter les coefficients individuels.

17.4.3 Sur-dispersion

La non-indépendance des observations individuelles peut causer une sur-dispersion des données par rapport aux suppositions de la distribution de Poisson. Cette sur-dispersion est représentée par un paramètre ϕ qui multiplie la variance attendue : pour une moyenne λ , la variance devient donc $\phi\lambda$.

De façon moins fréquente, il arrive que $\phi < 1$, correspondant à une *sous-dispersion* des observations. Contrairement à la sur-dispersion, où les observations tendent à être regroupées (ex. : quelques quadrats avec de nombreux individus, et plusieurs avec peu ou pas d'individus), la sous-dispersion signifie que les observations sont réparties de façon plus régulière que prévue. Cela pourrait être dû par exemple à la compétition intra-spécifique ou à la territorialité (dans le cas d'animaux) qui mène les individus à s'espacer de façon régulière.

Estimation du paramètre de dispersion

Pour détecter la présence de sur-dispersion ou de sous-dispersion, nous utiliserons la statistique du χ^2 , qui a déjà été vue dans le contexte des tableaux de contingence. Cette statistique est calculée à partir des écarts carrés entre les valeurs observées y et attendues \hat{y} , normalisés par la valeur attendue, pour chacun des n points du jeu de données.

$$\chi^2 = \sum_{i=1}^n \frac{(y_i - \hat{y}_i)^2}{y_i}$$

Si les données suivent la distribution de Poisson, la valeur moyenne du χ^2 est égale au nombre de degrés de liberté résiduels du modèle $df_{\text{resid}} = n - p$, où p est le nombre de paramètres estimés. Pour estimer le paramètre de dispersion ϕ , nous utiliserons l'estimateur :

$$\hat{\phi} = \frac{\chi^2}{df_{\text{resid}}}$$

Pour le modèle avec interactions, nous obtenons presque exactement la valeur théorique de 1 :

```
# Modèle avec interaction
```

```
chisq = np.sum((species.Species-model_inter.fittedvalues)**2/model_inter.fittedvalues)
disp = chisq/model_inter.df_resid
print(disp)
```

```
0.997001437878757
```

Pour le modèle sans interaction, nous détectons une légère surdispersion :

```
# Modèle sans interaction
```

```
chisq = np.sum((species.Species-result.fittedvalues)**2/result.fittedvalues)
disp = chisq/result.df_resid
print(disp)
```

```
1.1067496567714656
```

Statmodels retourne la valeur du χ^2 sous `pearson_chi2`.

```
print(result.pearson_chi2 /result.df_resid)
```

```
1.1067496567714656
```

Pour déterminer si le χ^2 diffère significativement de la valeur attendue selon la distribution de Poisson, nous pouvons calculer la probabilité d'avoir obtenu un χ^2 plus élevé si le modèle est correct.

```
# p-value
```

```
import scipy.stats as stats
pvalue = 1.0 - stats.chi2.cdf(chisq,result.df_resid)
print('p-value associée : %.4f' % (pvalue))
```

```
p-value associée : 0.2336
```

Dans ce cas, la sur-dispersion n'est pas significative, il n'est donc pas nécessaire d'ajuster les résultats du modèle.

Exemple de données sur-dispersées

Le tableau de données `galapagos.csv` donne le nombre d'espèces de plantes (`Species`) et le nombre d'espèces endémiques (`Endemics`) pour différentes îles de l'archipel des Galapagos en fonction de leur superficie (`Area`), de leur altitude moyenne (`Elevation`), de la superficie de l'île la plus près (`Adjacent`), de la distance à l'île la plus près (`Nearest`) et de la distance de Santa Cruz (`Scruz`).

```
# load dataset
glp = pd.read_csv('galapagos.csv')
print(glp.head())

      Name  Species  Endemics    Area  Elevation  Nearest  Scruz  Adjacent
0     Baltra       58        23  25.09       346      0.6      0.6     1.84
1  Bartolome       31        21   1.24       109      0.6     26.3   572.33
2   Caldwell        3         3   0.21       114      2.8     58.7     0.78
3   Champion       25         9   0.10       46      1.9     47.4     0.18
4   Coamano        2         1   0.05       77      1.9      1.9   903.82

print(glp.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Name        30 non-null    object 
 1   Species     30 non-null    int64  
 2   Endemics    30 non-null    int64  
 3   Area        30 non-null    float64
 4   Elevation   30 non-null    int64  
 5   Nearest     30 non-null    float64
 6   Scruz       30 non-null    float64
 7   Adjacent    30 non-null    float64
dtypes: float64(4), int64(3), object(1)
memory usage: 2.0+ KB
None
```

Supposons que nous voulons déterminer comment le nombre d'espèces de plantes endémiques (celles présentent sur une seule île) varie en fonction des prédicteurs `Area`, `Elevation`, `Nearest` et `Adjacent`. Puisque chacun de ces prédicteurs est distribué de façon très asymétrique (voir l'exemple ci-dessous pour la superficie), nous utiliserons le logarithme de chaque prédicteur.

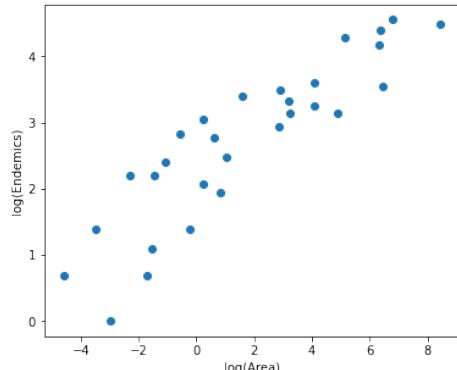
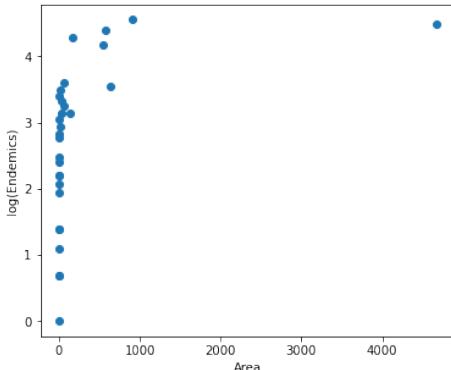
```
# Graphique
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(14, 5))
fig.subplots_adjust(wspace=0.3)
ax1.scatter(glp.Area,np.log(glp.Endemics))
ax2.scatter(np.log(glp.Area),np.log(glp.Endemics))

ax1.set_xlabel('Area')
ax1.set_ylabel('log(Endemics)')
```

```

ax2.set_xlabel('log(Area)')
ax2.set_ylabel('log(Endemics)')
plt.show()

```



Nous ajustons le modèle suivant sous Python. Notez que la réponse est `Endemics` et non `log(Endemics)`, car le lien log est inclus dans le modèle `glm` avec distribution de Poisson.

```

# New data
Data = np.log(glp[['Area','Elevation','Nearest','Adjacent']])
Data.columns = ['log_Area','log_Elevation','log_Nearest','log_Adjacent']
myData = pd.concat([glp.Endemics, Data], axis=1)
# Modèle
mod_glp = glm('Endemics~log_Area+log_Elevation+log_Nearest+log_Adjacent',
               data=myData, family=sm.families.Poisson()).fit()

# summary
infl3 = mod_glp.get_influence(observed=False)

fig, (ax1,ax2) = plt.subplots(1,2,figsize=(14, 5))
fig.subplots_adjust(wspace=0.3)

filtered4 = lowess(infl3.resid_studentized,mod_glp.fittedvalues)
filtered5 = lowess(abs(infl3.resid_studentized),mod_glp.fittedvalues)

ax1.scatter(mod_glp.fittedvalues,infl3.resid_studentized)
ax1.plot(filtered4[:,0],filtered4[:,1],color = 'black')
ax2.scatter(mod_glp.fittedvalues,np.abs(infl3.resid_studentized))
ax2.plot(filtered5[:,0],filtered5[:,1],color = 'black')

ax1.set_xlabel("Predicted values")
ax1.set_ylabel("Residuals")
ax1.set_title("Residuals vs Fitted")

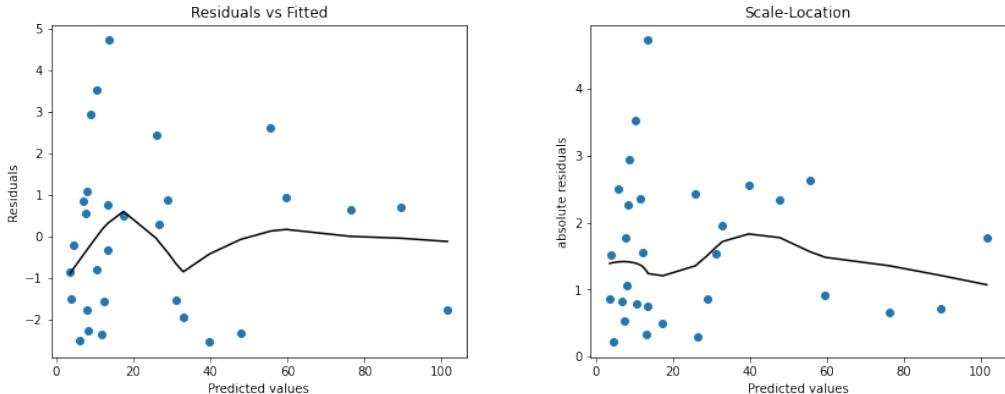
ax2.set_xlabel("Predicted values")

```

```

ax2.set_ylabel("absolute residuals")
ax2.set_title("Scale-Location")
plt.show()

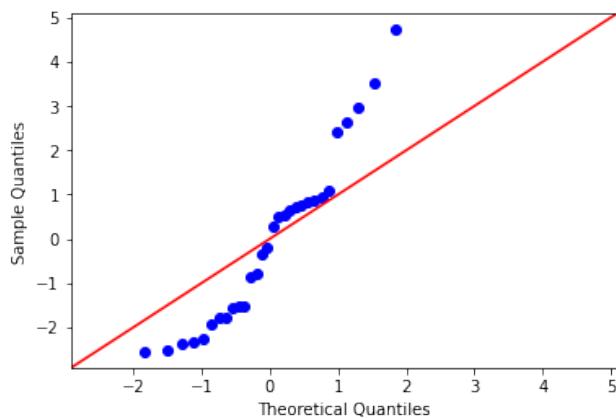
```



```

# QQ-plot
fig = sm.qqplot(infl3.resid_studentized,line='45')
plt.show()

```



Les graphiques de diagnostic indiquent la présence de quelques valeurs extrêmes, bien qu'aucune des distances de Cook soit >1 .

Le modèle indique des effets significatifs pour 3 paramètres : le nombre d'espèces endémiques augmente avec la superficie de l'île ; il diminue en fonction de la distance à l'île la plus près et de la superficie de cette île la plus près.

```
print(mod_glp.summary())
```

Generalized Linear Model Regression Results

Dep. Variable:	Endemics	No. Observations:	30
Model:	GLM	Df Residuals:	25
Model Family:	Poisson	Df Model:	4
Link Function:	log	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-114.31
Date:	Sat, 06 Nov 2021	Deviance:	95.660
Time:	22:38:58	Pearson chi2:	92.5
No. Iterations:	5		
Covariance Type:	nonrobust		

	coef	std err	z	P> z	[0.025	0.975]
Intercept	2.4768	0.458	5.406	0.000	1.579	3.375
log_Area	0.2750	0.029	9.414	0.000	0.218	0.332
log_Elevation	0.0125	0.093	0.135	0.892	-0.169	0.194
log_Nearest	-0.0613	0.021	-2.894	0.004	-0.103	-0.020
log_Adjacent	-0.0458	0.011	-4.162	0.000	-0.067	-0.024

Toutefois, en faisant le test du χ^2 , nous notons que les données sont sur-dispersées.

```
# chisq
chisq = np.sum((myData.Endemics-mod_glp.fittedvalues)**2/mod_glp.fittedvalues)
print('Statistique du test : %.4f' %(chisq))
# pvalue
pvalue = 1.0 - stats.chi2.cdf(chisq,mod_glp.df_resid)
print('p-value associée : %.4f' % (pvalue))

Statistique du test : 92.5110
p-value associée : 0.0000

# indice de dispersion
disp = chisq/mod_glp.df_resid
print(disp)

3.7004406931954805
```

L'estimé du paramètre de dispersion est égal à 3.7. Lorsque $\hat{\phi}$ n'est pas trop élevé (typiquement, on suggère $\hat{\phi} < 4$), les estimés des coefficients de la régression de Poisson demeurent valides, mais il faut multiplier leurs erreurs-types par $\sqrt{\hat{\phi}}$. Autrement dit, la sur-dispersion n'introduit pas de biais, mais augmente l'incertitude sur les valeurs des coefficients. Si la sur-dispersion est très grande, il est préférable d'utiliser un autre modèle.

Pour corriger les erreurs types, on utilise la famille *quasipoisson* qui effectue automatiquement l'estimation du paramètre de dispersion.

```
# Modèle de famille quasipoisson
mod_quasi = glm('Endemics~log_Area+log_Elevation+log_Nearest+log_Adjacent',
                 data=myData, family=sm.families.Tweedie()).fit()
print(mod_quasi.summary())
```

Generalized Linear Model Regression Results

Dep. Variable:	Endemics	No. Observations:	30
Model:	GLM	Df Residuals:	25
Model Family:	Tweedie	Df Model:	4
Link Function:	log	Scale:	3.7004
Method:	IRLS	Log-Likelihood:	nan
Date:	Sat, 06 Nov 2021	Deviance:	95.660
Time:	22:38:58	Pearson chi2:	92.5
No. Iterations:	7		
Covariance Type:	nonrobust		

	coef	std err	z	P> z	[0.025	0.975]
Intercept	2.4768	0.881	2.810	0.005	0.749	4.204
log_Area	0.2750	0.056	4.894	0.000	0.165	0.385
log_Elevation	0.0125	0.178	0.070	0.944	-0.336	0.361
log_Nearest	-0.0613	0.041	-1.504	0.133	-0.141	0.019
log_Adjacent	-0.0458	0.021	-2.164	0.030	-0.087	-0.004

Pour plus d'informations sur la régression de Poisson avec Statsmodels, cliquez [ici](#).

ANNEXES

A. Diagonalisation versus Singular Value Decomposition

	Diagonalisation	Singular Value Décomposition
Matrice utilisée	Matrice des corrélations $C = Z^T Z$	Matrice des données centrées et réduites Z
Formule de décomposition	$C = P D P^T$	$Z = U \Delta V^T$
Valeur propre	λ_α	$\lambda_\alpha = \frac{\delta_\alpha^2}{n}$
Coordonnées des individus sur l'axe α	$F_\alpha(i) = \sum_{k=1}^K p_{\alpha k} z_{ik}$	$F_\alpha(i) = \delta_\alpha u_\alpha(i)$
Coordonnées des variables sur l'axe α	$G_\alpha = p_\alpha \sqrt{\lambda_\alpha}$	$G_\alpha = V^T \sqrt{D}$ où $D = \begin{pmatrix} \lambda_1 & 0 & \dots \\ 0 & \lambda_K & \end{pmatrix}$

BIBLIOGRAPHIE

- [Abd03] Hervé Abdi. Factor rotations in factor analyses. *Encyclopedia for Research Methods for the Social Sciences*. Sage : Thousand Oaks, CA, pages 792–795, 2003.
- [B⁺73] Jean-Paul Benzécri et al. *L'analyse des données*, volume 2. Dunod Paris, 1973.
- [B⁺21] Régis Bourbonnais et al. *Econométrie*. Dunod, 2021.
- [Bre82] A Brefort. Letude des races canines a partir de leurs caracteristiques qualitatives. *Groupe HEC-Jouy en Josas*, 1982.
- [CHMLR19] Pierre-André Cornillon, Nicolas Hengartner, Eric Matzner-Løber, and Laurent Rouvière. *Régression avec R-2e édition*. EDP sciences, 2019.
- [EP98] Brigitte Escofier and Jérôme Pagès. Analyses factorielles simples et multiples. *Dunod, Paris*, 1998.
- [Gif90] Albert Gifi. *Nonlinear multivariate analysis*. Wiley-Blackwell, 1990.
- [Hil16] Joseph M Hilbe. *Practical guide to logistic regression*. crc Press, 2016.
- [HLP16] François Husson, Sébastien Lé, and Jérôme Pagès. *Analyse de données avec R*. Presses universitaires de Rennes, 2016.
- [Hot33] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6) :417, 1933.
- [JJ99] Johnston Jack and Dinardo John. Méthodes économétriques. *éd ECONOMICA, 4ème éd*, 1999.
- [Kai91] Henry F Kaiser. Coefficient alpha for a principal component and the kaiser-guttman rule. *Psychological reports*, 68(3) :855–858, 1991.
- [LSHJ13] Stanley Lemeshow, Rodney X Sturdivant, and David W Hosmer Jr. *Applied logistic regression*. John Wiley & Sons, 2013.
- [Pag13] Jérôme Pagès. *Analyse factorielle multiple avec R*. EDP sciences, 2013.
- [Rak] Ricco Rakotomalala. Pratique de l'analyse discriminante linéaire.

- [Rak11] Ricco Rakotomalala. Pratique de la régression logistique. *Régression logistique binaire et polytomique*, Université Lumière Lyon, 2 :258, 2011.
- [Ras18] Sebastian Raschka. Mlxtend : Providing machine learning and data science utilities and extensions to python's scientific computing stack. *The Journal of Open Source Software*, 3(24), April 2018.
- [Rou15] Laurent Rouvière. Régression logistique avec r. *Universités Rennes*, 2, 2015.
- [Sap06] Gilbert Saporta. *Probabilités, analyse des données et statistique*. Editions technip, 2006.
- [Sté12] Tufféry Stéphane. *Data mining et statistique décisionnelle : l'intelligence des données*. Editions Technip, 2012.
- [Ten07] Michel Tenenhaus. *Statistique : méthodes pour décrire, expliquer et prévoir*, volume 680. Dunod Paris, France :, 2007.
- [TM87] Ronald J Tallarida and Rodney B Murray. Mann-whitney test. In *Manual of Pharmacologic Calculations*, pages 149–153. Springer, 1987.



En formation de Data Scientist et Quantitative Risk Management à l'Ecole Nationale de la Statistique et de l'Analyse de l'Information (ENSAI-Bruxelles) en France, **Duvérier DJIFACK ZEBAZE** est originaire du Cameroun, pays situé en Afrique Centrale. Il est titulaire d'un Master 2 recherche en Economie quantitative obtenu à la Faculté des Sciences Economiques et de Gestion Appliquée de l'Université de Douala. Il est également un ancien élève de l'Ecole Nationale de la Statistique et de l'Analyse Economique (ENSAE-Dakar) au Sénégal. **Statistique avec Python : de la théorie à la pratique** est son tout premier ouvrage consacré à la pratique du langage de programmation Python dans le domaine de la statistique.