Formation Introduction à R — URFIST Bordeaux

Frédéric Santos
frederic.santos@u-bordeaux.fr

22–23 octobre 2015

Contenu de la formation

1. Présentation du logiciel R
2. Objets R, instructions élémentaires en ligne de commande
3. Concevoir, importer et manipuler des jeux de données
4. Faire de la statistique descriptive
5. Décrire graphiquement ses données avec R
6. Réaliser quelques uns des tests statistiques usuels
7. TP : application « libre » sur un jeu de données

Généralités sur le logiciel R

- R : à la fois logiciel de statistique et langage de programmation.
- Logiciel libre, gratuit et multiplateforme, développé par des bénévoles.
- Utilisable en ligne de commande, ou *via* une interface graphique : R Commander.
- Logiciel le plus utilisé dans la recherche en sciences humaines et bio-médicales.

P.-A. Cornillon et co-auteurs : *Statistique avec R*. Presses universitaires de Rennes, 2012.

M. Maumy-Bertrand et F. Bertrand : *Initiation à la statistique avec R*. Dunod, 2010.

Obtenir R

On peut trouver le logiciel R sur son site officiel (le CRAN, Comprehensive R Archive Network) :

`http://cran.r-project.org/`

En plus de la version Windows :

- R est nativement disponible sur toutes les distributions Linux usuelles (openSUSE, Mageia, Ubuntu, etc.), et se maintiendra automatiquement à jour si l'utilisateur ajoute les dépôts adéquats.
- Pour Mac OS X, en fonction de la version de votre OS (10.4, 10.6, 10.9, ...), R peut nécessiter l'installation de certains logiciels tiers pour fonctionner correctement. Consulter la FAQ spécifique sur le CRAN.

Comment obtenir de l'aide pour utiliser R ?

- Une communauté d'utilisateurs très active, à retrouver sur : `http://forums.cirad.fr/logiciel-R/index.php`
- De la documentation en Français et en Anglais sur le CRAN : `http://cran.r-project.org/other-docs.html`
- De nombreuses chaînes Youtube à découvrir (par exemple celle de François Husson), pour des didacticiels vidéo traitant de méthodes avancées.
- R est open source : le code des fonctions peut être consulté (simplement en tapant leur nom dans la console R !), et l'aide de chaque fonction fournit l'adresse mail de son auteur.

La console R

R se présente basiquement comme une console. Le signe > (*prompt*) en début de ligne indique R est prêt à recevoir une commande.

Tester les commandes suivantes (touche Entrée pour les exécuter) :

```
> 2+3
> 5^2
```

Si vous interrompez la saisie d'une commande en appuyant trop tôt sur la touche Entrée, R offre la possibilité de compléter la saisie de la commande sur la ligne suivante (la ligne commence alors par le symbole +) :

```
> 9-
+ 1
```

Comment utiliser R ?

Il y a plusieurs approches possibles pour utiliser le logiciel :

1. Utiliser le langage R en saisissant des lignes de commandes directement dans la console R (déconseillé).
2. Utiliser *uniquement* l'interface graphique R Commander, et réaliser toutes les analyses « à la souris » (déconseillé).
3. Utiliser séparément un éditeur de script (gedit, Notepad++, Tinn-R, ...) dans lequel on saisira et sauvegardera nos commandes R ; et une console R dans laquelle on exécutera (par un copier-coller) ces scripts.
4. Utiliser un EDI (environnement de développement intégré) tel que R Studio, intégrant à la fois un éditeur de script, une console R et plusieurs raccourcis utiles.

Base et packages additionnels

R est livré avec une certaine collection de fonctions de base, permettant de réaliser les tâches les plus courantes (représentations graphiques usuelles, tests classiques, etc.).

Pour mettre en œuvre des méthodes « rares » ou spécifiques à une discipline, il existe plusieurs centaines de *packages* additionnels, dont on peut obtenir la liste grâce à la commande :

```
install.packages()
```

Il n'est nécessaire de les installer qu'une seule fois, mais ils doivent ensuite être chargés à chaque démarrage du logiciel grâce à la commande `library`.

Quelques commandes basiques pour R

Trouver une commande répondant à un besoin précis (accès Internet requis) : `RSiteSearch(" ")`.

Exemple : `RSiteSearch("mantel test")`.

Obtenir de l'aide sur une commande : `help()`.

Exemple : `help(mean)`.

Installer un package additionnel : `install.packages(" ")`.

Exemple : pour l'installer l'interface graphique, `install.packages("Rcmdr", dep=TRUE)`.

Charger un package additionnel : `library()`.

Exemple : pour charger l'interface graphique : `library(Rcmdr)`.

Premier exemple d'utilisation

On se propose de :

1. Rechercher de l'information sur le package et la fonction permettant de réaliser un test de normalité assez peu connu, le test d'Anderson-Darling.
2. Installer le package additionnel permettant de réaliser ce test.
3. Charger ce package.
4. Consulter l'aide de la fonction correspondante.
5. Exécuter (par un copier-coller dans la console R) les exemples figurant en bas de la page de l'aide de cette fonction.

Objets R — I

Objet

Un objet R est une « boîte étiquetée » dans laquelle vous pouvez placer l'information qui vous intéresse.

Plus formellement, il s'agit d'un espace de stockage dans la mémoire de R, auquel on va donner un nom, et que l'on va remplir avec un certain type d'information ou de données. On peut ensuite y accéder à tout moment et réutiliser cette information ou ces données dans le cadre de calculs, de tests, etc.

Ces objets peuvent être de différents *types* : c'est une notion importante car chaque fonction de R ne peut travailler qu'avec un type précis d'objet.

Objets R — II

1. Un **vecteur** est une suite indexée de valeurs toutes du même type : une suite de nombres, de caractères, etc. On peut par exemple saisir un vecteur avec la commande c :

```
> x <- c(2, 5.9, 6.2, 8.3) # définir un vecteur x
> x # afficher x
[1] 2.0 5.9 6.2 8.3
> x[1] # accéder au premier élément de x
[1] 2
> x[3] # accéder au troisième élément de x
[1] 6.2
> x[c(1,2)] # accéder aux éléments n° 1 et 2 de x
[1] 2.0 5.9
```

Note : <- est l'*opérateur d'affectation* : il sert à ranger une certaine information dans un certain objet.

Objets R — III

2. Un **facteur** est un vecteur qui contient uniquement des données qualitatives.

```
> fac <- c("a", "a", "b", "a", "b")
> fac # à ce stade, fac n'est pas un facteur
[1] "a" "a" "b" "a" "b"
> fac <- factor(fac) # on le convertit en facteur
> fac # la gestion est alors différente !
[1] a a b a b
Levels: a b
> levels(fac) # lister les niveaux du facteur
[1] "a" "b"
> nlevels(fac) # nombre de niveaux du facteur
[1] 2
```

Objets R — IV

3. Une **matrice** est un tableau constitué de valeurs toutes du même type : des nombres, des caractères, etc. Chaque valeur est indexée par son numéro de ligne et de colonne.

```
> M <- matrix(c(1,2,3,4), ncol=2)
> M # afficher la matrice
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> M[1,2] # la valeur en ligne 1 et colonne 2
[1] 3
> M[ ,1] # le contenu de toute la colonne 1
[1] 1 2
> M[2, ] # le contenu de toute la ligne 2
[1] 2 4
```

Objets R — V

4. Une **liste** est un objet indexé très général permettant de stocker des informations de type potentiellement différent : le premier élément peut être un nombre, le deuxième un caractère, le troisième une matrice, etc.

   Pour créer une liste, on utilise la commande `list` (mais a priori ce sera rarement le cas en pratique).

   Pour accéder au k^{e} élément d'une liste nommé `L`, on utilise une notation en double crochet : `L[[k]]`.

Objets R — VI

5. Un **data frame** est le format par défaut donné par R aux jeux de données que l'on importe depuis un tableur.

   Il s'agit d'un tableau de données à n lignes (1 ligne $=$ 1 individu) et p colonnes (1 colonne $=$ 1 variable). Il peut y avoir des variables de différents types (des facteurs, des variables numériques, etc.)

   Comme dans le cas des matrices, chaque valeur est repérée par son numéro de ligne et de colonne : si le data frame s'appelle `df`, la commande `df[2,3]` permet d'accéder à la valeur de la 3e variable du 2e individu.

Objets R — VII

On peut faire passer (sous réserve que l'opération ait du sens) un objet d'un type à un autre : par exemple, si les chiffres 1 et 2 n'indiquent pas des données numériques mais l'appartenance à un groupe donné, on peut vouloir convertir un vecteur numérique en facteur. On peut également vouloir convertir un data frame en matrice, etc. On pioche alors parmi les fonctions suivantes : `as.factor`, `as.numeric`, `as.data.frame`, `as.matrix`, etc.

```
> x <- c(1,2,1,2,2)
> x
[1] 1 2 1 2 2
> x <- as.factor(x)
> x
[1] 1 2 1 2 2
Levels: 1 2
```

Opérations élémentaires

Pour résumer :

- on peut affecter une certaine valeur à un objet `x` en utilisant `x <- ...` ;
- on peut afficher l'objet `x` en tapant simplement `x` dans la console (ou `print(x)` qui a le même effet) ;
- les objets R sont *indexés* : on peut accéder à un certain élément de `x` (ou un certain ensemble d'éléments) en précisant leur indexation entre crochets ;
- le type d'un objet est une caractéristique "flottante", modifiable. Il existe des fonctions pour convertir un objet vers un nouveau type (sans en modifier le contenu), afin qu'il soit traité différemment.

Opérations sur des vecteurs — I

Quelques commandes élémentaires à utiliser sur des vecteurs numériques :

```
> x <- c(10, 12, 8, 10) # ex. : des notes sur 20
> x
[1] 10 12 8 10
> mean(x) # calculer la moyenne de x
[1] 10
> length(x) # nb de valeurs dans x
[1] 4
> max(x) # maximum de x
[1] 12
> x/2 # les mêmes notes sur 10
[1] 5 6 4 5
```

Opérations sur des vecteurs — II

```
> y <- c(9, 14) # on saisit deux nouvelles notes
> z <- c(x,y) # z = concaténation de x et y
> z # afficher z
[1] 10 12 8 10 9 14
> c(y,x) # attention à l'ordre : c(x,y) ≠ c(y,x) !
[1] 9 14 10 12 8 10
> z[1:3] # 1:3 est un raccourci pour c(1,2,3)
[1] 10 12 8
> z[-3] # afficher tous les éléments de z sauf le n°3
[1] 10 12 10 9 14
> sort(z) # tri croissant de z
[1] 8 9 10 10 12 14
```

Opérations sur des vecteurs — III

On peut attribuer des noms à chaque valeur de z :

```
> names(z) <- c("Jim","Eva","Anna","Dan","Mary","Leo")
> z
Jim Eva Anna Dan Mary Leo
10   12    8   10    9   14
```

On peut alors accéder aux valeurs par leur nom plutôt que leur index :

```
> z["Anna"]
Anna
8
> z["Anna"] <- 9 # changer la note d'Anna
> z # afficher la série de notes corrigée
Jim Eva Anna Dan Mary Leo
10   12    9   10    9   14
```

Manipulation d'un data frame

Comme spécifié auparavant, chaque feuille de données importée dans R se verra attribuer la classe `data.frame` : il est donc essentiel de savoir manipuler de tels objets puisqu'ils seront à la base de toute pratique de R.

Nous allons voir les principales fonctions à connaître pour cette classe d'objets, et surtout, la façon d'importer un data frame dans R !

Avant cela, deux digressions :

- sur la notion de répertoire courant ;
- sur les règles de mise en forme d'un jeu de données.

Répertoire courant

Le *répertoire courant* d'une session R est le répertoire où (sauf instructions contraires) le logiciel ira chercher les feuilles de données, sauvegarder les résultats, figures, etc. Il est donc essentiel qu'il soit correctement défini, à chaque fois que l'on ouvre le logiciel.

- Sous Windows : aller dans Fichier > Changer le répertoire courant.
- Sous Linux : il suffit tout simplement d'ouvrir une console directement dans le répertoire où l'on souhaite travailler.
- Sous Mac OS : icône Divers > Changer de répertoire de travail.
- Tous OS : utiliser la commande `setwd()`dans la console R.

Mise en forme des données — I

Important

Utiliser R implique de respecter les conventions usuelles de mise en forme des feuilles de données, faute de quoi il sera impossible de les charger !

1. Pour éviter les soucis, **texte brut uniquement** (.txt, .csv) : jamais de .xls ou .ods, encore moins de .xslx.
2. Conséquences : aucune mise en forme autorisée ; **fusion de cellules interdite** ; les informations ne sont indiquées *que* par du texte, jamais par de la mise en forme ou des couleurs.
3. Individus en lignes, variables en colonnes.
4. Correspondance parfaite entre lignes et individus : une ligne correspond à un individu (au sens d'unité statistique).

Mise en forme des données — II

5. Doublons interdits dans les noms des individus.
6. Certains caractères spéciaux sont interdits : ♯, guillemets, apostrophes. Autres caractères spéciaux (accents, espaces, lettres grecques) fortement déconseillés pour éviter les problèmes d'encodage.
7. Les données manquantes s'indiquent de préférence par une case vide, ou sinon par un codage **unique** qui sera spécifié au logiciel (par exemple : NA, pour Not Available).
8. Réciproque : pas de case vide si les données sont connues.
9. Séparateur décimal (point ou virgule) unique pour tout le jeu de données.
10. Utiliser la fonction « Commentaire » de votre tableur pour toute info qui n'a rien à voir avec le traitement statistique.

Mise en forme des données — III

11. Jamais de ligne ou de colonne vide ou « aérer » ou « séparer ».
12. Noms de variables courts et sans espaces (utiliser des underscores à la place), occupant la première ligne (et elle seule !) des données.

Important

Penser à vérifier *minutieusement* les données avant toute analyse.

Mise en forme des données — IV

	A	B	C	D	E	F	G	H	
1	num_inventaire	sexe	long_max	larg_max	largeur_bizygomatique	Opisthion_Vertex	mast_hauteur_G	mast_largeur_G	mas
2	17473	M	183	148	132	145	26	58	
3	17474	M	186	131	136	142	27	19	
4	17476	M	174	146	128	140	25	62	
5	17477	M	166	141	124	142	31	5	
6	17479	M	172	143	131	143	29	27	
7	17480	M	178	135	130	147	25	52	
8	17482	M	185	140		148	29	44	
9	17484	M	183	134	137	144	24	68	
10	17486	M	171	139	137	150	27	2	
11	17487	M	174	136	130	143	29	94	
12	17488	M	177	133	122	146	27	29	
13	17490	M	199	149	146	146	29	58	
14	17491	M	174	142	130	151	33	14	
15	17495	M	175	138	122	149	24	45	
16	17497	M	175	146	138	137	26	89	
17	17498	M	180	139	135	144	31	81	
18	17499	M	166	147	133	144	26	58	

FIGURE : Un exemple de mise en forme correcte d'une feuille de données...

Mise en forme des données — V

	A	B	C	D	E	F	G	H	I	J	K
1	Num individus	**Racine appendiciforme**						**Tubercule paracone**			
2		UI1 D		UI2 D		UC D		LM1 D		LM2 D	
3		Stade	Seuil	Stade	Seuil	Stade	Seuil	Stade	Seuil	Stade	Seuil
4	PREMIERE OBS.										
5	Indiv1 Droite	?	?	0	A	NO	NO			NO	NO
6	Indiv1 Gauche	1	A	0	A	0	A			0	A
7	Indiv2 Droite	1	A	NO	NO	0	A	0	A	1	P
8	Indiv2 Gauche			1	P	3	P	0	A	0	A
9	Indiv 3 Droite	2	P	1	P	3	P	NO	NO	NO	NO
10	Indiv 3 Gauche	1	A	1	P	0	A	1	A	NO	NO
11	Indiv 4 Droite			0	A			2	P	NO	NO
12	Indiv 4 Gauche	??	??					?	?	0	A
13											

FIGURE : ... et un exemple de tout ce que ne doit pas être une feuille de données.

Importer une feuille de données — I

La commande générale à utiliser est `read.table`, ou plus spécifiquement `read.csv` pour les fichiers CSV.

Exemple

Après avoir défini correctement le répertoire courant, exécuter la commande :

```
> dat <- read.csv("data_archeo.csv", header=TRUE,
row.names=1, sep=";", dec=",", na.strings="")
```

Le jeu de données correspondant au fichier "data_archeo.csv" sera désormais stocké dans l'objet R `dat`.

Pour des détails à propos de chaque argument de la commande, consulter l'aide de la fonction `read.csv`.

Importer une feuille de données — II

Astuce/Raccourci : la commande `read.csv` convient par défaut aux fichiers en norme anglaise (séparateur de colonnes = virgule, séparateur décimal = point), tandis que la commande `read.csv2` convient par défaut aux fichiers en norme française (séparateur de colonnes = point-virgule, séparateur décimal = virgule).

Ici, comme le fichier est en norme française, la simple commande :
`> dat <- read.csv2("data_archeo.csv", na.strings="")`
suffit à le charger correctement.

Résumé et contrôle d'un tableau de données — I

Une fois importé, on peut connaître différentes choses sur le jeu de données et procéder à diverses vérifications :

```
> dim(dat) # son nb de lignes et de colonnes
[1] 43 8
```

```
> colnames(dat) # le nom des variables
[1] "Site" "Sexe" "FM1"
[4] "TM6" "RM1" "Hypoplasie"
[7] "Presence_ parure" "Orientation_ corps"
```

Résumé et contrôle d'un tableau de données — II

```
> str(dat) # contrôle du type des variables
'data.frame': 43 obs. of 8 variables:
$ Site : Factor w/ 2 levels "A","B": 1 1 1 1 1 1 1 1 1 1
...
$ Sexe : Factor w/ 2 levels "F","M": 2 2 2 1 2 2 1 2 2 1
...
$ FM1 : num 420 509 465 426 465 ...
$ TM6 : num 51.8 58 53.4 49.3 59.5 ...
$ RM1 : num 251 256 251 244 251 ...
$ Hypoplasie : Factor w/ 2 levels "Non","Oui": 1 1 1 2 1 1
1 1 1 2 ...
$ Presence_ parure : Factor w/ 2 levels "Non","Oui": 1 1 1
2 1 1 2 1 1 1 ...
$ Orientation_ corps: Factor w/ 4 levels
"Est","Nord","Ouest",..: 2 2 1 3 1 1 3 1 2 4 ...
```

Résumé et contrôle d'un tableau de données — III

Ainsi que d'autres commandes incontournables dont la sortie n'est pas reproduite ici :

```
> head(dat) # affichage des 6 premières lignes
> dat[1:6, ] # même chose, de manière plus "manuelle"

> summary(dat) # résumé des données
```

Résumé (`summary`)

La commande `summary` permet notamment de détecter toute erreur de saisie dans les données (présence d'un max ou min aberrant, variable reconnue comme facteur alors qu'elle devrait être numérique, etc.) et son emploi doit être systématique après avoir importé une feuille de données.

Manipulation d'un data frame — I

1. **Travailler sur une variable précise**. Dans le jeu de données précédemment importé, imaginons que l'on veuille calculer l'écart-type (`sd`) de la variable FM1, qui est en colonne 3. Il existe plusieurs façons équivalentes de procéder :
   - la commande `sd(dat[ , 3])` calcule l'écart-type de la troisième colonne du tableau `dat`, ce qui est bien le but recherché ;
   - plus explicite : on peut faire appel au nom de variable plutôt qu'à son index, par exemple en écrivant `sd(dat[ , "FM1"])` ;
   - encore mieux : `sd(dat$FM1)`. Le signe `$` permet de faire appel à une variable précise d'un data frame de façon condensée.

Manipulation d'un data frame — II

2. **Sélection de sous-ensembles.** On peut « filtrer » le jeu de données selon des critères précis.

Utilisation de subset

```
> femmes <- subset(dat, Sexe=="F")
> femmes
> femmes_A <- subset(dat, Sexe=="F" & Site=="A")
> femmes_A
> pas_sud <- subset(dat, Orientation_corps !="Sud")
> pas_sud
```

On peut aussi tout simplement écrire :

```
> femmes <- dat[dat$Sexe=="F", ]
```

Manipulation d'un data frame — III

3. **Centrer et réduire des variables.** On utilise la commande `scale`.

Utilisation de `scale`

Que réalise le code suivant ?

```
> head(dat)
> cent <- scale(dat[ , 3:5], scale=TRUE, center=TRUE)
> head(cent)
> dat[ , 3:5] <- cent
> head(dat)
```

Manipulation d'un data frame — IV

4. **Appliquer une fonction en lignes ou colonnes.** On peut souhaiter appliquer la même fonction à toutes les lignes ou toutes les colonnes d'un tableau de données (par exemple, calculer la médiane de toutes les colonnes, ou la somme de toutes les lignes). La fonction dédiée est `apply`.

Utilisation de `apply`

```
> cent <- scale(dat[ , 3:5], scale=TRUE, center=TRUE)
> head(cent)
> apply(cent, MARGIN=2, FUN=sum, na.rm=TRUE)
> apply(cent, MARGIN=1, FUN=max, na.rm=TRUE)
```

`MARGIN=1` permet d'agir sur les lignes, `MARGIN = 2` agit sur les colonnes.

Manipulation d'un data frame — V

5. **Repérer des valeurs manquantes.** Les valeurs manquantes sont indiquées par `NA` sous R. Observer les résultats du code suivant.

Comment repérer et éliminer des valeurs manquantes ?

```
> net <- na.omit(dat)
> net

> x <- dat$FM1 ; names(x) <- rownames(dat)
> is.na(x)
> which(is.na(x))
> y <- x[! is.na(x)]
> y
```

Manipulation d'un data frame — VI

6. **Ajouter et supprimer des variables**. Tester les commandes suivantes :

```
> head(dat)
> dat$Somme <- dat$FM1 + dat$RM1
> head(dat)
> dat$Somme <- NULL
> head(dat)
```

Export de données

Il est possible d'écrire dans un fichier .txt ou .csv le contenu d'un data frame, à l'aide des commandes `write.table`, `write.csv` ou encore `write.csv2` (consulter l'aide de chaque commande pour connaître leurs particularités).

Par exemple, pour exporter le data frame `net` créé dans la diapositive précédente, on écrira :

```
> write.csv2(net, "donnees_netoyees.csv")
```

Le fichier est écrit dans le répertoire courant de la session.

Obtenir un résumé par groupes d'individus — I

Lorsque les individus sont naturellement organisés en groupes (par exemple femmes et hommes sur le fichier utilisé dans notre exemple), un résumé global peut être insatisfaisant. Obtenir un résumé séparé pour chaque groupe d'individu est plus pertinent. Il existe de très nombreux moyens de réaliser cela avec R, mais en voici trois.

1. Utiliser la fonction `describeBy` du package `psych`. On obtient un résumé complet, semblable à celui de la commande `summary`, grâce à la commande :
   ```
   > describeBy(dat, dat$Sexe)
   ```

Obtenir un résumé par groupes d'individus — II

2. Utiliser la fonction `split`. Cette dernière permet de séparer un data frame selon les niveaux de l'un de ses facteurs, et de stocker les data frames correspondant aux différents niveaux dans une liste.

Les fonctions `split` et `lapply`

```
> sp <- split(dat, dat$Sexe)
> str(sp)
> sp[[1]]
> sp[[2]]

> lapply(sp, FUN=summary)
```

La fonction `lapply` applique une même fonction à l'ensemble des éléments d'une liste.

Obtenir un résumé par groupes d'individus — III

3. Utiliser la fonction aggregate. Elle prend en argument un data frame, le divise en sous-groupes définis par un facteur, et applique *une* fonction (moyenne, écart-type, ...) à chaque sous-groupe.

La fonction aggregate

```
> aggregate(dat[,c("FM1", "RM1")],
by=list(Sexe=dat$Sexe), FUN=mean, na.rm=TRUE)

Sexe   FM1      RM1
1 F 414.2471 222.7993
2 M 445.0561 241.7250
```

Corrélations entre plusieurs variables

La fonction `cor` donne le coefficient de corrélation entre deux variables, ou une matrice de corrélation si plus de deux variables sont impliquées.
Tester les commandes suivantes :

La fonction cor

```
> cor(dat$RM1, dat$FM1, use="complete.obs")
# l'argument use="complete.obs" est nécessaire en
présence de valeurs manquantes

> cor(dat[ , c("RM1", "FM1", "TM6")],
use="complete.obs")
```

Création de graphiques avec R

R permet de créer des graphiques d'excellente qualité avec des possibilités de personnalisation infinies.

Le package standard de R comporte déjà de bonnes possibilités graphiques, mais quelques packages additionnels permettent de réaliser des graphiques plus fins, au rendu plus convaincant. On citera notamment les packages `ggplot2` (non abordé dans cette formation), ou `lattice`.

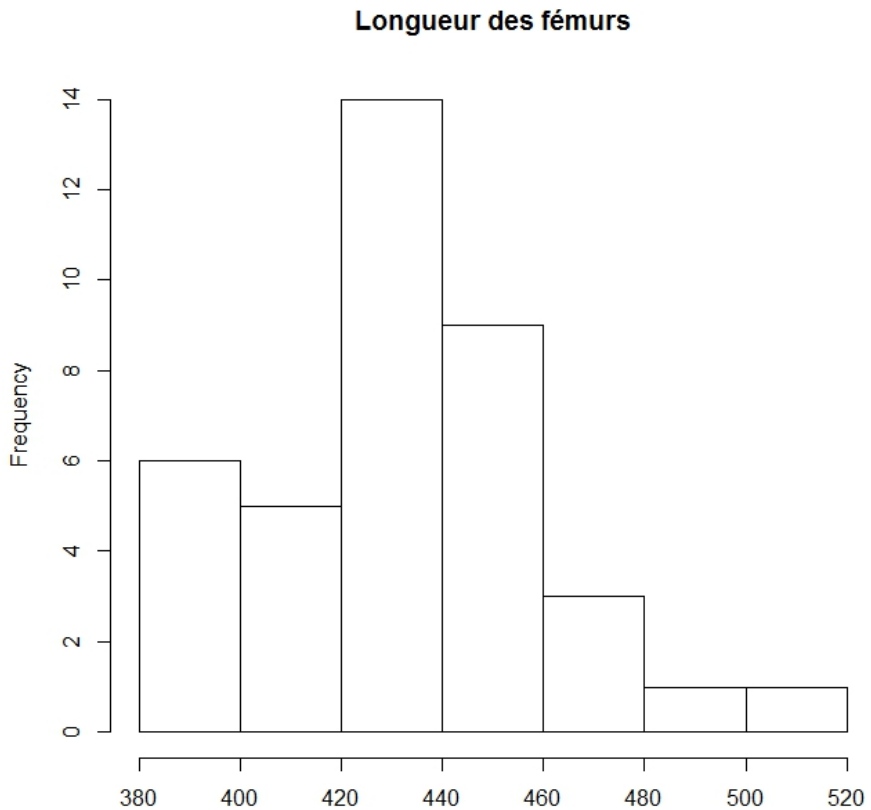
D. Sarkar *Lattice. Multivariate Data Visualization with R.* Springer, 2008.

H. Wickham *ggplot2. Elegant Graphics for Data Analysis.* Springer, 2010.

Histogramme

Un exemple avec les mêmes données que précédemment

```
> hist(dat$FM1, main="Longueur des fémurs",
xlab="Longueur")
```



Histogrammes par groupes d'individus — I

En présence de données structurées en groupes d'individus très différents, les représentations graphiques globales ont peu d'intérêt. Effectuer une représentation séparée pour chaque groupe est plus pertinent : on parle alors de graphiques conditionnels.

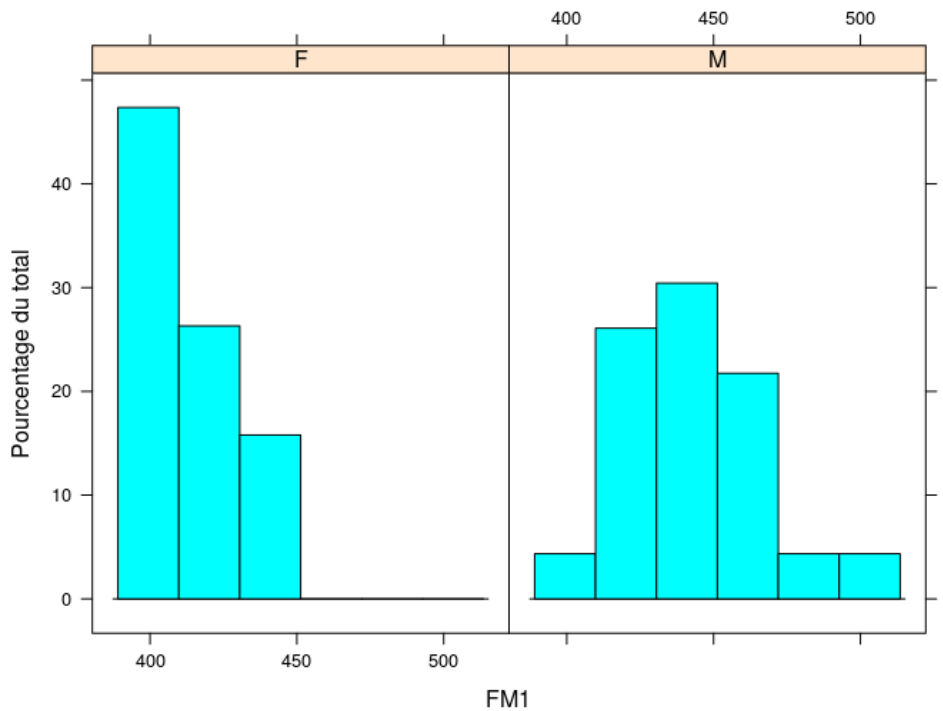
Dans R, les fonctions de base permettent toujours de réaliser de tels graphiques, mais il faut alors « ruser » et scripter un peu. Pour avoir directement accès à de telles représentations sans (trop d')effort, la library `lattice` est spécialisée dans les graphiques conditionnels.

En revanche, elle possède sa syntaxe propre, différente des fonctions de base.

Histogrammes par groupes d'individus — II

Histogrammes en fonction du sexe des individus

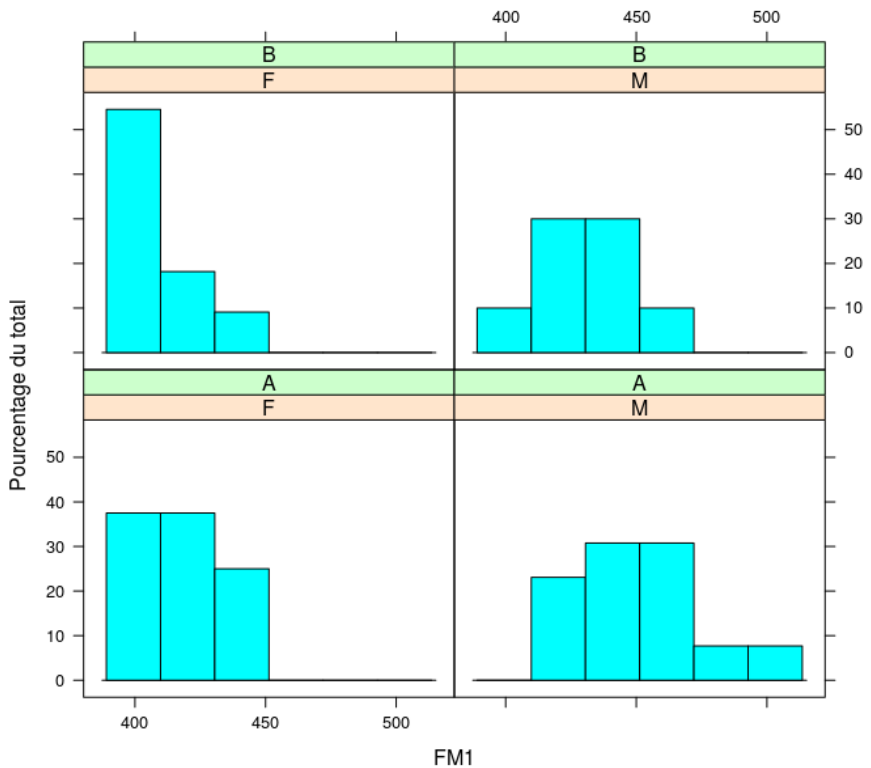
```
> library(lattice) # charge le package
> histogram(~ FM1 | Sexe, data=dat)
```



Histogrammes par groupes d'individus — III

Histogrammes en fonction du sexe et du site

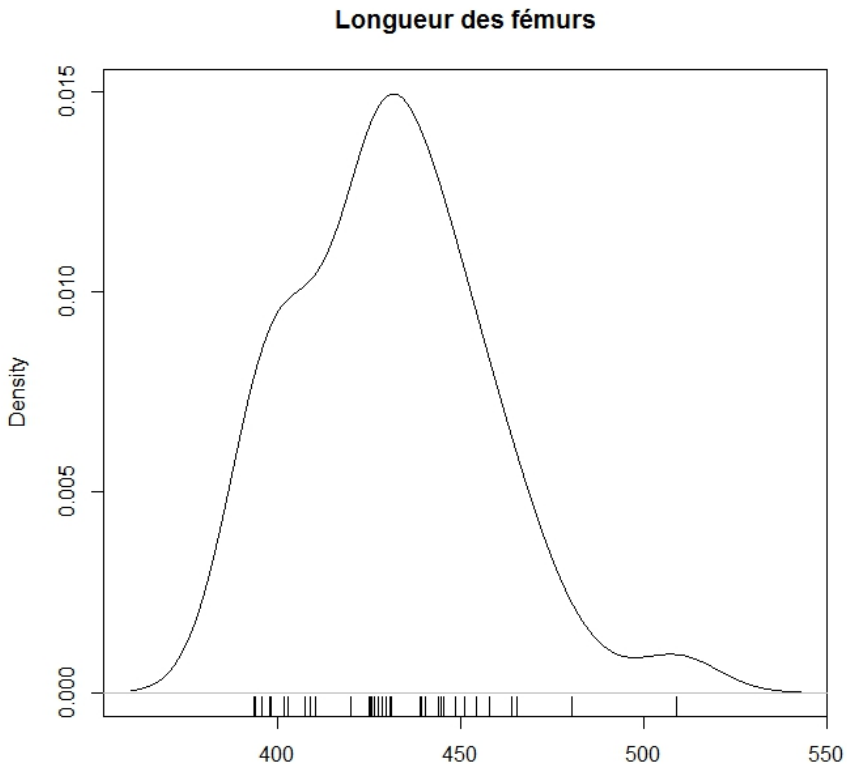
```
> histogram(~ FM1 | Sexe * Site, data=dat)
```



Estimation de densité (méthodes à noyau)

La fonction kde : Kernel density estimation

```
> kde <- density(dat$FM1, na.rm=TRUE)
> plot(kde, main="Longueur des fémurs")
> rug(dat$FM1) # ajoute les valeurs isolées
```

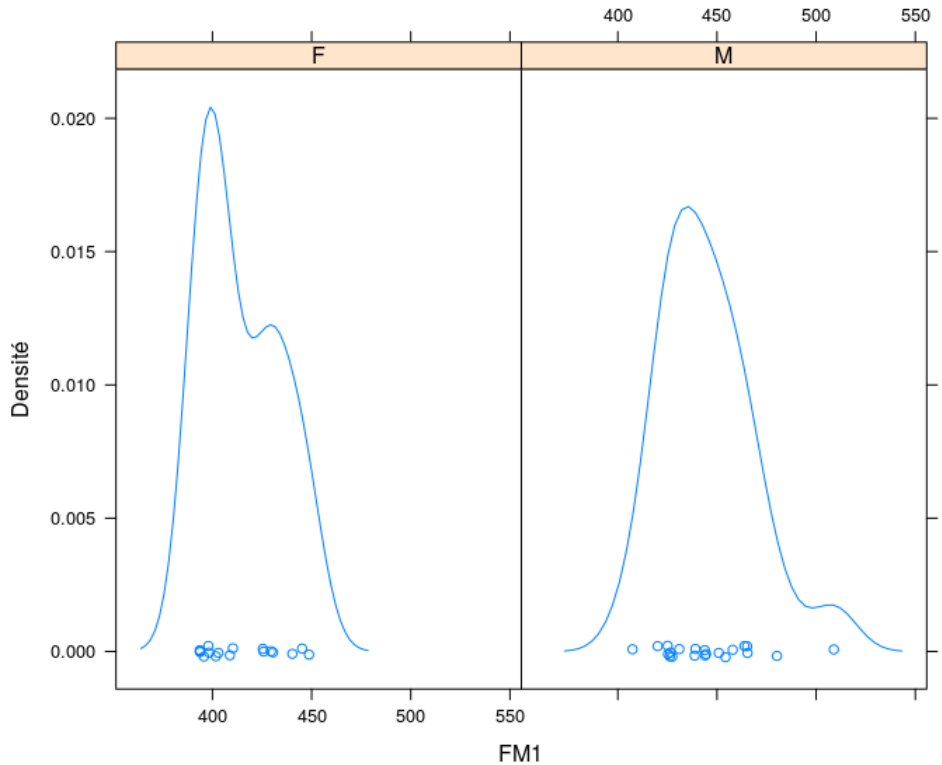


Estimation de densité — Graphiques conditionnels

La fonction `densityplot` du package `lattice`

```
> densityplot(~ FM1 | Sexe, data=dat)
```

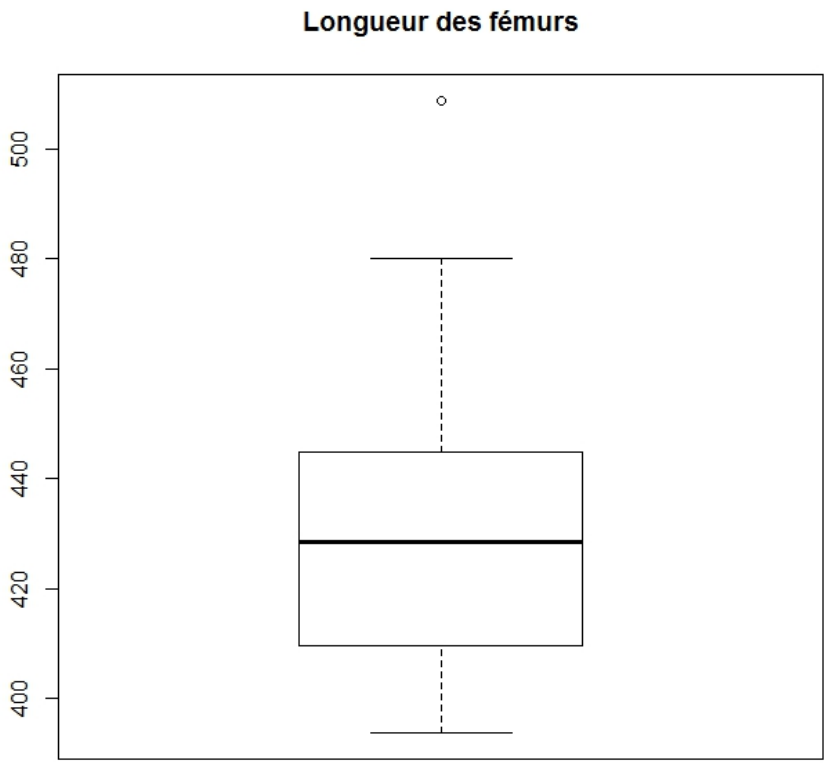
La syntaxe est la même que pour la commande `histogram`.



Boîte de dispersion (*boxplot*)

Un exemple avec les mêmes données que précédemment

```
> boxplot(dat$FM1, main="Longueur des fémurs")
```



Aparté : les formules

L'aide de nombreuses fonctions R (comme celle de `boxplot`) montre qu'on peut les appeler en leur donnant une instruction sous forme de *formule*. Une formule, en R, s'écrit :

`Y ~ X`

et signifie génériquement « `Y` en fonction de `X` ».

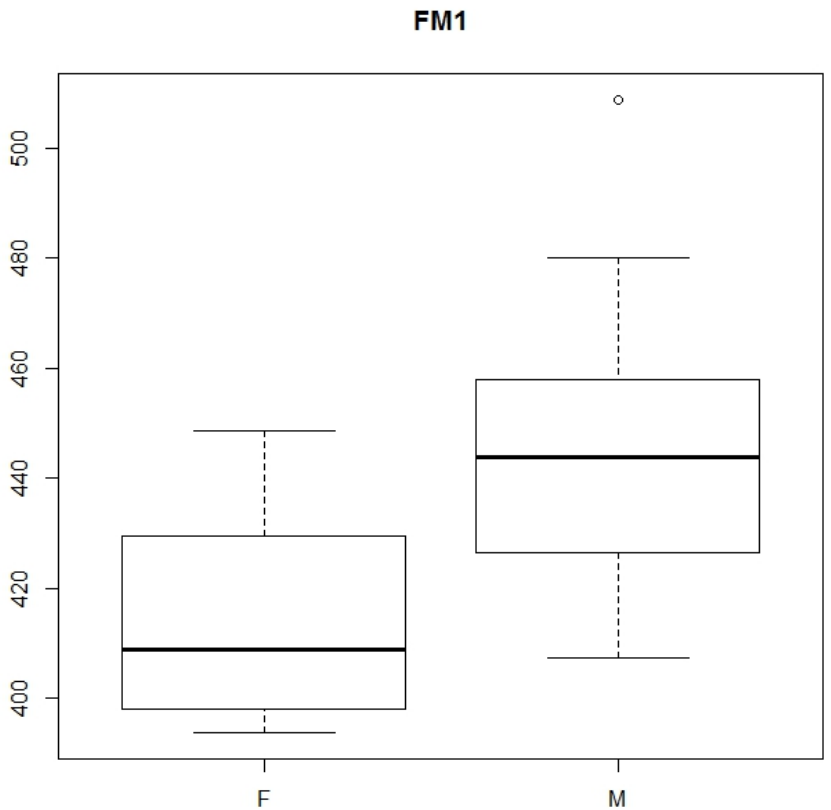
Remarques :

- `Y ~ X1 + X2` veut dire « `Y` en fonction de `X1` et `X2` » ;
- `Y ~ X1 * X2` veut dire « `Y` en fonction de l'interaction de `X1` et `X2` ».

Boîtes de dispersion en parallèle — I

Un seul facteur de classement

```
> boxplot(FM1 ~ Sexe, main="FM1", data=dat)
```



Boîtes de dispersion en parallèle — II

Deux facteurs de classement en interaction

```
> boxplot(FM1 ~ Sexe * Site, main="FM1", data=dat)
```

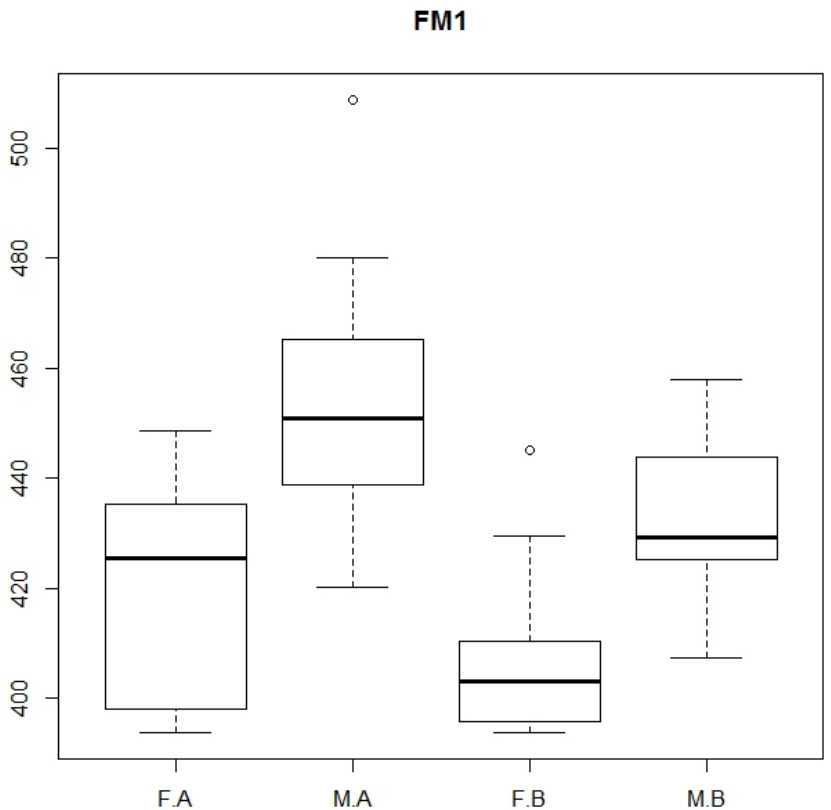
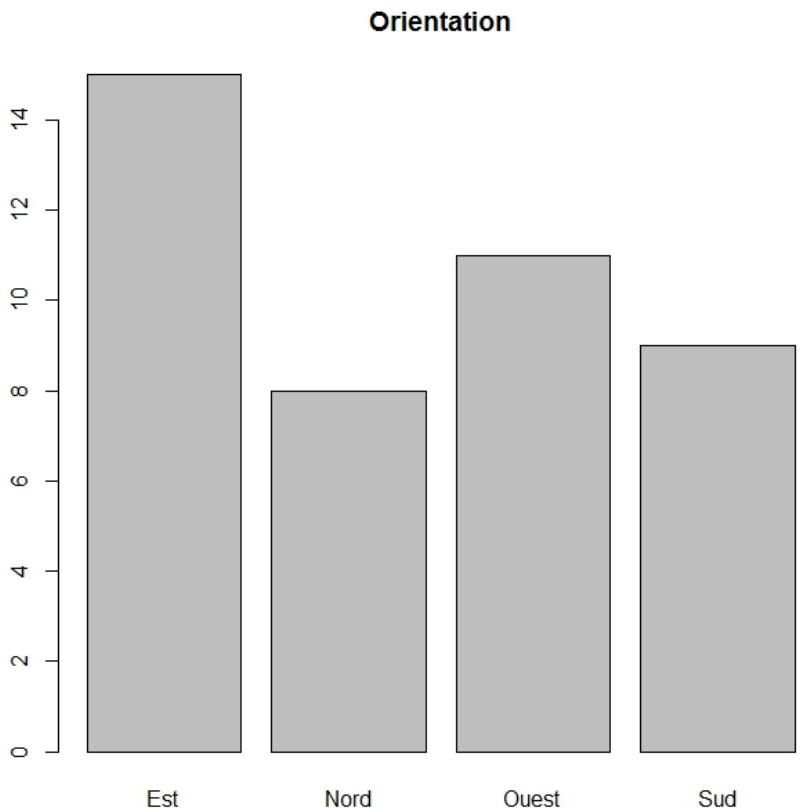


Diagramme en barres

Un exemple avec les mêmes données que précédemment

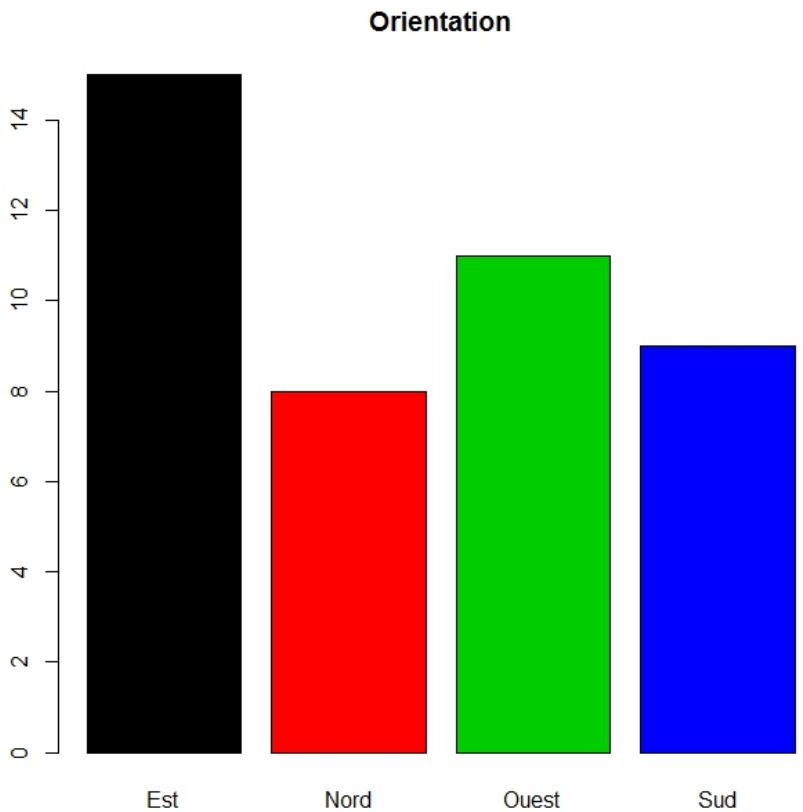
```
> plot(dat$Orientation_corps, main="Orientation")
```



Un peu de couleur...

Un exemple avec les mêmes données que précédemment

```
> plot(dat$Orientation_corps, main="Orientation",
col=1:4)
```



Gestion des couleurs — I

On peut spécifier les couleurs que l'on souhaite dans l'argument `col` de la commande `plot`.

Par défaut, R gère 8 couleurs dont on obtient la liste par la commande :

```
> palette()
[1] "black" "red" "green3" "blue" "cyan" "magenta"
[7] "yellow" "gray"
```

Comme dans l'exemple de la diapo précédente, on peut alors faire appel à une couleur soit par son nom (`"black"`, `"red"`, ...) soit par son numéro (1, 2, ...).

Gestion des couleurs — II

Au total, R gère bien sûr beaucoup plus de couleurs (657 au total), et on peut en obtenir la liste complète sur Internet.

Une possibilité intéressante est de définir la palette de couleurs comme un dégradé (palettes `rainbow`, `terrain.colors`, ...).

Pour plus d'information, consulter :
`http://rug.mnhn.fr/semin-r/PDF/semin-R_couleurs_MBaylac_080108.pdf`

Les options graphiques : par

La commande `par` permet de modifier de très nombreux paramètres graphiques (cf. aide de la commande) qui s'appliqueront à la prochaine fenêtre créée.
On retiendra notamment :

Augmenter la taille : cex

```
> par(cex=1.2) # augmenter la taille de 20%
```

Diviser la fenêtre

```
> par(mfrow=c(2, 2)) # divise la fenêtre en 4 : 2
lignes et 2 colonnes
```

Nuages de points — I

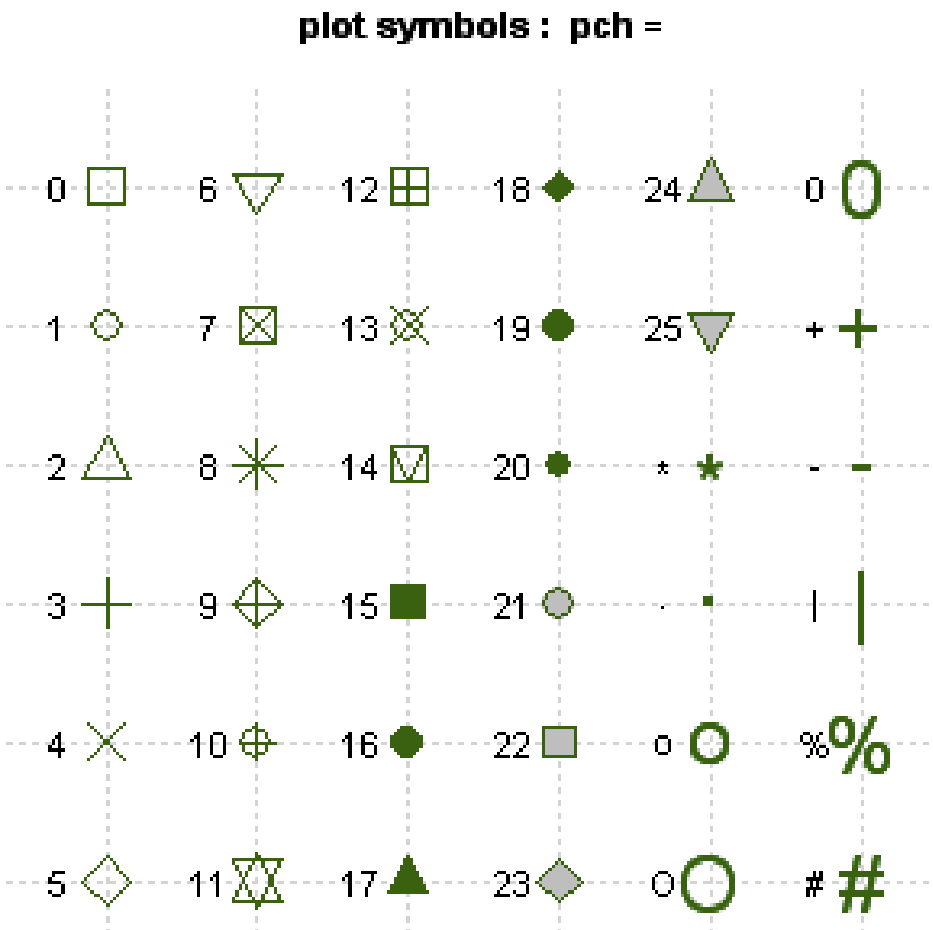
Pour croiser deux variables quantitatives et observer leur liaison, un nuage de points peut être réalisé à l'aide de la commande `plot` :

Nuage de points « simple »

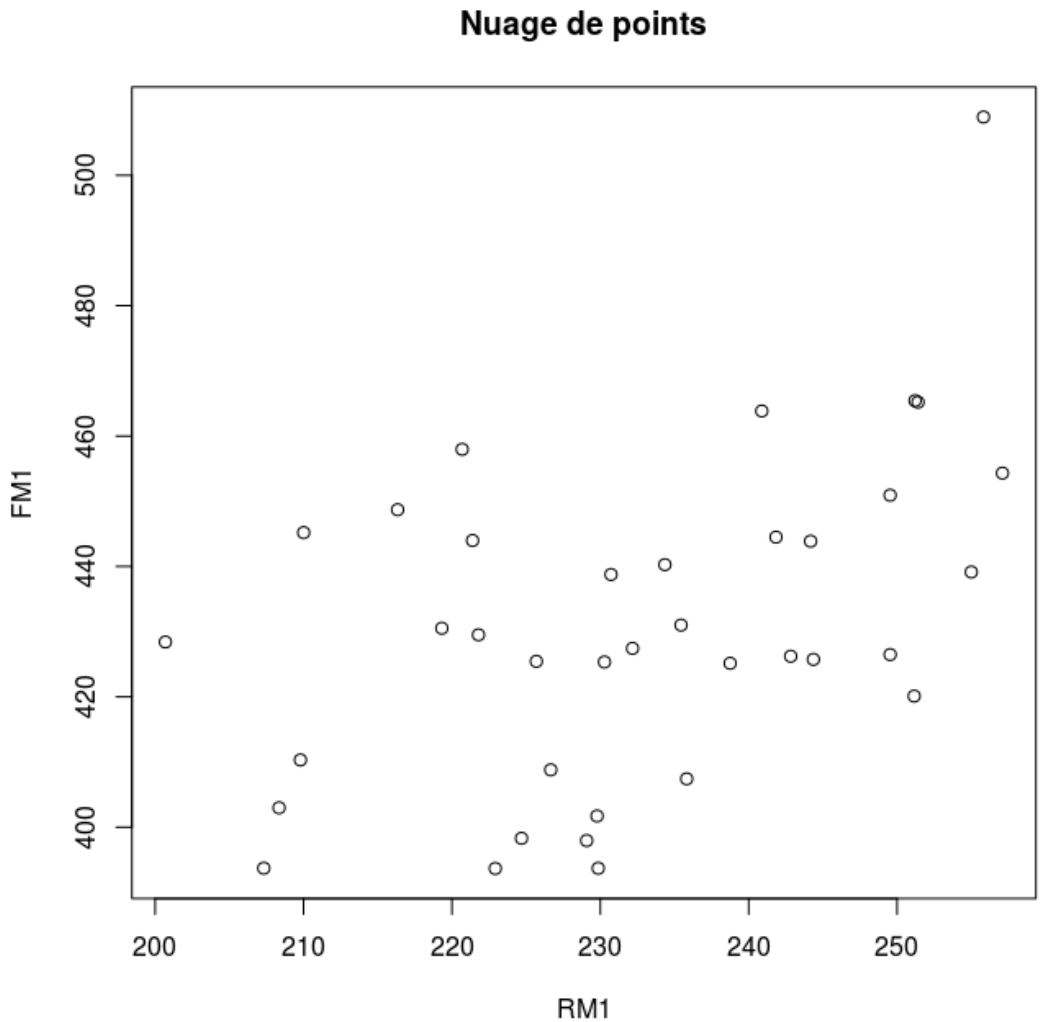
```
> plot(FM1 ~ RM1, data=dat, main="Nuage de points",
pch=1)
```

L'argument `pch` permet de préciser le type de figuré souhaité, et l'argument `main` d'ajouter un titre au graphique.

Nuages de points — II



Nuages de points — III



Nuages de points conditionnels — I

Une solution est d'utiliser le package `lattice` :

La commande `xyplot`

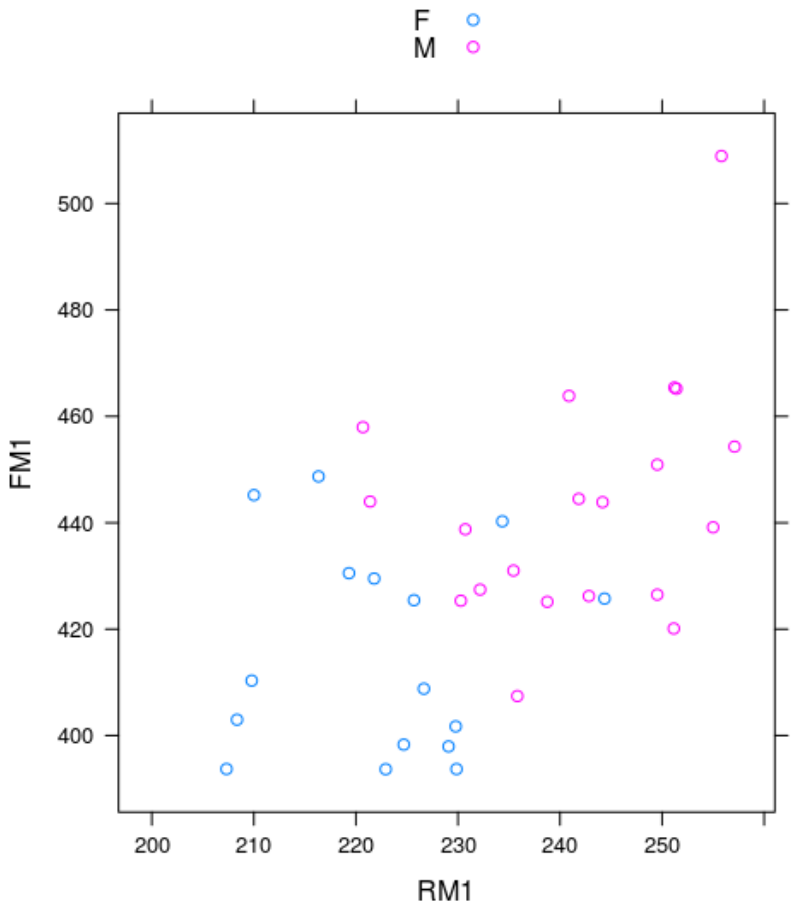
```
> xyplot(FM1 ~ RM1, groups=Sexe, auto.key=TRUE,
data=dat)
```

L'argument `auto.key = TRUE` est utilisé pour afficher automatiquement une légende. On peut la personnaliser (cf. l'aide).

Remarque : on pourrait aussi demander un graphique séparé pour les hommes et les femmes avec la commande suivante (la même échelle est conservée pour les deux graphiques afin de faciliter l'interprétation) :

```
> xyplot(FM1 ~ RM1 | Sexe, auto.key=TRUE, data=dat)
```

Nuages de points conditionnels — II



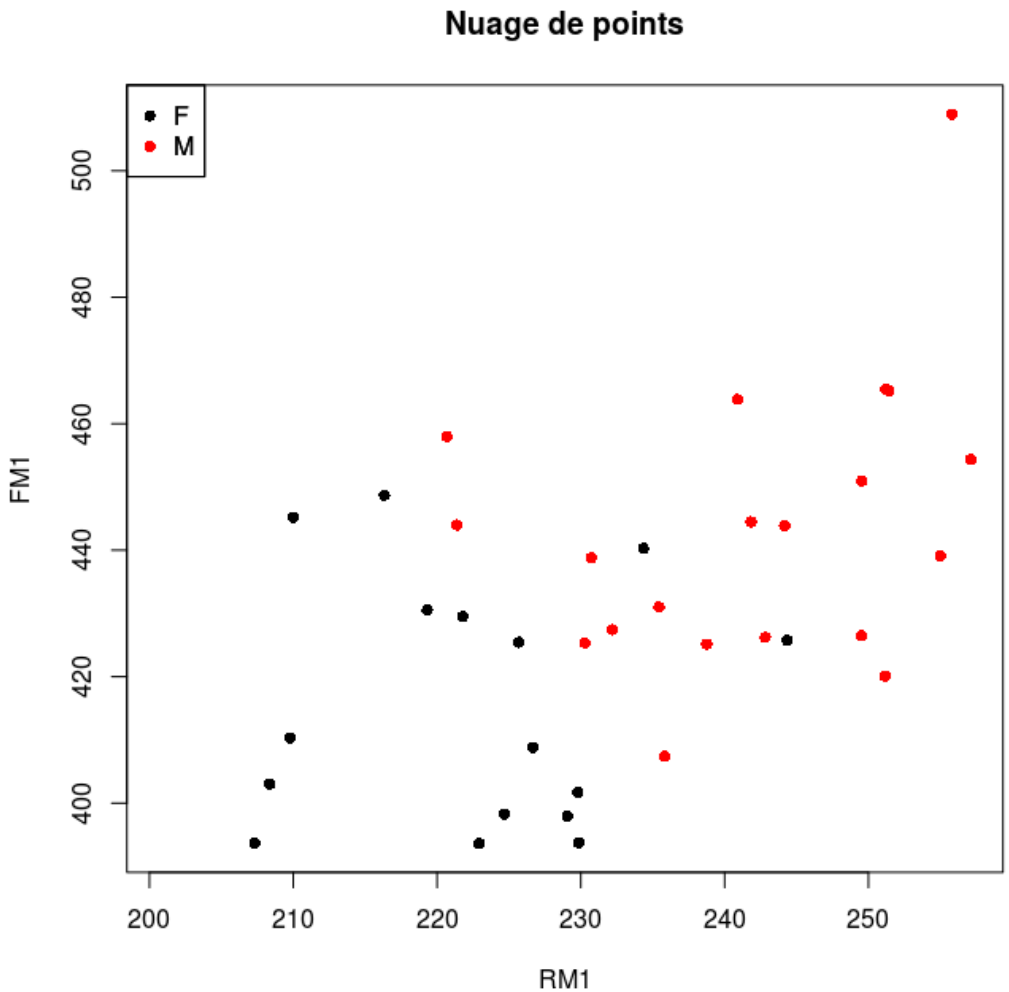
Nuages de points conditionnels — III

Toutefois, le code suivant permet d'obtenir le même genre de graphique à l'aide de la basique fonction `plot` de R :

Nuage de points conditionnel avec la fonction `plot`

```
> plot(FM1 ~ RM1, data=dat, pch=16,
col=as.numeric(dat$Sexe), main="Nuage de points")
> legend("topleft", pch=16, col=1:2,
legend=levels(dat$Sexe))
```

Nuages de points conditionnels — IV



La commande `jitter`

Dans la pratique, il arrive fréquemment que deux (ou parfois beaucoup plus !) points se superposent sur un graphique, car les individus associés prennent exactement les mêmes valeurs.

Lorsque cela constitue une gêne, la commande `jitter` apporte une solution : elle ajoute un très léger « bruit » dans les données afin que les points ne se superposent pas parfaitement.

Taper `help(jitter)` pour un exemple d'utilisation et plus d'information.

Ajout aux graphiques — I

La commande `plot` ouvre systématiquement une nouvelle fenêtre et y trace un graphiques. D'autres commandes permettent en revanche d'ajouter de nouveaux éléments sur la fenêtre graphique active.

1. La commande `abline` permet de tracer sur le graphique toute droite d'équation $y = a + bx$ ou $x = c$. Sur le graphique précédent, tester l'effet des commandes suivantes :

```
> abline(v=200, col="green") # droite verticale
> abline(h=440, col="blue", lty=2) # horiz.
> abline(a=0, b=2, lty=3) # trace la droite y=2x
```

Ajout aux graphiques — II

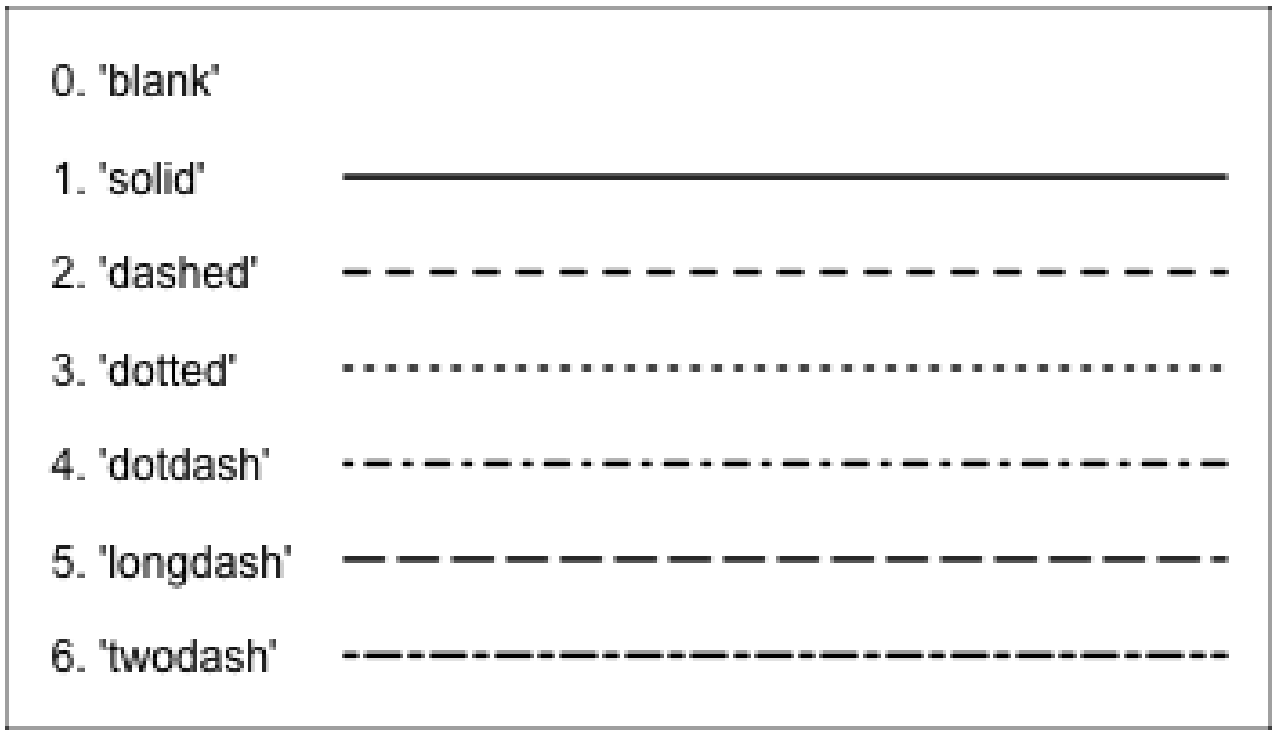


FIGURE : Types de lignes en R (argument `lty` de la commande `abline` ou `lines`).

Ajout aux graphiques — III

2. La commande `lines` permet de tracer sur le graphique toute ligne brisée passant par des points dont on spécifie les coordonnées.
   Tester la commande suivante :
   ```
   > lines(x=c(210,230,240), y=c(420,450,440),
   col="purple")
   ```
3. La commande `points` permet de rajouter un ou des points sur un graphique déjà existant, en spécifiant leurs coordonnées.
   Tester la commande suivante :
   ```
   > points(x=230, y=500, pch=8, cex=2)
   ```

Ajout aux graphiques — IV

4. La commande `text` permet d'ajouter du texte sur un graphique déjà existant, par exemple pour nommer les points sur le tracé.
   Tester les commandes suivantes :

```
> plot(FM1 ~ RM1, data=dat, pch=16,
col=as.numeric(dat$Sexe))
> text(x=dat$RM1, y=dat$FM1,
labels=rownames(dat), pos=3)
```

   L'argument `pos=3` signifie que les labels se positionneront au-dessus des points.

Exporter ses graphiques

R peut exporter vos graphiques en de très nombreux formats vectorisés ou non (.jpeg, .gif, .pdf, .eps, etc., et .emf dans le cas d'une utilisation sous Windows).

Sous Windows, l'export de graphique est très simple : il suffit de passer par le menu adéquat de la fenêtre graphique.

Sous Linux, il conviendra d'utiliser les commandes `jpeg`, `pdf`, etc., afin d'écrire directement dans un fichier le graphique que vous cherchez à produire. (Consulter alors l'aide de la fonction correspondant à votre format préféré.)

Graphiques en 3D

La library `rgl` est spécialisée dans les statistiques tridimensionnelles. Sa fonction `plot3d` offre la possibilité de générer des graphiques pour visualiser des données dans l'espace.

La commande `plot3d` : nuage de points (conditionnel) en 3D

```
> library(rgl)
> plot3d(x=dat$FM1, y=dat$RM1, z=dat$RM6, type="s",
col=as.numeric(dat$Sexe))
```

L'argument `type="s"` indique que l'on souhaite ici que les points soient représentés par de petites sphères. On peut changer le rayon de ces sphères avec l'argument `radius`.

Réaliser des tests de normalité — I

En de nombreuses situations (vérification des hypothèses requises en modélisation linéaire, réalisation de tests paramétriques, ...), il est nécessaire de vérifier la normalité des données avant de débuter les analyses.

Il existe beaucoup de tests de normalité mais le plus courant est celui de Shapiro-Wilk. Il est très facile d'utilisation sous R :

Tester la normalité d'une variable

```
> shapiro.test(dat$RM1)

        Shapiro-Wilk normality test

data: dat$RM1
W = 0.97451, p-value = 0.4777
```

Réaliser des tests de normalité — II

L'intérêt de R en tant que langage de programmation est aussi d'automatiser les tâches. On peut obtenir le résultat de tests de normalité pour toutes les variables numériques du jeu de données d'un seul coup, soit en programmant une boucle, soit en utilisant de petits artifices comme ci-dessous :

Tester la normalité de plusieurs variables

```
> norm = dat[ , c("RM1", "FM1", "TM6")]
> norm # uniquement les variables numeriques
> apply(norm, MARGIN=2, FUN=shapiro.test)
```

Réaliser des tests de normalité — III

En réalité, il est surtout nécessaire de tester la normalité d'une variable *au sein de chaque groupe d'individus* (par exemple pour les hommes puis pour les femmes).
Pour cela, on peut se servir (par exemple, ou sinon avec une boucle !) de la commande `tapply`.

Tester la normalité par groupe d'individus

```
> tapply(dat$FM1, INDEX=dat$Sexe, FUN=shapiro.test)
```

La commande ci-dessus lance la fonction `shapiro.test` sur la variable FM1, à l'intérieur de chaque groupe défini (« indexé ») par le facteur Sexe.

Test de Fisher (F-test) pour deux variances

La commande à utiliser est `var.test` :

Comparer les variances de deux groupes

```
> var.test(FM1 ~ Sexe, data=dat)

        F test to compare two variances

data: FM1 by Sexe
F = 0.69319, num df = 16, denom df = 20, p-value = 0.4611
alternative hypothesis: true ratio of variances is not
equal to 1
95 percent confidence interval:
0.2722086 1.8582994
sample estimates:
ratio of variances
0.6931902
```

Test de Student (t-test) pour deux échantillons indépendants

La commande à utiliser est `t.test` :

Comparer les moyennes de deux groupes

```
> t.test(FM1 ~ Sexe, var.equal=TRUE, data=dat)

                        Two Sample t-test

data: FM1 by Sexe
t = -4.3553, df = 36, p-value = 0.0001056
alternative hypothesis: true difference in means ≠ 0
95 percent confidence interval:
-45.15418 -16.46173
sample estimates:
mean in group F mean in group M
414.2482 445.0562
```

Test de corrélation de deux variables

La commande à utiliser est `cor.test` :

Test de significativité du coefficient de corrélation

```
> cor.test(dat$FM1, dat$RM1)

        Pearson's product-moment correlation

data: dat$FM1 and dat$RM1
t = 3.2213, df = 35, p-value = 0.002757
alternative hypothesis: true correlation ≠ 0
95 percent confidence interval:
0.1824562 0.6945972
sample estimates:
cor
0.4782041
```

Test de Wilcoxon pour échantillons indépendants

Si les conditions de normalité ne sont pas vérifiées, on doit appliquer des tests non paramétriques. Pour l'équivalent non paramétrique du test de Student, la commande à utiliser est `wilcox.test` :

Comparer les médianes de deux groupes

```
> wilcox.test(FM1 ~ Sexe, data=dat)

                    Wilcoxon rank sum test

data: FM1 by Sexe
W = 60, p-value = 0.0002933
alternative hypothesis: true location shift ≠ 0
```

Construire et analyser une table de contingence — I

Imaginons que l'on souhaite comparer la présence d'objets de parure dans l'échantillon de tombes fouillées en fonction du sexe des individus.

Construire une table de contingence

```
> contin <- table(dat$Sexe, dat$Presence_parure)
> contin

   Non Oui
F   7  12
M  19   4
```

Construire et analyser une table de contingence — II

Comme les effectifs bruts ne sont pas forcément parlants, on peut préférer une table donnant les distributions marginales en % :

La commande `prop.table` :

```
> prop.table(contin, margin=1)

         Non       Oui
  F 0.3684211 0.6315789
  M 0.8260870 0.1739130

> prop.table(contin, margin=2)

         Non       Oui
  F 0.2692308 0.7500000
  M 0.7307692 0.2500000
```

Construire et analyser une table de contingence — III

Il semble clair que les objets de parure sont associés aux tombes féminines. Pour vérifier cette hypothèse, le test adéquat est un χ^2 :

La commande `chisq.test` :

```
> chisq.test(contin)

  Pearson's Chi-squared test with
  Yates' continuity correction

data:  contin
X-squared = 7.4025, df = 1, p-value = 0.006513
```

L'association entre le sexe des individus et la présence de parure est bien « significative ».

Construire et analyser une table de contingence — IV

En cas de faibles effectifs (ce qui n'est pas le cas ici), réaliser un test exact de Fisher sera préférable :

La commande `fisher.test` :

```
> fisher.test(contin)

  Fisher's Exact Test for Count Data

data:  contin
p-value = 0.003883
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.02236343 0.60861101
sample estimates:
odds ratio
 0.1302994
```

Calcul du nombre de sujets nécessaire

R possède, via la library `pwr`, une série d'outils de calcul de puissance des tests statistiques usuels, permettant notamment de déterminer le nombre de sujets nécessaires pour atteindre une puissance de test suffisante.

Calcul du nombre de sujets pour un test de Student

```
> library(pwr)
> power.t.test(power=0.8, delta=3, sd=2.5)

     Two-sample t test power calculation
                      n = 11.94228
```

Il faudra donc 12 sujets dans chaque groupe pour pouvoir détecter avec une puissance suffisante une différence égale à 3 entre les moyennes des deux groupes d'une étude (pour $\sigma = 2.5$).