

Introduction à R pour le cours : Données longitudinales et modèles de survie

Nicolas S. Müller

Département des sciences économiques, Université de Genève

Semestre de printemps 2011

Plan

- 1 Introduction
- 2 Bases
- 3 Un cas concret : les primates
- 4 Statistiques descriptives
- 5 Modélisation

Plan

- 1 Introduction
- 2 Bases
- 3 Un cas concret : les primates
- 4 Statistiques descriptives
- 5 Modélisation

R

R est ...

- un environnement statistique,
- gratuit,
- disponible pour Windows, Mac et Linux/Unix.

Ressources indispensables

- Le CRAN : <http://cran.r-project.org>
- rseek : <http://www.rseek.org>
- Le site du prof : <http://www.andreberchtold.com>
- Le site de l'assistant :
<http://www.unige.ch/ses/metri/assistants/muller>

Installer R

Pour Windows, il suffit d'aller à cette adresse :

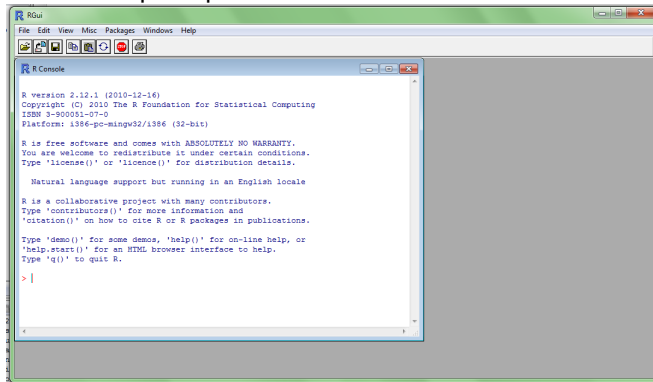
<http://cran.r-project.org/bin/windows/base/>

Pour Mac :

<http://cran.r-project.org/bin/macosx/>

Premiers pas...

- La fenêtre principale de R est la console :



```
RGui
File Edit View Misc Packages Windows Help
R Console
R version 2.12.1 (2010-12-16)
Copyright (C) 2010 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-pc-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

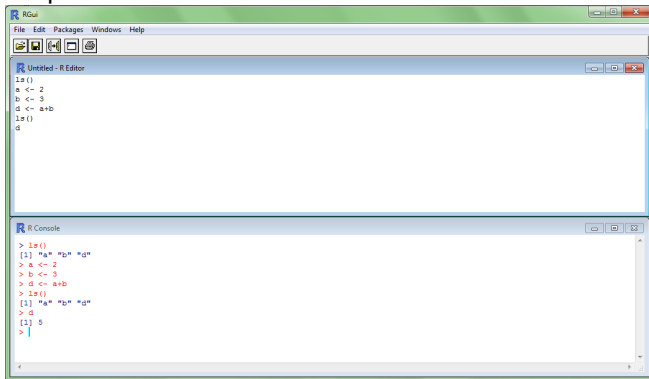
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

- C'est dans cette fenêtre qu'on tape les commandes.

Scripts

- Le meilleur moyen d'utiliser R est de passer par un fichier de script :



The screenshot shows the R GUI interface. The top window is titled "Untitled - R Editor" and contains the following R code:

```
ls()  
a <- 2  
b <- 3  
d <- a+b  
ls()  
d
```

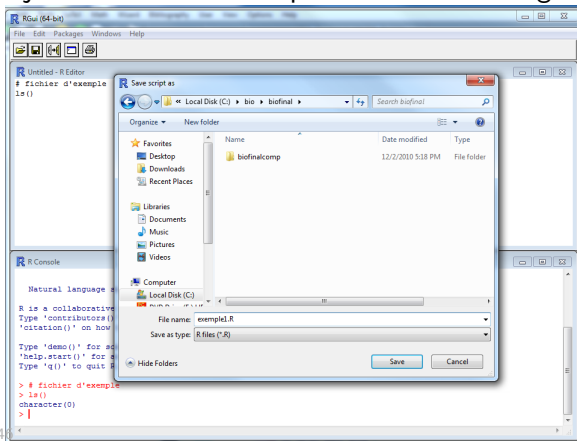
The bottom window is titled "R Console" and shows the execution of the code:

```
> ls()  
[1] "a" "b" "d"  
> a <- 2  
> b <- 3  
> d <- a+b  
> ls()  
[1] "a" "b" "d"  
> d  
[1] 5  
> |
```

- On écrit les commandes, on les surligne et on tape Ctrl+R pour les exécuter.

Sauver les scripts

- Sauvegardez tous vos scripts (ils vous seront demandé comme annexe de votre projet!) !
- Ajoutez l'extension .R quand vous les sauvegardez.



Paquets supplémentaires

- R est un logiciel modulaire, beaucoup de paquets externes peuvent être installés pour ajouter de nouvelles fonctionnalités.
- Pour installer un paquet, on utilise la commande `install.packages("nom_du_paquet")`.
- Les paquets que nous utiliserons sont généralement déjà installés.
- Pour charger un paquet installé, on utilise la commande `library(nom_du_paquet)` (!!pas de guillemets!!)

Paquets supplémentaires

- R est un logiciel modulaire, beaucoup de paquets externes peuvent être installés pour ajouter de nouvelles fonctionnalités.
- Pour installer un paquet, on utilise la commande `install.packages("nom_du_paquet")`.
- Les paquets que nous utiliserons sont généralement déjà installés.
- Pour charger un paquet installé, on utilise la commande `library(nom_du_paquet)` (!!pas de guillemets!!)

Paquets supplémentaires

- R est un logiciel modulaire, beaucoup de paquets externes peuvent être installés pour ajouter de nouvelles fonctionnalités.
- Pour installer un paquet, on utilise la commande `install.packages("nom_du_paquet")`.
- Les paquets que nous utiliserons sont généralement déjà installés.
- Pour charger un paquet installé, on utilise la commande `library(nom_du_paquet)` (!!pas de guillemets!!)

Paquets supplémentaires

- R est un logiciel modulaire, beaucoup de paquets externes peuvent être installés pour ajouter de nouvelles fonctionnalités.
- Pour installer un paquet, on utilise la commande `install.packages("nom_du_paquet")`.
- Les paquets que nous utiliserons sont généralement déjà installés.
- Pour charger un paquet installé, on utilise la commande `library(nom_du_paquet)` (!!pas de guillemets!!)

Plan

- 1 Introduction
- 2 Bases**
- 3 Un cas concret : les primates
- 4 Statistiques descriptives
- 5 Modélisation

Environnement

- L'environnement de travail contient des objets.
- Un objet peut être :
 - un nombre, un caractère, un booléen (TRUE/FALSE)
 - un vecteur de nombres, de caractères ou de booléens,
 - un tableau de nombres (une matrice),
 - un tableau de données (un dataframe),
 - une fonction ...

Environnement

- L'environnement de travail contient des objets.
- Un objet peut être :
 - un nombre, un caractère, un booléen (TRUE/FALSE)
 - un vecteur de nombres, de caractères ou de booléens,
 - un tableau de nombres (une matrice),
 - un tableau de données (un dataframe),
 - une fonction ...

Environnement

- L'environnement de travail contient des objets.
- Un objet peut être :
 - un nombre, un caractère, un booléen (TRUE/FALSE)
 - un vecteur de nombres, de caractères ou de booléens,
 - un tableau de nombres (une matrice),
 - un tableau de données (un dataframe),
 - une fonction ...

Environnement

- L'environnement de travail contient des objets.
- Un objet peut être :
 - un nombre, un caractère, un booléen (TRUE/FALSE)
 - un vecteur de nombres, de caractères ou de booléens,
 - un tableau de nombres (une matrice),
 - un tableau de données (un dataframe),
 - une fonction ...

Environnement

- L'environnement de travail contient des objets.
- Un objet peut être :
 - un nombre, un caractère, un booléen (TRUE/FALSE)
 - un vecteur de nombres, de caractères ou de booléens,
 - un tableau de nombres (une matrice),
 - un tableau de données (un dataframe),
 - une fonction ...

Environnement

- L'environnement de travail contient des objets.
- Un objet peut être :
 - un nombre, un caractère, un booléen (TRUE/FALSE)
 - un vecteur de nombres, de caractères ou de booléens,
 - un tableau de nombres (une matrice),
 - un tableau de données (un dataframe),
 - une fonction ...

Opérateurs

- Les opérateurs permettent d'effectuer des opérations sur des nombres ou des objets.
- Opérateurs courants : $+$, $-$, $/$, $*$
- Opérateur spécial : l'opérateur d'assignation \leftarrow :
a \leftarrow 2 assigne la valeur 2 à l'objet a.
- Si l'objet n'existe pas, il est créé; s'il existe, il est écrasé!

Opérateurs

- Les opérateurs permettent d'effectuer des opérations sur des nombres ou des objets.
- Opérateurs courants : $+$, $-$, $/$, $*$
- Opérateur spécial : l'opérateur d'assignation \leftarrow :
a \leftarrow 2 assigne la valeur 2 à l'objet a.
- Si l'objet n'existe pas, il est créé; s'il existe, il est écrasé!

Opérateurs

- Les opérateurs permettent d'effectuer des opérations sur des nombres ou des objets.
- Opérateurs courants : $+$, $-$, $/$, $*$
- Opérateur spécial : l'opérateur d'assignation \leftarrow :
a \leftarrow 2 assigne la valeur 2 à l'objet a.
- Si l'objet n'existe pas, il est créé; s'il existe, il est écrasé!

Opérateurs

- Les opérateurs permettent d'effectuer des opérations sur des nombres ou des objets.
- Opérateurs courants : $+$, $-$, $/$, $*$
- Opérateur spécial : l'opérateur d'assignation \leftarrow :
a \leftarrow 2 assigne la valeur 2 à l'objet a.
- Si l'objet n'existe pas, il est créé; s'il existe, il est écrasé!

Opérateurs

- Les opérateurs permettent d'effectuer des opérations sur des nombres ou des objets.
- Opérateurs courants : $+$, $-$, $/$, $*$
- Opérateur spécial : l'opérateur d'assignation \leftarrow :
a \leftarrow 2 assigne la valeur 2 à l'objet a.
- Si l'objet n'existe pas, il est créé; s'il existe, il est écrasé!

Exemple objets

```
> a <- 2
> b <- 3
> a + b
[1] 5
> somme <- a+b
> somme
[1] 5
> vec1 <- c(3,6,3,2,19)
> vec1
[1] 3 6 3 2 19
> vec1*2
[1] 6 12 6 4 38
```

Les fonctions (1/3)

- Une fonction exécute des opérations sur un objet passé en argument.
- Les arguments ont un nom, mais les noms peuvent être omis quand il n'y a pas d'ambiguïté.
- Les arguments sont toujours séparés par une virgule.
- Par exemple, la fonction `c(...)` crée un vecteur à partir de ses arguments.
- Autre exemple, la fonction `sum(..., na.rm=FALSE)` :
- `sum(1,3,5)` additionne 1, 3 et 5.
- `sum(d)` additionne les éléments de l'objet `d`.

Les fonctions (1/3)

- Une fonction exécute des opérations sur un objet passé en argument.
- Les arguments ont un nom, mais les noms peuvent être omis quand il n'y a pas d'ambiguïté.
- Les arguments sont toujours séparés par une virgule.
- Par exemple, la fonction `c(...)` crée un vecteur à partir de ses arguments.
- Autre exemple, la fonction `sum(..., na.rm=FALSE)` :
- `sum(1,3,5)` additionne 1, 3 et 5.
- `sum(d)` additionne les éléments de l'objet `d`.

Les fonctions (1/3)

- Une fonction exécute des opérations sur un objet passé en argument.
- Les arguments ont un nom, mais les noms peuvent être omis quand il n'y a pas d'ambiguïté.
- Les arguments sont toujours séparés par une virgule.
- Par exemple, la fonction `c(...)` crée un vecteur à partir de ses arguments.
- Autre exemple, la fonction `sum(..., na.rm=FALSE)` :
- `sum(1,3,5)` additionne 1, 3 et 5.
- `sum(d)` additionne les éléments de l'objet `d`.

Les fonctions (1/3)

- Une fonction exécute des opérations sur un objet passé en argument.
- Les arguments ont un nom, mais les noms peuvent être omis quand il n'y a pas d'ambiguïté.
- Les arguments sont toujours séparés par une virgule.
- Par exemple, la fonction `c(...)` crée un vecteur à partir de ses arguments.
- Autre exemple, la fonction `sum(...,na.rm=FALSE)` :
- `sum(1,3,5)` additionne 1, 3 et 5.
- `sum(d)` additionne les éléments de l'objet `d`.

Les fonctions (1/3)

- Une fonction exécute des opérations sur un objet passé en argument.
- Les arguments ont un nom, mais les noms peuvent être omis quand il n'y a pas d'ambiguïté.
- Les arguments sont toujours séparés par une virgule.
- Par exemple, la fonction `c(...)` crée un vecteur à partir de ses arguments.
- Autre exemple, la fonction `sum(..., na.rm=FALSE)` :
- `sum(1,3,5)` additionne 1, 3 et 5.
- `sum(d)` additionne les éléments de l'objet `d`.

Les fonctions (2/3)

- La plupart du temps, une fonction renvoie elle-même un objet.
Ex : `sum(d)` renvoie un nombre.
- On peut assigner le résultat à un objet :
Ex : `resultat <- sum(d)`

Les fonctions (2/3)

- La plupart du temps, une fonction renvoie elle-même un objet.
Ex : `sum(d)` renvoie un nombre.
- On peut assigner le résultat à un objet :
Ex : `resultat <- sum(d)`

Les fonctions (3/3)

- Certaines fonctions n'ont pas d'arguments, mais les parenthèses sont obligatoires!
- Ex : `ls()` donne la liste des objets de l'environnement courant.

Charger/sauvegarder

- R vous demande si vous voulez sauvegarder votre environnement en quittant.
- Environnement (workspace) \neq objet
- `save.image("c:/test.RData")` sauve l'environnement avec tous ces objets.
- `save(a, "c:/a.RData")` sauve l'objet 'a'.
- La commande `load(...)` charge indifféremment un objet ou un environnement.
- **Mais surtout, sauvegardez vos scripts !!!**

Charger/sauvegarder

- R vous demande si vous voulez sauvegarder votre environnement en quittant.
- Environnement (workspace) \neq objet
- `save.image("c:/test.RData")` sauve l'environnement avec tous ces objets.
- `save(a, "c:/a.RData")` sauve l'objet 'a'.
- La commande `load(...)` charge indifféremment un objet ou un environnement.
- **Mais surtout, sauvegardez vos scripts !!!**

Charger/sauvegarder

- R vous demande si vous voulez sauvegarder votre environnement en quittant.
- Environnement (workspace) \neq objet
- `save.image("c:/test.RData")` sauve l'environnement avec tous ces objets.
- `save(a, "c:/a.RData")` sauve l'objet 'a'.
- La commande `load(...)` charge indifféremment un objet ou un environnement.
- **Mais surtout, sauvegardez vos scripts !!!**

Charger/sauvegarder

- R vous demande si vous voulez sauvegarder votre environnement en quittant.
- Environnement (workspace) \neq objet
- `save.image("c:/test.RData")` sauve l'environnement avec tous ces objets.
- `save(a, "c:/a.RData")` sauve l'objet 'a'.
- La commande `load(...)` charge indifféremment un objet ou un environnement.
- **Mais surtout, sauvegardez vos scripts !!!**

Accéder à l'aide

- L'aide en ligne est indispensable.
- Une page d'aide pour chaque fonction est accessible par :
`?fonction`.
- La structure est toujours la même.
- Les manuels sont également une référence indispensable, n'hésitez pas à les consulter!

Lire une page d'aide

- **Description** : décrit la fonction succinctement.
- **Usage** : donne la syntaxe de la commande. Les arguments qui sont suivi d'un signe '=' et d'une valeur sont des arguments dont la valeur par défaut est déjà défini.
- **Arguments** : décrit l'utilité de chaque argument.
- **Details** : comme son nom l'indique...
- **Value** : décrit ce que la fonction renvoie comme valeur.
- **Examples** : une série d'exemples d'utilisation de la fonction. Tout ce qui commence par '# ' est un commentaire et n'est pas interprété.

Lire une page d'aide

- **Description** : décrit la fonction succinctement.
- **Usage** : donne la syntaxe de la commande. Les arguments qui sont suivi d'un signe '=' et d'une valeur sont des arguments dont la valeur par défaut est déjà défini.
- **Arguments** : décrit l'utilité de chaque argument.
- **Details** : comme son nom l'indique...
- **Value** : décrit ce que la fonction renvoie comme valeur.
- **Examples** : une série d'exemples d'utilisation de la fonction. Tout ce qui commence par '#' est un commentaire et n'est pas interprété.

Lire une page d'aide

- **Description** : décrit la fonction succinctement.
- **Usage** : donne la syntaxe de la commande. Les arguments qui sont suivi d'un signe '=' et d'une valeur sont des arguments dont la valeur par défaut est déjà défini.
- **Arguments** : décrit l'utilité de chaque argument.
- **Details** : comme son nom l'indique...
- **Value** : décrit ce que la fonction renvoie comme valeur.
- **Examples** : une série d'exemples d'utilisation de la fonction. Tout ce qui commence par '#' est un commentaire et n'est pas interprété.

Lire une page d'aide

- **Description** : décrit la fonction succinctement.
- **Usage** : donne la syntaxe de la commande. Les arguments qui sont suivi d'un signe '=' et d'une valeur sont des arguments dont la valeur par défaut est déjà défini.
- **Arguments** : décrit l'utilité de chaque argument.
- **Details** : comme son nom l'indique...
- **Value** : décrit ce que la fonction renvoie comme valeur.
- **Examples** : une série d'exemples d'utilisation de la fonction. Tout ce qui commence par '#' est un commentaire et n'est pas interprété.

Lire une page d'aide

- **Description** : décrit la fonction succinctement.
- **Usage** : donne la syntaxe de la commande. Les arguments qui sont suivi d'un signe '=' et d'une valeur sont des arguments dont la valeur par défaut est déjà défini.
- **Arguments** : décrit l'utilité de chaque argument.
- **Details** : comme son nom l'indique...
- **Value** : décrit ce que la fonction renvoie comme valeur.
- **Examples** : une série d'exemples d'utilisation de la fonction. Tout ce qui commence par '#' est un commentaire et n'est pas interprété.

Lire une page d'aide

- **Description** : décrit la fonction succinctement.
- **Usage** : donne la syntaxe de la commande. Les arguments qui sont suivi d'un signe '=' et d'une valeur sont des arguments dont la valeur par défaut est déjà défini.
- **Arguments** : décrit l'utilité de chaque argument.
- **Details** : comme son nom l'indique...
- **Value** : décrit ce que la fonction renvoie comme valeur.
- **Examples** : une série d'exemples d'utilisation de la fonction. Tout ce qui commence par '#' est un commentaire et n'est pas interprété.

Plan

- 1 Introduction
- 2 Bases
- 3 Un cas concret : les primates**
- 4 Statistiques descriptives
- 5 Modélisation

Les données primate

- On a un fichier de données au format R :
`Monkey_introduction.rda`.
- Les variables sont les suivantes :
 - session : numéro de la session
 - temps : temps écoulé en secondes depuis le début de la session
 - age : âge en jour au moment de chaque session
 - age_3c : âge recodé en 3 catégories (1 : 41-100, 2 : 101-200, 3 : 201-261)
 - sujet : comportement du singe observé (1 à 4)
 - interacteur : comportement du singe en interaction avec le sujet observé
 - anormal : nombre de singes anormaux dans le groupe de socialisation
- On le charge avec la commande `load(...)`

Les données primate

- On a un fichier de données au format R :
`Monkey_introduction.rda`.
- Les variables sont les suivantes :
 - `session` : numéro de la session
 - `temps` : temps écoulé en secondes depuis le début de la session
 - `age` : âge en jour au moment de chaque session
 - `age_3c` : âge recodé en 3 catégories (1 : 41-100, 2 : 101-200, 3 : 201-261)
 - `sujet` : comportement du singe observé (1 à 4)
 - `interacteur` : comportement du singe en interaction avec le sujet observé
 - `anormal` : nombre de singes anormaux dans le groupe de socialisation
- On le charge avec la commande `load(...)`

Travailler avec un dataframe

- Un dataframe est un tableau de données, souvent organisé avec les variables en colonne et les observations en ligne.
- Ce tableau a deux dimensions, lignes et colonnes (comme une matrice).

session	temps	age	...
1	10	24	...
2	12	26	...
3	14	28	...
...

Commandes utiles pour les dataframes

- `names(dataframe)` donne le nom des colonnes.
- `nrow(dataframe)` donne le nombre de lignes.
- `head(dataframe)` affiche les 10 premières lignes.
- `summary(dataframe)` donne un résumé statistique de chaque variable.
- `fix(dataframe)` affiche le dataframe dans un tableur.

Indices pour les dataframes

- Le premier indice représente les lignes, le deuxième indice les colonnes.
- `Monkey [1,]` affiche la première ligne, toutes les colonnes.
- `Monkey [, 1]` affiche toutes les lignes, seulement la première colonne.
- `Monkey [1, 1]` affiche le premier élément de la première ligne.
- `Monkey [10:20,]` affiche les lignes 10 à 20.
- Si les colonnes ont des noms :
`Monkey [, "age"]` affiche la colonne "age".

Indices pour les vecteurs

- Un vecteur est considéré comme un dataframe à 1 dimension.
- On accède à ces éléments avec un seul indice.
- `age[1]` donne le premier élément du vecteur "age".
- `age[10:20]` donne les éléments 10 à 20 du vecteur "age".
- On peut créer un vecteur à partir de la colonne d'un dataframe :
 - `age <- Monkey[, "age"]`
 - `age <- Monkey$age`

Condition sur les indices

- Les indices peuvent faire l'objet de conditions.
- Trois opérateurs logiques principaux : $<$, $>$, $==$
- La négation est définie par ! (ex: $!=$ signifie 'différent de').
- Si l'on veut afficher les primates âgés de moins de 60 mois :
- `Monkey[Monkey[, "age"] < 60,]`
- On peut créer un nouveau dataframe avec les primates âgés de moins de 60 mois :
`MonkeyYoung <- Monkey[Monkey[, "age"] < 60,]`
ou : `MonkeyYoung <- Monkey[Monkey$age < 60,]`

Recodage de facteurs

- Dans R, on manipule surtout deux types de variables : numériques ou facteurs.
- Une variable "factor" est une variable catégorielle.
- La fonction `levels(nom_du_facteur)` a trois utilités :
 - Elle affiche les noms des catégories de la variable.
Ex : `levels(Monkey$age)`
 - Elle permet de les modifier (on attribue de nouvelles valeurs).
Ex : `levels(Monkey$age) <- c("jeune", "moyen", "vieux")`
 - Elle permet de regrouper des catégories.
Ex : `levels(Monkey$age) <- c("jeune", "jeune", "vieux")`

Recodage de variables numériques

- Pour une variable numérique, on peut recoder de différentes manières :
- En utilisant la fonction `cut(x, breaks=bornes)`

```
> age_3c2 <- cut(Monkey$age, breaks=3)
> summary(age_3c2)
(40.8,114]  (114,188]  (188,261]
          4354          3686          4867
```

Plan

- 1 Introduction
- 2 Bases
- 3 Un cas concret : les primates
- 4 Statistiques descriptives**
- 5 Modélisation

Outils de base

- Première chose, connaître ses variables.
- Suivant le type d'objet sur lequel s'applique une fonction, le résultat diffère!

```
> summary(Monkey$age)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
41.0	95.0	153.0	153.5	217.0	261.0

```
> summary(Monkey$age_3c)
```

<= 100 jours	101-200 jours	> 200 jours
3691	5275	3941

Moyenne, médiane, écart-type...

- moyenne : `mean(..., na.rm=FALSE)`
- médiane : `median(..., na.rm=FALSE)`
- écart-type : `sd(..., na.rm=FALSE)`
- variance ... ?

Données manquantes

- R attribue le code 'NA' lorsqu'une donnée est manquante.
- Sinon, NaN = Not a Number, $-\text{Inf}/+\text{Inf} = +/- \infty$
- **Attention** : NA n'est pas une chaîne de caractère mais un symbole spécial. Toute comparaison avec 'NA' doit se faire via la fonction `is.na(...)`.
- Une valeur n'est pas manquante quand `is.na(...)` renvoie FALSE.

Exemple données manquantes

```
> a <- NA
> a==NA
[1] NA
> a=="NA"
[1] NA
> is.na(a)
[1] TRUE
> !is.na(a)
[1] FALSE
```

Exemple données manquantes

```
> a <- c(1,4,3,NA,2)
> is.na(a)
[1] FALSE FALSE FALSE TRUE FALSE
> a[!is.na(a)]
[1] 1 4 3 2
> mean(a)
[1] NA
> mean(a, na.rm=TRUE)
[1] 2.5
```

Récupérer les outputs

- Tous les résultats sont en mode texte.
- Si le résultat est une table, on peut l'exporter p. ex. dans Excel.
- L'exemple suivant copie un tableau dans le presse-papiers Windows.

```
write.table(summary(Monkey$age_3c), "clipboard",  
sep="\t", row.names=FALSE)
```

Petit exercice...

- 1 Description des variables du fichier Monkey, sous forme de tableaux de fréquence ou de résumés statistiques, dans un fichier Word.
- 2 Création d'une nouvelle variable `age_4c` à partir de l'âge, quatre catégories : de 0 à 50, de 50 à 100, de 100 à 150, de 150 à 300.

Petit exercice...

- 1 Description des variables du fichier Monkey, sous forme de tableaux de fréquence ou de résumés statistiques, dans un fichier Word.
- 2 Création d'une nouvelle variable `age_4c` à partir de l'âge, quatre catégories : de 0 à 50, de 50 à 100, de 100 à 150, de 150 à 300.

Petit exercice...

- 1 Description des variables du fichier Monkey, sous forme de tableaux de fréquence ou de résumés statistiques, dans un fichier Word.
- 2 Création d'une nouvelle variable `age_4c` à partir de l'âge, quatre catégories : de 0 à 50, de 50 à 100, de 100 à 150, de 150 à 300.

Plan

- 1 Introduction
- 2 Bases
- 3 Un cas concret : les primates
- 4 Statistiques descriptives
- 5 **Modélisation**

Principe général

- Un modèle s'écrit comme une équation : $y = \beta_0 + \beta_1x + \epsilon$
- Dans R, on n'écrit que les variables, pas les coefficients :
 $y \sim x$.
- Une interaction peut s'écrire directement : $y \sim x + z + x * z$.
- Régression linéaire : `lm(formule)`
- Régression logistique : `glm(formule, family=binomial)`
- On place **toujours** le résultat dans un objet!

Principe général

- Un modèle s'écrit comme une équation : $y = \beta_0 + \beta_1x + \epsilon$
- Dans R, on n'écrit que les variables, pas les coefficients :
 $y \sim x$.
- Une interaction peut s'écrire directement : $y \sim x + z + x * z$.
- Régression linéaire : `lm(formule)`
- Régression logistique : `glm(formule, family=binomial)`
- On place **toujours** le résultat dans un objet!

Principe général

- Un modèle s'écrit comme une équation : $y = \beta_0 + \beta_1x + \epsilon$
- Dans R, on n'écrit que les variables, pas les coefficients :
 $y \sim x$.
- Une interaction peut s'écrire directement : $y \sim x + z + x * z$.
- Régression linéaire : `lm(formule)`
- Régression logistique : `glm(formule, family=binomial)`
- On place **toujours** le résultat dans un objet!

Principe général

- Un modèle s'écrit comme une équation : $y = \beta_0 + \beta_1 x + \epsilon$
- Dans R, on n'écrit que les variables, pas les coefficients :
 $y \sim x$.
- Une interaction peut s'écrire directement : $y \sim x + z + x * z$.
- Régression linéaire : `lm(formule)`
- Régression logistique : `glm(formule, family=binomial)`
- On place **toujours** le résultat dans un objet!

Principe général

- Un modèle s'écrit comme une équation : $y = \beta_0 + \beta_1 x + \epsilon$
- Dans R, on n'écrit que les variables, pas les coefficients :
 $y \sim x$.
- Une interaction peut s'écrire directement : $y \sim x + z + x * z$.
- Régression linéaire : `lm(formule)`
- Régression logistique : `glm(formule, family=binomial)`
- On place **toujours** le résultat dans un objet!

Principe général

- Un modèle s'écrit comme une équation : $y = \beta_0 + \beta_1x + \epsilon$
- Dans R, on n'écrit que les variables, pas les coefficients :
 $y \sim x$.
- Une interaction peut s'écrire directement : $y \sim x + z + x * z$.
- Régression linéaire : `lm(formule)`
- Régression logistique : `glm(formule, family=binomial)`
- On place **toujours** le résultat dans un objet!

Exemple : régression linéaire

- Charger données "swiss" avec `data(swiss)`.
- Régression : `mod1 <- lm(Fertility ~ Catholic, data=swiss)`
- On examine les résultats : `summary(mod1)`

Exemple : régression linéaire

- Charger données "swiss" avec `data(swiss)`.
- Régression : `mod1 <- lm(Fertility ~ Catholic, data=swiss)`
- On examine les résultats : `summary(mod1)`

Exemple : régression linéaire

- Charger données "swiss" avec `data(swiss)`.
- Régression : `mod1 <- lm(Fertility ~ Catholic, data=swiss)`
- On examine les résultats : `summary(mod1)`

summary(mod1)

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	64.42826	2.30510	27.950	< 2e-16	***
Catholic	0.13889	0.03956	3.511	0.00103	**

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11.19 on 45 degrees of freedom

Multiple R-squared: 0.215, Adjusted R-squared: 0.1976

F-statistic: 12.33 on 1 and 45 DF, p-value: 0.001029

Deuxième régression

- `mod2 <- lm(Fertility ~ Catholic + Education, data=swiss)`
- Comparaison : `anova(mod1,mod2)`

summary(mod2)

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	74.23369	2.35197	31.562	< 2e-16	***
Catholic	0.11092	0.02981	3.721	0.00056	***
Education	-0.78833	0.12929	-6.097	2.43e-07	***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.331 on 44 degrees of freedom

Multiple R-squared: 0.5745, Adjusted R-squared: 0.5552

F-statistic: 29.7 on 2 and 44 DF, p-value: 6.849e-09

Comparaison mod1 et mod2

Analysis of Variance Table

Model 1: Fertility ~ Catholic

Model 2: Fertility ~ Catholic + Education

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	45	5634.7				
2	44	3054.2	1	2580.5	37.176	2.428e-07 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Régression logistique

- En premier lieu, on crée une variable dichotomique :

```
FertDicho <- cut(swiss$Fertility,  
breaks=c(0,70,100))
```
- On ajuste deux modèles linéaires généralisés avec une fonction de lien binomial (logit) :

```
mod1logit <- glm(FertDicho ~ swiss$Catholic,  
family=binomial, data=swiss)  
mod2logit <- glm(FertDicho ~ swiss$Education,  
family=binomial, data=swiss)
```
- Test du rapport de vraisemblance :

```
anova(modlog1,  
modlog2, test="Chisq")
```

Régression logistique

- En premier lieu, on crée une variable dichotomique :

```
FertDicho <- cut(swiss$Fertility,  
breaks=c(0,70,100))
```
- On ajuste deux modèles linéaires généralisés avec une fonction de lien binomial (logit) :

```
mod1logit <- glm(FertDicho ~ swiss$Catholic,  
family=binomial, data=swiss)  
mod2logit <- glm(FertDicho ~ swiss$Education,  
family=binomial, data=swiss)
```
- Test du rapport de vraisemblance :

```
anova(modlog1,  
modlog2, test="Chisq")
```

Régression logistique

- En premier lieu, on crée une variable dichotomique :

```
FertDicho <- cut(swiss$Fertility,  
breaks=c(0,70,100))
```
- On ajuste deux modèles linéaires généralisés avec une fonction de lien binomial (logit) :

```
mod1logit <- glm(FertDicho ~ swiss$Catholic,  
family=binomial, data=swiss)  
mod2logit <- glm(FertDicho ~ swiss$Education,  
family=binomial, data=swiss)
```
- Test du rapport de vraisemblance :

```
anova(modlog1,  
modlog2, test="Chisq")
```

`anova(modlog1,modlog, test="Chisq")`

Analysis of Deviance Table

Model 1: `FertDich ~ Catholic`

Model 2: `FertDich ~ Catholic + Education`

	Resid. Df	Resid. Dev	Df	Deviance
1	45	55.051		
2	44	48.262	1	6.7891