



# Rapport sur le code R et son exécution

Mehdi Bahi 22014246 (G.I)



Professor: Phd. Aniss Moumen

École Nationale des Sciences Appliquées Kénitra  
Année Académique 2024-2025



# Contents

<b>1</b>	<b><i>Exemples de concepts de R Base</i></b>	<b>3</b>
1.1	Déclaration de variables . . . . .	3
1.1.1	Affichage dans l'environnement . . . . .	3
1.1.2	Opérateur typeof() . . . . .	4
1.1.3	Fonction as.integer() . . . . .	4
1.1.4	Déclaration d'un nombre flottant . . . . .	4
1.1.5	Variable de type complexe . . . . .	5
1.2	Représentation graphique d'une variable . . . . .	5
1.3	Comparaison logique . . . . .	6
1.3.1	Déclaration d'une variable logique . . . . .	7
1.3.2	Fonction is.logical() . . . . .	7
1.4	Déclaration d'un caractère . . . . .	7
1.4.1	Fonction is.character() . . . . .	8
1.4.2	Déclaration d'une variable NA et d'une variable NULL & méthode is.null() . . . . .	8
1.5	Déclaration d'un vecteur . . . . .	9
1.5.1	Fonction is.na() . . . . .	9
1.5.2	Déclaration d'un vecteur avec des variables de type brut (as.raw() fonction) . . . . .	9
1.6	Fonction mode() . . . . .	10
1.7	Tentative de calcul de la moyenne . . . . .	10
1.7.1	Exclusion des données manquantes pour calculer la moyenne . . . .	11
1.8	Déclaration d'une chaîne de caractères . . . . .	11
1.8.1	Fonction as.character() . . . . .	12
1.8.2	Fonction as.numeric() . . . . .	12
1.8.3	Fonction as.integer() . . . . .	12
1.9	Conversion automatique . . . . .	12
1.10	Conversion impossible . . . . .	13
1.11	Vecteurs (Suite.) . . . . .	13
1.11.1	Priorité des valeurs dans les vecteurs . . . . .	13
1.11.2	Fonction seq() . . . . .	13
1.11.3	Déclaration d'un vecteur avec des compréhensions . . . . .	14
1.11.4	Fonction is.vector() . . . . .	14
1.11.5	Déclaration d'un vecteur avec des compréhensions . . . . .	15
1.11.6	Déclaration d'un vecteur avec la fonction c() . . . . .	15
1.11.7	Fonction class() . . . . .	15
1.12	Exemples de Matrices et Tableaux . . . . .	16
1.12.1	Déclaration d'une matrice avec la fonction matrix() . . . . .	16
1.12.2	Déclaration d'un tableau avec la fonction array() . . . . .	18



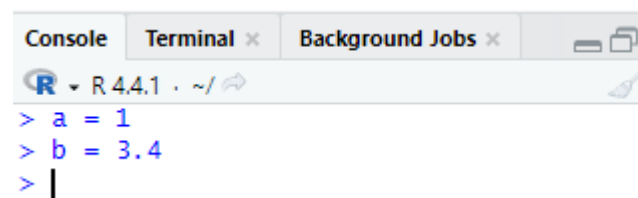
1.13	Exemple de Listes . . . . .	Mehdi Badi
1.13.1	Déclaration d'une liste avec la fonction <code>list()</code> . . . . .	19
1.14	Exemple de Facteurs . . . . .	20
1.14.1	Déclaration d'un facteur avec la fonction <code>factor()</code> . . . . .	20
1.14.2	Déclaration d'un facteur ordonné avec <code>ordered(c(), levels=c())</code> . . . . .	20
1.15	Exemple de Data Frames . . . . .	21
1.15.1	Déclaration d'un data frame avec la fonction <code>data.frame()</code> . . . . .	21
<b>2</b>	<b>Exemple de Pré-Traitement</b>	<b>22</b>
2.1	Étape 1: Définition de la problématique et des données . . . . .	22
2.1.1	Question de recherche . . . . .	22
2.2	Étape 2: Collecte des données . . . . .	22
2.2.1	Collecte manuelle (d1) - Saisie des données . . . . .	22
2.2.2	Collecte à partir d'un fichier (d2) . . . . .	22
2.3	Étape 3: Pré-traitement . . . . .	23
2.3.1	Codification et conversion . . . . .	23
2.3.2	Nettoyage des données . . . . .	24
<b>3</b>	<b>Exemple de Pré-traitement et Traitement</b>	<b>25</b>
3.1	Étape 1 : Collecte de données . . . . .	25
3.2	Étape 2 : Pré-traitement . . . . .	25
3.2.1	Conversion et recodification . . . . .	25
3.2.2	Traitement des valeurs aberrantes/manquantes . . . . .	27
3.2.3	Conversion en entier pour <code>Scolarité</code> . . . . .	29
3.2.4	Test de normalité des variables quantitatives . . . . .	29
3.2.5	Quasi-normalité . . . . .	29
3.3	Étape 3 : Traitement . . . . .	31
3.3.1	Statistique univariée . . . . .	31
3.3.2	La vérification du revenu . . . . .	33
3.4	Calcul de alpha . . . . .	34



# Chapter 1

## *Exemples de concepts de R Base*

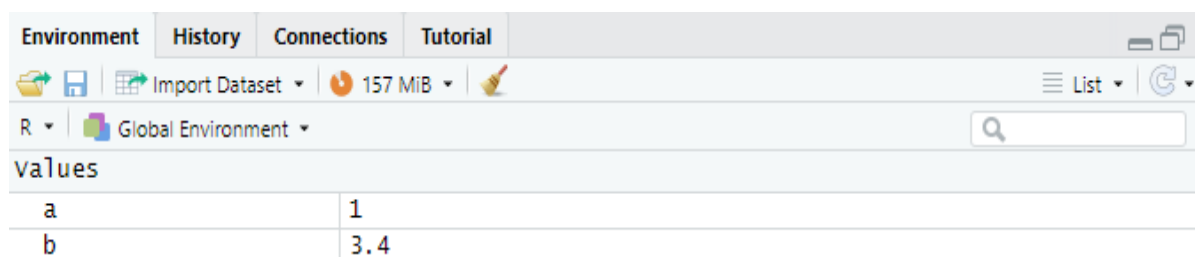
### 1.1 Déclaration de variables



```
Console Terminal x Background Jobs x  
R 4.4.1 ~/  
> a = 1  
> b = 3.4  
> |
```

Figure 1.1: Déclaration des variables

#### 1.1.1 Affichage dans l'environnement



values	
a	1
b	3.4

Figure 1.2: Affichage des variables dans l'environnement



### 1.1.2 Opérateur typeof()

Mehdi Bahi

```
> typeof(a)
[1] "double"
> typeof(b)
[1] "double"
> |
```

Figure 1.3: Opérateur \*typeof\* pour trouver le type d'une variable en R

Lorsque le type d'une variable n'est pas explicitement déclaré et qu'il s'agit d'un nombre, son type est toujours un *double* par défaut.

### 1.1.3 Fonction as.integer()

```
> c = as.integer(b)
> c
[1] 3
```

Figure 1.4: Déclaration de la variable C avec \*as.integer\* d'une autre variable entière

c		3
---	--	---

Figure 1.5: Affichage de la valeur de C dans l'environnement

### 1.1.4 Déclaration d'un nombre flottant

```
> c = 4.5
>
```

Figure 1.6: Déclaration d'un nombre flottant

c		4.5
---	--	-----

Figure 1.7: Affichage du nombre flottant déclaré



### 1.1.5 Variable de type complexe

Mehdi Bahi

```
> x <- 1 + 2i
> Re(x)
[1] 1
> Im(x)
[1] 2
>
```

Figure 1.8: Déclaration d'une variable de type complexe

La fonction `Re()` affiche la partie réelle de la variable complexe  $x$ .

La fonction `Im()` affiche la partie imaginaire de la variable complexe  $x$ .

```
x      | 1+2i
```

Figure 1.9: Affichage de la variable complexe

## 1.2 Représentation graphique d'une variable

```
> plot(x)
>
```

Figure 1.10: Fonction `plot()`

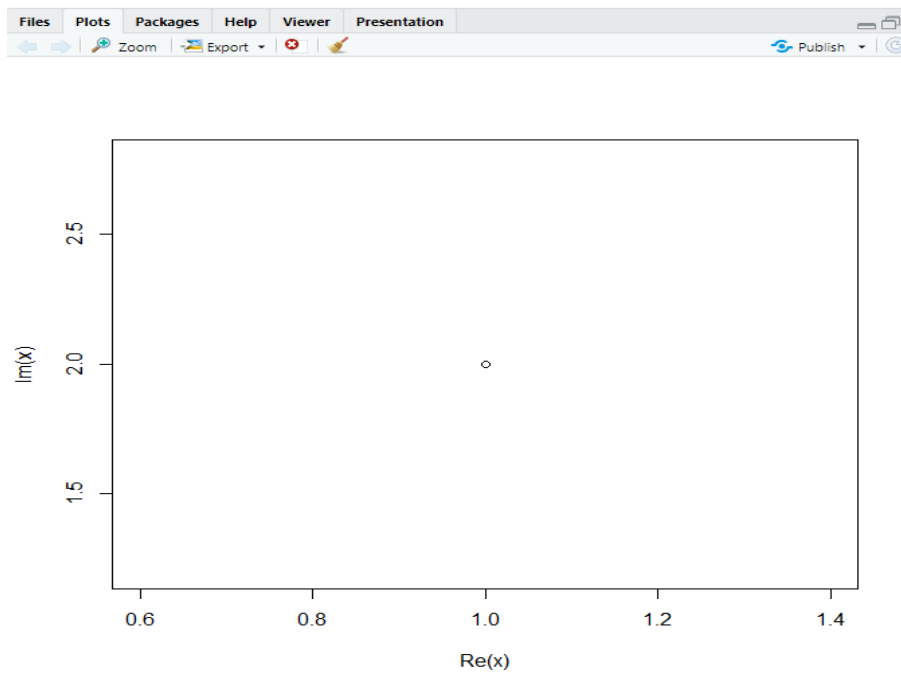


Figure 1.11: Représentation graphique de  $x$  (fonction `*plot*`)



La fonction `plot()` affiche la représentation graphique d'une variable (**x dans cet exemple**).

## Opérateur Mod

```
> Mod(x)
[1] 2.236068
> |
```

Figure 1.12: Utilisation de la fonction `Mod` pour obtenir le module de  $x$

La fonction `Mod()` retourne le module de  $x$ .

## Opérateur Arg

```
> Arg(x)
[1] 1.107149
> |
```

Figure 1.13: Utilisation de la fonction `Arg` pour obtenir l'argument de  $x$

La fonction `Arg()` retourne l'argument de  $x$ .

## 1.3 Comparaison logique

```
> b>a
[1] TRUE
> a>b
[1] FALSE
> b==a
[1] FALSE
> |
```

Figure 1.14: Comparaison logique entre  $a$  et  $b$

Les comparaisons logiques entre variables retournent des valeurs booléennes (**TRUE** ou **FALSE**).



### 1.3.1 Déclaration d'une variable logique

Mehdi Bahi

```
> y = TRUE  
>
```

Figure 1.15: Déclaration d'une variable logique (image 14)

y	TRUE
---	------

Figure 1.16: Affichage de la variable logique (image 15)

### 1.3.2 Fonction is.logical()

```
> is.logical(y)  
[1] TRUE
```

Figure 1.17: Utilisation de la fonction is.logical()

La fonction `is.logical()` retourne `TRUE` si la variable est un booléen.

## 1.4 Déclaration d'un caractère

```
> s = 'F'  
> t = "H"  
>
```

Figure 1.18: Déclaration d'un caractère

s	"F"
t	"H"

Figure 1.19: Affichage du caractère

Indépendamment de la manière dont le caractère est déclaré (**avec ' ' ou " "**), cela reste le même.





### 1.4.1 Fonction `is.character()`

Mehdi Bahi

```
> is.character(s)
[1] TRUE
> is.character(t)
[1] TRUE
> |
```

Figure 1.20: Utilisation de la fonction `is.character()`

La fonction `is.character()` retourne `TRUE` si la variable est un caractère.

### 1.4.2 Déclaration d'une variable `NA` et d'une variable `NULL` & méthode `is.null()`

```
> x = NA
> |
```

Figure 1.21: Déclaration d'une variable `NA`

x	NA
---	----

Figure 1.22: Affichage d'une variable `NA`

```
> x = NA
> is.null(x)
[1] FALSE
> x = NULL
> is.null(x)
[1] TRUE
> |
```

Figure 1.23: Déclaration d'une variable `NULL`

x	NULL
---	------

Figure 1.24: Affichage d'une variable `NULL`

La fonction `is.null()` vérifie si une variable est stockée en tant que `NULL`.



## 1.5 Déclaration d'un vecteur

Mehdi Bahi

```
> x <- c(3,NA,6)
> |
```

Figure 1.25: Déclaration du vecteur x

Environment	History	Connections	Tutorial
151 MiB			
R Global Environment			
values			
x	num [1:3] 3 NA 6		

Figure 1.26: Affichage du vecteur x

La fonction `c()` retourne un vecteur avec le type de ses éléments et leur nombre  $[i : n]$ .

### 1.5.1 Fonction `is.na()`

```
> is.na(x)
[1] FALSE TRUE FALSE
```

Figure 1.27: Utilisation de la fonction `is.na()`

La fonction `is.na()` vérifie si les variables du vecteur sont des valeurs NA (Not Available). Elle retourne un vecteur avec la valeur `TRUE` si un élément est de type NA, et `FALSE` sinon.

### 1.5.2 Déclaration d'un vecteur avec des variables de type brut (`as.raw()` fonction)

```
> x <- as.raw(c(9,10,15,16))
> |
```

Figure 1.28: Déclaration d'un vecteur avec des variables de type brut

Environment	History	Connections	Tutorial
152 MiB			
R Global Environment			
values			
x	raw [1:4] 09 0a 0f 10		

Figure 1.29: Affichage du vecteur brut dans l'environnement



```
> x  
[1] 09 0a 0f 10  
>
```

Figure 1.30: Affichage du vecteur brut dans la console

Nous utilisons la fonction `as.raw()` pour convertir l'argument du vecteur en type brut.

## 1.6 Fonction `mode()`

```
> mode(x)  
[1] "raw"  
> |
```

Figure 1.31: Utilisation de la fonction `mode()`

La fonction `mode()` retourne le type des éléments du vecteur.

## 1.7 Tentative de calcul de la moyenne

```
> mean(x)  
[1] NA  
> |
```

Figure 1.32: Tentative de calcul de la moyenne avec des données manquantes

La moyenne du vecteur ne peut pas être mesurée en raison des données manquantes dans le vecteur.

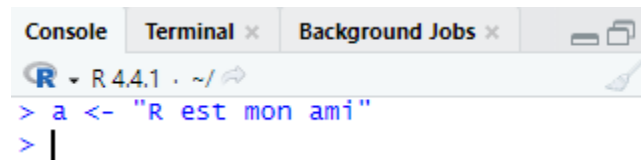


### 1.7.1 Exclusion des données manquantes pour calculer la moyenne

```
> mean(x, na.rm=TRUE)
[1] 4.5
> |
```

Figure 1.33: Ignorer les données manquantes pour calculer la moyenne

## 1.8 Déclaration d'une chaîne de caractères



```
Console Terminal x Background Jobs x
R 4.4.1 ~ /
> a <- "R est mon ami"
> |
```

Figure 1.34: Déclaration d'une chaîne de caractères (image 33)

a	"R est mon ami"
---	-----------------

Figure 1.35: Affichage de la chaîne de caractères (image 34)

Une chaîne de caractères est déclarée avec " <- ".

### Vérification du type de a

```
> mode(a)
[1] "character"
> is.character(a)
[1] TRUE
> |
```

Figure 1.36: Vérification du type de \*a\* (image 35)

Une chaîne de caractères est de type `character`.



### 1.8.1 Fonction `as.character()`

Mehdi Bahi

```
> as.character(2.3)
[1] "2.3"
>
```

Figure 1.37: Utilisation de la fonction `*as.character()*` (image 36)

La fonction `as.character()` retourne un vecteur dont les éléments sont de type `character`.

### 1.8.2 Fonction `as.numeric()`

```
> b <- "2.3"
> as.numeric(b)
[1] 2.3
> |
```

Figure 1.38: Utilisation de la fonction `*as.numeric()*` (image 37)

La fonction `as.numeric()` transforme une chaîne numérique en sa forme numérique.

### 1.8.3 Fonction `as.integer()`

```
> as.integer("3.4")
[1] 3
```

Figure 1.39: Utilisation de la fonction `as.integer()`

La fonction `as.integer()` retourne la valeur entière d'un nombre / chaîne numérique.

## 1.9 Conversion automatique

```
> c(2, "3")
[1] "2" "3"
> |
```

Figure 1.40: Conversion automatique

La priorité est donnée aux caractères, ce qui entraîne la conversion automatique de la variable numérique en chaîne.



## 1.10 Conversion impossible

Mehdi Bahi

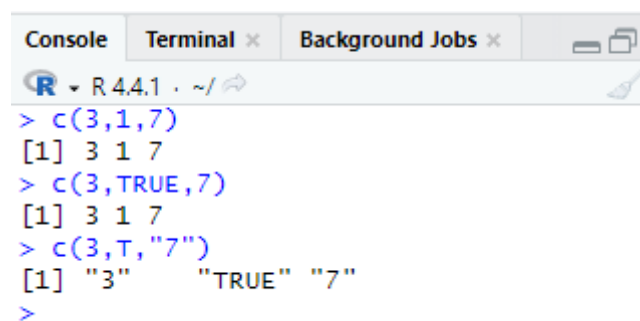
```
> as.integer("3.quatre")
[1] NA
warning message:
NAS introduced by coercion
> |
```

Figure 1.41: Conversion impossible

Une valeur de type chaîne ne peut pas être transformée en entier.  
La fonction `as.integer()` retourne NA à la place.

## 1.11 Vecteurs (Suite.)

### 1.11.1 Priorité des valeurs dans les vecteurs



```
Console Terminal x Background Jobs x
R 4.4.1 ~ /
> c(3,1,7)
[1] 3 1 7
> c(3,TRUE,7)
[1] 3 1 7
> c(3,T,"7")
[1] "3" "TRUE" "7"
>
```

Figure 1.42: Priorité des valeurs dans les vecteurs

Les vecteurs ont la plus haute priorité, suivis des nombres (entiers, doubles, etc.), et enfin des booléens.

### 1.11.2 Fonction `seq()`

```
> seq(from=0,to=1,by=0.1)
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6
[8] 0.7 0.8 0.9 1.0
>

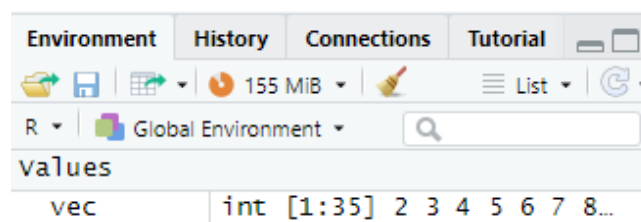
> seq(from=0,to=20,length=5)
[1] 0 5 10 15 20
> |
```

La fonction `seq()` génère un vecteur d'éléments allant de 'from' à 'to', avec un pas défini par 'by' ou une longueur définie par 'length'.



### 1.11.3 Déclaration d'un vecteur avec des compréhensions

```
> vec <- 2:36
> vec
[1]  2  3  4  5  6  7  8  9 10 11
[11] 12 13 14 15 16 17 18 19 20 21
[21] 22 23 24 25 26 27 28 29 30 31
[31] 32 33 34 35 36
```



### 1.11.4 Fonction is.vector()

```
> is.vector(vec)
[1] TRUE
> |
```

Figure 1.43: Utilisation de la fonction `is.vector()`



### 1.11.5 Déclaration d'un vecteur avec des compréhensions

```
> x <- 1:3  
> x  
[1] 1 2 3  
> |
```

x	int [1:3] 1 2 3
---	-----------------

### 1.11.6 Déclaration d'un vecteur avec la fonction c()

```
> y <- c(1,2,3)  
> y  
[1] 1 2 3  
> |
```

y	num [1:3] 1 2 3
---	-----------------

### 1.11.7 Fonction class()

```
> class(x)  
[1] "integer"  
> class(y)  
[1] "numeric"  
>
```

Figure 1.44: Différence entre les méthodes de déclaration

On conclut que si une variable est déclarée avec des compréhensions, son type sera `integer`, tandis que si elle est déclarée avec la méthode `c()`, son type sera `numeric`.

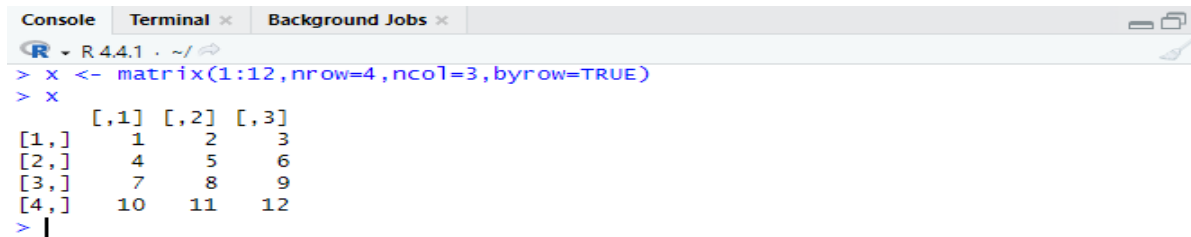




## 1.12 Exemples de Matrices et Tableaux

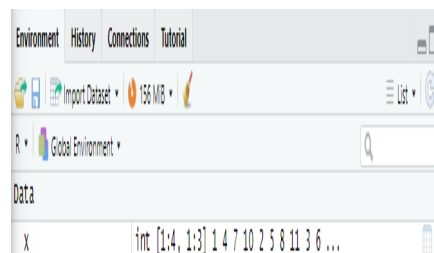
Mehdi Bahi

### 1.12.1 Déclaration d'une matrice avec la fonction `matrix()`



```
> x <- matrix(1:12,nrow=4,ncol=3,byrow=TRUE)
> x
      [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> |
```

Figure 1.45: Déclaration d'une matrice (`byrow = TRUE`)



Environment History Connections Tutorial

Import Dataset 156 MB

R Global Environment

Data

x int [1:4, 1:3] 1 4 7 10 2 5 8 11 3 6 ...

Figure 1.46: L'affichage de la matrice dans l'environnement

	V1	V2	V3
1	1	2	3
2	4	5	6
3	7	8	9
4	10	11	12

Figure 1.47: L'affichage de la matrice en utilisant `view()`

`byrow = FALSE`

```
> y <- matrix(1:12,nrow=4,ncol=3,byrow=FALSE)
> y
      [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> |
```

Figure 1.48: Déclaration d'une matrice (`byrow = FALSE`)

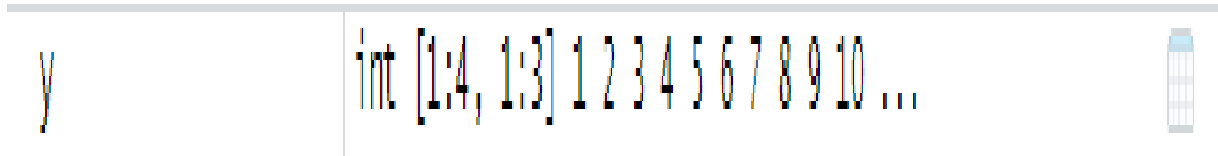


Figure 1.49: L’affichage de la matrice dans l’environnement





		V1		V2		V3	
	1		1		5		9
	2		2		6		10
	3		3		7		11
	4		4		8		12

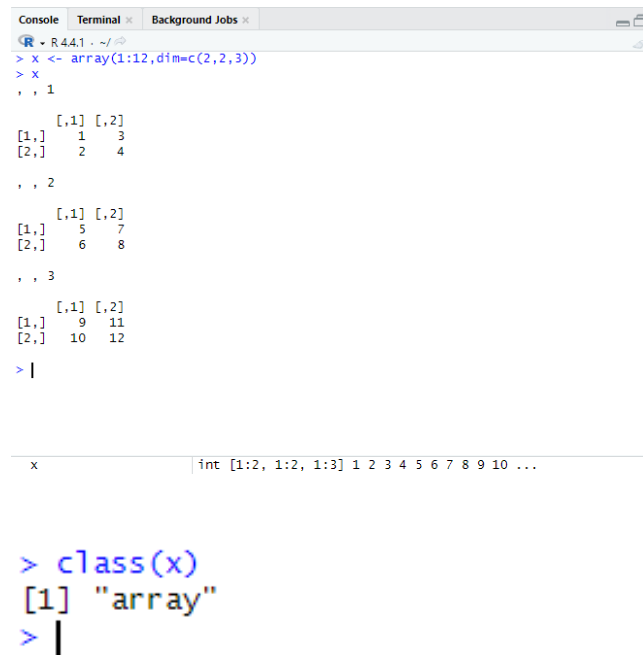
Figure 1.50: L’affichage de la matrice en utilisant view()



On conclut que l'argument `byrow` détermine si la matrice est remplie par ligne (`byrow=TRUE`) ou par colonne (`byrow=FALSE`).

```
> class(y)
[1] "matrix" "array"
>
```

### 1.12.2 Déclaration d'un tableau avec la fonction `array()`



```
Console Terminal Background Jobs
> x <- array(1:12,dim=c(2,2,3))
> x
, , 1
      [,1] [,2]
[1,]    1    3
[2,]    2    4
, , 2
      [,1] [,2]
[1,]    5    7
[2,]    6    8
, , 3
      [,1] [,2]
[1,]    9   11
[2,]   10   12
> |

x               int [1:2, 1:2, 1:3] 1 2 3 4 5 6 7 8 9 10 ...

> class(x)
[1] "array"
> |
```



## 1.13 Exemple de Listes

Mehdi Bahi

### 1.13.1 Déclaration d'une liste avec la fonction `list()`

```
Score Terminal Background Jobs
> R > 6.6.1 --
> a <- list(TRUE, -1:3, matrix(1:4, nrow=2), c(2+1i, 3), "une chaîne de caracteres")
> a
[[1]]
[1] TRUE
[[2]]
[1] -1 0 1 2 3
[[3]]
[1,2] 1 2
[2,] 3 4
[[4]]
[1] 2+1i 3+0i
[[5]]
[1] "une chaîne de caracteres"
```

A	list [5]	List of length 5
[[1]]	logical [1]	TRUE
[[2]]	integer [5]	-1 0 1 2 3
[[3]]	integer [2 x 2]	1 2 3 4
[[4]]	complex [2]	1+2i 3+0i
[[5]]	character [1]	'Une chaîne de caracteres'

```
> class(A)
[1] "list"
> |
```

Une liste peut contenir des variables de tout type.



## 1.14 Exemple de Facteurs

Mehdi Bahi

### 1.14.1 Déclaration d'un facteur avec la fonction factor()

```
Console Terminal x Background Jobs x
R - R44.1 - ~/
> x <- factor(c("bleu","vert","bleu","rouge","bleu","vert","vert"))
> x
[1] bleu vert bleu rouge bleu
[6] vert vert
Levels: bleu rouge vert
> |
```

```
x Factor w/ 3 levels "bleu","rouge",...: 1 3 1 2 1 3 3
```

```
> class(x)
[1] "factor"
> |
```

#### Fonction levels()

```
> levels(x)
[1] "bleu" "rouge" "vert"
> |
```

### 1.14.2 Déclaration d'un facteur ordonné avec ordered(c(), levels=c())

```
> z <- ordered(c("Petit","Grand","Moyen","Grand","Moyen","Petit","Petit"),levels=c
("Petit","Moyen","Grand"))
> class(z)
[1] "ordered" "factor"
> |
```

```
z Ord.factor w/ 3 levels "Petit"<"Moyen"<"...: 1 3 2 3 2 1 1
```

Un facteur ordonné est similaire à un facteur classique, mais l'ordre est défini manuellement.



## 1.15 Exemple de Data Frames

Mehdi Bahi

### 1.15.1 Déclaration d'un data frame avec la fonction `data.frame()`

```
Console Terminal Background Jobs
R 4.4.1 - ~/
> IMC <- data.frame(Sexe=c("H","F","H","F","H","F"),Taille=c(1,83,1,76,1,82,1,60,
1,90,1,66), Poids=c(67,58,66,48,75,55,row.names=c("Remy","Lo1","Pierre","Domi","Be
n","Cecile")))
> IMC
  Sexe Taille Poids
1    H      1   67
2    F     83   58
3    H      1   66
4    F     76   48
5    H      1   75
6    F     82   55
7    H      1 Remy
8    F     60  Lo1
9    H      1 Pierre
10   F     90  Domi
11   H      1   Ben
12   F     66 Cecile
>
```

Environment History Connections Tutorial

Import Dataset 157 MiB

R Global Environment

Data

IMC 12 obs. of 3 variables

	Sexe	Taille	Poids
1	H	1	67
2	F	83	58
3	H	1	66
4	F	76	48
5	H	1	75
6	F	82	55
7	H	1	Remy
8	F	60	Lo1
9	H	1	Pierre
10	F	90	Domi
11	H	1	Ben
12	F	66	Cecile



## Chapter 2

# Exemple de Pré-Traitement

### 2.1 Étape 1: Définition de la problématique et des données

On souhaite étudier les notes des étudiants du module de statistique.

#### 2.1.1 Question de recherche

Quelle est la performance des étudiants ?

Reformulation de la question de recherche : y'a-t-il un lien entre les caractéristiques des étudiants et leur performance ?

### 2.2 Étape 2: Collecte des données

#### 2.2.1 Collecte manuelle (d1) - Saisie des données

```
1 age=c(19,20,21,50)
2 genre=c("H","F","H","F")
3 niveau=c("1A","2A","1A","3A")
4 filiere=c("GI","GIndus","GI","GI")
5 pays=c("Mauritanie","Maroc","Maroc","Cameroun")
6 note=c(12.5,19,17,3.5)
7 d1=data.frame(age,genre,niveau,filiere,pays,note)
8 View(d1)
```

Listing 2.1: Saisie des données

#### 2.2.2 Collecte à partir d'un fichier (d2)

```
1 library(readxl)
2 d2 <- read_excel("data.xlsx")
3 View(d2)
```

Listing 2.2: Chargement des données depuis un fichier



## 2.3 Étape 3: Pré-traitement

Mehdi Bahi

### 2.3.1 Codification et conversion

```
1 if (!is.numeric(d2$age)){
2   d2$age=as.numeric(d2$age)
3 }
4 if (!is.numeric(d2$note)){
5   d2$note=as.numeric(d2$note)
6 }
7 if (!is.character(d2$genre)){
8   d2$genre=as.character(d2$genre)
9 }
10 if (!is.character(d2$pays)){
11   d2$pays=as.character(d2$pays)
12 }
13 if (!is.character(d2$niveau)){
14   d2$niveau=as.character(d2$niveau)
15 }
```

Listing 2.3: Codification des variables

- Les variables qualitatives sont à la base des textes (nominale ou ordinale). - Les variables qualitatives sont par défaut nominales (ordinale dans le cas d'un ordre conventionnel).

```
1 if (is.character(d2$niveau)){
2   d2$niveau=as.ordered(d2$niveau)
3 }
4 if (is.character(d2$genre)){
5   d2$genre=as.factor(d2$genre)
6 }
7 if (is.character(d2$pays)){
8   d2$pays=as.factor(d2$pays)
9 }
10 if (is.character(d2$filiere)){
11   d2$filiere=as.factor(d2$filiere)
12 }
```

Listing 2.4: Conversion des variables qualitatives





## 2.3.2 Nettoyage des données

Mehdi Bahi

### Traitement des valeurs aberrantes

```
1 boxplot(d2$age)
2 out=boxplot.stats(d2$age)$out
3 for (i in 1:length(d2$age))
4 {
5     for (j in 1:length(out)){
6         if(d2$age[i]==out[j] && !is.na(d2$age[i])){
7             d2$age[i]<-NA
8         }
9     }
10 }
11 boxplot(d2$note)
```

Listing 2.5: Détection et traitement des valeurs aberrantes

**Note:** On ne va pas traiter les valeurs de `note` car c'est normal de trouver des valeurs entre 0 et 20.

### Traitement des valeurs manquantes

```
1 out2=boxplot.stats(d2$note)$out
2 for (i in 1:length(d2$note))
3 {
4     for (j in 1:length(out2)){
5         if(d2$note[i]==out2[j] && !is.na(d2$note[i])){
6             d2$note[i]<-NA
7         }
8     }
9 }
10 c=0
11 for (i in 1:length(d2$age))
12 {
13     if (is.na(d2$age[i])){
14         c=c+1
15     }
16 }
17 propNA=c/length(d2$age)
18 if(propNA>=0.05){
19     print("estimation")
20 }else{
21     print("supprimer")
22 }
```

Listing 2.6: Estimation ou suppression des valeurs manquantes



## Chapter 3

# Exemple de Pré-traitement et Traitement

### 3.1 Étape 1 : Collecte de données

Charger la bibliothèque pour lire un fichier Excel.

```
1 library(readxl)
2 d2 <- read_excel("C:/Users/Mehdi/Downloads/Data-SalaryGender.xlsx")
3 View(d2)
```

Listing 3.1: Chargement des données Excel

### 3.2 Étape 2 : Pré-traitement

#### 3.2.1 Conversion et recodification

```
if (!is.numeric(d2$Age)){
  d2$Age=as.numeric(d2$Age)
}
if (!is.character(d2$Sexe)){
  d2$Sexe=as.character(d2$Sexe)
}
if (!is.numeric(d2$Scolarité)){
  d2$Scolarité=as.numeric(d2$Scolarité)
}
if (!is.numeric(d2$'Revenuen$')){
  d2$'Revenuen$'=as.numeric(d2$'Revenuen$')
}
if (!is.character(d2$Fonction)){
  for (i in 1:length(d2$Fonction)){
    if (d2$Fonction[i]==1) {
      d2$Fonction[i]="responsable"
    }
    if (d2$Fonction[i]==2) {
```



```
    d2$Fonction[i]="ingenieur"
  }
  if (d2$Fonction[i]==3) {
    d2$Fonction[i]="technicien"
  }
}
d2$Fonction=as.character(d2$Fonction)
}
if (!is.numeric(d2$Sexe)){
  d2$Sexe=as.numeric(d2$Sexe)
  for (i in 1:length(d2$Sexe)){
    if (d2$Sexe[i]==1){
      d2$Sexe[i]="homme"
    }
    if (d2$Sexe[i]==2){
      d2$Sexe[i]="femme"
    }
  }
}
d2$Sexe=as.character(d2$Sexe)
}
```

Suite de codification

```
1 if (!is.numeric(d2$Questionno1)){
2   d2$Questionno1=as.numeric(d2$Questionno1)
3   for (i in 1:length(d2$Questionno1)){
4     if (d2$Questionno1[i]==1){
5       d2$Questionno1[i]="oui"
6     }
7     if (d2$Questionno1[i]==2){
8       d2$Questionno1[i]="non"
9     }
10  }
11  d2$Questionno1=as.character(d2$Questionno1)
12 }
13 if (is.character(d2$Fonction)){
14   d2$Fonction=as.ordered(d2$Fonction)
15 }
16 if (is.character(d2$Sexe)){
17   d2$Sexe=as.factor(d2$Sexe)
18 }
19 if (is.character(d2$Questionno1)){
20   d2$Questionno1=as.factor(d2$Questionno1)
21 }
```

Listing 3.2: Suite de la recodification



### 3.2.2 Traitement des valeurs aberrantes/manquantes Mehdi Bahi

**Observation :** Il n'y a pas de valeurs manquantes dans d2.

#### Traitement des valeurs aberrantes

```
boxplot(d2$Age)
```

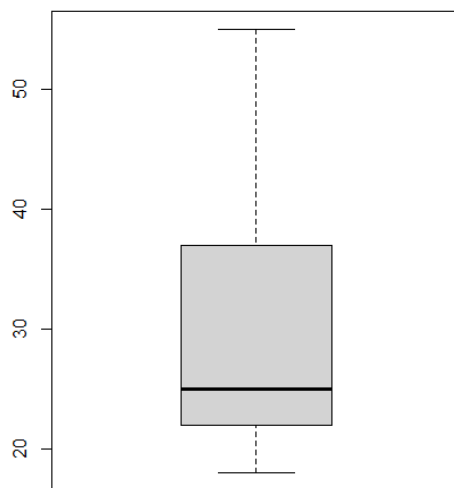
# Pas de valeurs aberrantes pour Age

```
boxplot(d2$Scolarité)
```

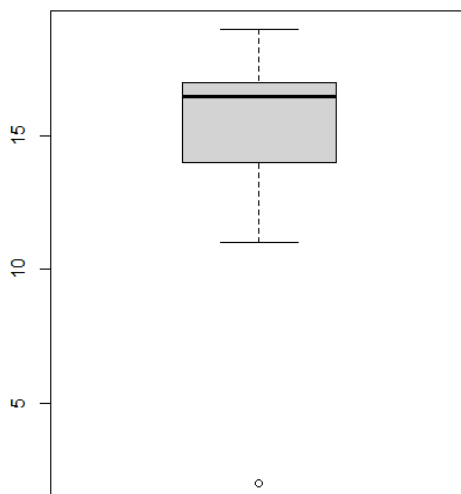
# Une valeur aberrante détectée pour Scolarité

```
boxplot(d2$'Revenuen$')
```

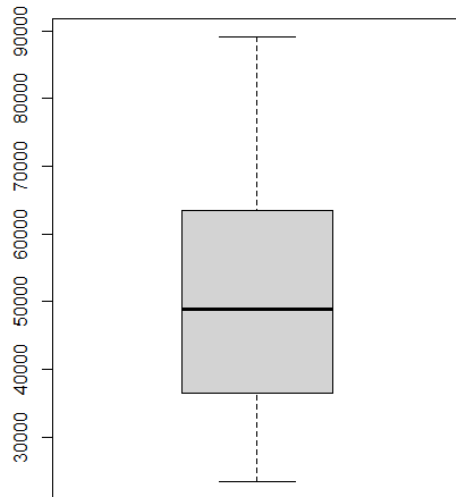
# Pas de valeurs aberrantes pour Revenuen\$



C/C : Pas de valeurs aberrantes pour Age.



C/C : Une valeur aberrante détectée pour Scolarité.



C/C : Pas de valeurs aberrantes pour Revenuen\$.

### Traitement des valeurs aberrantes

```
out=boxplot.stats(d2$Scolarité)$out
for (i in 1:length(d2$Scolarité)){
  if (d2$Scolarité[i]==out && !is.na(d2$Scolarité[i])){
    d2$Scolarité[i]=NA
  }
}

# Remplacement des valeurs aberrantes par la moyenne
for (i in 1:length(d2$Scolarité)){
  if (is.na(d2$Scolarité[i])){
    d2$Scolarité[i]=mean(d2$Scolarité, na.rm=TRUE)
  }
}
```



### 3.2.3 Conversion en entier pour Scolarité

Mehdi Bahi

```
d2$Scolarité = as.integer(d2$Scolarité)
```

### 3.2.4 Test de normalité des variables quantitatives

```
shapiro.test(d2$Scolarité)
```

```
shapiro-wilk normality test
data:  d2$scolarité
W = 0.77705, p-value = 2.573e-05
```

**Conclusion :** Pas de normalité pour Scolarité.

```
shapiro.test(d2$Age)
```

```
shapiro-wilk normality test
data:  d2$Age
W = 0.8371, p-value = 0.0003368
```

**Conclusion :** Pas de normalité pour Age.

```
shapiro.test(d2$'Revenuen$')
```

**Output :** La normalité est respectée pour Revenuen\$.

### 3.2.5 Quasi-normalité

**Définition :** La quasi-normalité désigne une distribution proche de la loi normale. Pour déterminer cela, on vérifie :

- **Aplatissement (Kurtosis)** et **asymétrie (Skewness)**.
- Si ces valeurs appartiennent à l'intervalle  $[-3, 3]$ , la distribution est considérée quasi-normale.

**Cas d'utilisation :**

- Si les conclusions des tests paramétriques et non-paramétriques coïncident, on utilise le test paramétrique.
- Sinon, on opte pour le test non-paramétrique.



```
1 library(moments)
2 skewness(d2$Age)
3 kurtosis(d2$Age)
```

Listing 3.3: Calcul de skewness et kurtosis

```
> library(moments)
> skewness(d2$Age)
[1] 1.034434
> kurtosis(d2$Age)
[1] 2.817338
```

**Conclusion :** L'Age est quasi-symétrique, ce qui permet l'utilisation des deux types de tests :

- Tests paramétriques.
- Tests non-paramétriques.



## 3.3 Étape 3 : Traitement

Mehdi Bahi

### 3.3.1 Statistique univariée

summary(d2)

Output :

```

Participantes  Sexe      Age      Scolarité  Fonction  Revenuen$
Min. : 1.00    Length:30    Min. :18.0    Min. : 2.00    Min. :1.000    Min. :23443
1st Qu.: 8.25    class :character 1st Qu.:22.0    1st Qu.:14.00    1st Qu.:1.000    1st Qu.:38213
Median :15.50    Mode  :character Median :25.0    Median :16.50    Median :2.000    Median :48889
Mean :15.50      Mean :29.6      Mean :15.47    Mean :2.033    Mean :50484
3rd Qu.:22.75    3rd Qu.:35.5    3rd Qu.:17.00  3rd Qu.:3.000    3rd Qu.:62560
Max. :30.00      Max. :55.0      Max. :19.00    Max. :3.000    Max. :89098
questionno1
Length:30
Class :character
Mode :character

```

L'échantillon est équilibré par rapport au genre, alors on ne peut pas conclure.

```
chisq.test(table(d2$Sexe))
```

Output :

```

Chi-squared test for given probabilities
data: table(d2$Sexe)
X-squared = 0, df = 1, p-value = 1

```

Pas de différence significative entre hommes et femmes.

```
chisq.test(table(d2$Fonction))
```

Output :

```

Chi-squared test for given probabilities
data: table(d2$Fonction)
X-squared = 0.2, df = 2, p-value = 0.9048

```

Pas de différence significative en termes de scolarité.

```
chisq.test(table(d2$Fonction, d2$Sexe))
```

```
table(d2$Fonction, d2$Sexe)
```

```
fisher.test(table(d2$Fonction, d2$Sexe))
```

Output :

```

> chisq.test(table(d2$Fonction,d2$Sexe))

Pearson's Chi-squared test

data: table(d2$Fonction, d2$Sexe)
X-squared = 3.6727, df = 2, p-value = 0.1594

warning message:
In chisq.test(table(d2$Fonction, d2$Sexe)) :
  Chi-squared approximation may be incorrect
> table(d2$Fonction,d2$Sexe)

  1 2
1 6 4
2 6 3
3 3 8
> fisher.test(table(d2$Fonction,d2$Sexe))

Fisher's Exact Test for Count Data

data: table(d2$Fonction, d2$Sexe)
p-value = 0.2211
alternative hypothesis: two.sided

```

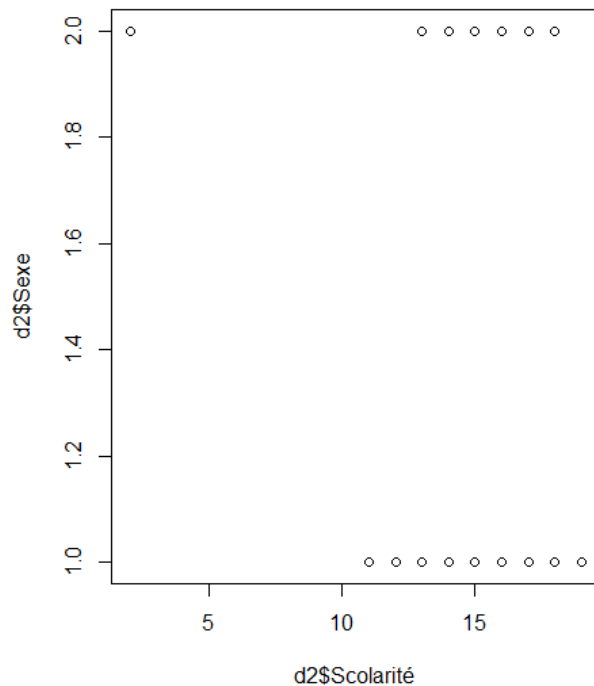
Conclusion : Il n'y a pas de différence entre les hommes et les femmes par fonction.





```
plot(d2$Sxex, d2$Scolarité)
```

Output :



```
library(car)  
leveneTest(d2$Scolarité ~ d2$Sxex)
```

Output :

```
Levene's Test for Homogeneity of Variance (center = median)  
Df F value Pr(>F)  
group 1 0.0041 0.9497  
      28  
Warning message:  
In leveneTest.default(y = y, group = group, ...) : group coerced to factor.
```

```
t.test(d2$Scolarité ~ d2$Sxex, var.equal = TRUE)
```

Output :

```
Two Sample t-test  
data: d2$Scolarité by d2$Sxex  
t = 0.65354, df = 28, p-value = 0.5187  
alternative hypothesis: true difference in means between group 1 and group 2 is not equal to 0  
95 percent confidence interval:  
-1.707448 3.307448  
sample estimates:  
mean in group 1 mean in group 2  
15.86667 15.06667
```

Conclusion : Les hommes et les femmes ont les mêmes années scolaires.



### 3.3.2 La vérification du revenu

Mehdi Bahi

```
leveneTest(d2$Revenuen ~ d2$Sexe)
t.test(d2$Revenuen ~ d2$Sexe, var.equal = TRUE)
```

Output :

```
> leveneTest(d2$Revenuen ~ d2$Sexe)
Levene's Test for Homogeneity of Variance (center = median)
      Df F value Pr(>F)
group 1  0.003  0.957
      28
warning message:
In leveneTest.default(y = y, group = group, ...) : group coerced to factor.
> t.test(d2$Revenuen ~ d2$Sexe, var.equal = TRUE)

Two Sample t-test

data:  d2$Revenuen by d2$Sexe
t = 2.6836, df = 28, p-value = 0.01209
alternative hypothesis: true difference in means between group 1 and group 2 is not equal to 0
95 percent confidence interval:
 3401.911 25343.156
sample estimates:
mean in group 1 mean in group 2
 57669.80      43297.27
```

Conclusion : Il y a une différence entre les femmes et les hommes, alors l'hypothèse H1 est acceptée. On étudie s'il y a réellement une différence ou non.

```
chisq.test(table(d2$Questionno1))
```

Output :

```
Chi-squared test for given probabilities
data:  table(d2$Questionno1)
X-squared = 4.8, df = 1, p-value = 0.02846
```

Conclusion : Les Canadiens pensent qu'il n'y a pas de différence entre les revenus des femmes et des hommes, mais en réalité, il y en a une.



### 3.4 Calcul de alpha

Mehdi Bahi

```
library(psych)
d3 <- read_excel("C:/Users/Mehdi/Downloads/Items-SalaryGender.xlsx")
View(d3)
alpha(data.frame(d3$item1, d3$item2, d3$item3, d3$item4, d3$item5))
```

Output :

```
Reliability analysis
Call: alpha(x = data.frame(d3$item1, d3$item2, d3$item3, d3$item4,
  d3$item5))

raw_alpha std.alpha G6(smc) average_r S/N ase mean sd median_r
0.77 0.77 0.76 0.4 3.3 0.018 3.2 0.9 0.34

95% confidence boundaries
lower alpha upper
Feldt 0.73 0.77 0.8
Duhachek 0.73 0.77 0.8

Reliability if an item is dropped:
raw_alpha std.alpha G6(smc) average_r S/N alpha se var.r med.r
d3.Item1 0.67 0.67 0.66 0.34 2.0 0.027 0.033 0.29
d3.Item2 0.75 0.75 0.74 0.43 3.0 0.020 0.060 0.43
d3.Item3 0.67 0.67 0.66 0.34 2.1 0.026 0.032 0.29
d3.Item4 0.70 0.70 0.67 0.37 2.3 0.024 0.021 0.34
d3.Item5 0.81 0.80 0.78 0.51 4.1 0.015 0.027 0.53

Item statistics
n raw.r std.r r.cor r.drop mean sd
d3.Item1 420 0.82 0.82 0.79 0.68 3.2 1.3
d3.Item2 420 0.66 0.67 0.52 0.47 3.3 1.2
d3.Item3 420 0.81 0.81 0.78 0.67 3.3 1.3
d3.Item4 420 0.76 0.77 0.73 0.61 3.6 1.2
d3.Item5 420 0.54 0.53 0.33 0.29 2.7 1.3

Non missing response frequency for each item
1 2 3 4 5 miss
d3.Item1 0.13 0.17 0.24 0.28 0.19 0
d3.Item2 0.07 0.17 0.30 0.25 0.20 0
d3.Item3 0.15 0.11 0.31 0.20 0.23 0
d3.Item4 0.07 0.10 0.26 0.33 0.24 0
d3.Item5 0.23 0.24 0.22 0.21 0.10 0
```