3_1_cours

October 15, 2018

Table of Contents

1 Importation des données

Nous avons vu dans le dernier cours les *data frames*, nous l'avions créée manuellement. Toutefois, nous allons souvent importer des données en pratique sous plusieurs formats. Ces fichiers que nous allons importer seront souvent dans des fichiers `.csv` (*Comma-separated values*). Ces fichiers sont très populaires et ils sont générés par Excel .

1.1 read.csv

```
In [1]: options(repr.matrix.max.cols=8, repr.matrix.max.rows=8) #seulement pour afficher 8 ligne
```

On peut lire les fichiers `.csv` localement en précisant le chemin exacte menant vers du fichier en question.

```
In [2]: # test_csv <-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/exemple_1.
        test_csv <-read.csv("exemple_1.csv")
        test_csv
```

Segment	VitesseM	PuissanceEstim
Km1	31.9	130
Km2	33.3	165
Km3	28.1	130
Km4	30.8	133
Km5	27.7	103
Km6	31.2	154

Ou directement à partir du web:

```
In [1]: test_csv <-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/exemple_1.cs
        test_csv
```

Segment	VitesseM	PuissanceEstim
Km1	31.9	130
Km2	33.3	165
Km3	28.1	130
Km4	30.8	133
Km5	27.7	103
Km6	31.2	154

Lorsque nous écrivons `read.csv`, R traite importe ce fichier sous format `data frame`, il nous retourne les noms de colonnes, les lignes ainsi que la classe du df

```
In [2]: attributes(test_csv)
```

$names 1. 'Segment' 2. 'VitesseM' 3. 'PuissanceEstim'

$class 'data.frame'

$row.names 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6

Dans la méthode `read.csv`, il existe un argument optionnel `_header_` qui est par défaut `header=T`. Cet argument spécifie si les données que nous voulons importer possèdent des noms de colonne (`header=TRUE` ~ `header=T`.) ou pas (`header=FALSE` ~ `header=F`.). Regardons ce que ça donnerait si nous changeons la valeur `header=F`;

```
In [9]: test_csv <-read.csv("exemple_1.csv", header = F)
        test_csv
```

V1	V2	V3
Segment	VitesseM	PuissanceEstim
Km1	31.9	130
Km2	33.3	165
Km3	28.1	130
Km4	30.8	133
Km5	27.7	103
Km6	31.2	154

On remarque que R crée des noms de colonnes appelés `V1, V2...etc.`

```
In [11]: exemple <-read.csv("exemple_1.csv", header = T)
         exemple
```

Segment	VitesseM	PuissanceEstim
Km1	31.9	130
Km2	33.3	165
Km3	28.1	130
Km4	30.8	133
Km5	27.7	103
Km6	31.2	154

Regardons la classe de la variable "Segment";

```
In [12]: class(exemple$Segment)
```

'factor'

Surprise!. En effet, il existe une autre option dans la méthode `read.csv` qui permet de traiter les catégories en type caractère.

```
In [14]: exemple <-read.csv("exemple_1.csv", header = T, stringsAsFactors=F)
         exemple
```

Segment	VitesseM	PuissanceEstim
Km1	31.9	130
Km2	33.3	165
Km3	28.1	130
Km4	30.8	133
Km5	27.7	103
Km6	31.2	154

Maintenant, regardons la classe de la variable "Segment"

```
In [15]: class(exemple$Segment)
```

'character'

1.1.1 Une partie seulement du df

Il est possible de lire seulement certaines colonnes du fichier csv qu'on veut importer;

```
In [30]: exemple <-read.csv("exemple_1.csv", header = T, stringsAsFactors=F)[,2:3]
         exemple
```

VitesseM	PuissanceEstim
31.9	130
33.3	165
28.1	130
30.8	133
27.7	103
31.2	154

```
In [31]: exemple <-read.csv("exemple_1.csv", header = T, stringsAsFactors=F)[,c(1,3)]
         exemple
```

Segment	PuissanceEstim
Km1	130
Km2	165
Km3	130
Km4	133
Km5	103
Km6	154

1.2 read.table

Une autre façon d'importer des données à l'intérieur des df est d'utiliser la méthode `read.table` qui traite les fichiers `text`;

```
In [17]: read.table("exemple_1.txt", header=T)
```

Segment.VitesseM.PuissanceEstim
Km1,31.9,130
Km2,33.3,165
Km3,28.1,130
Km4,30.8,133
Km5,27.7,103
Km6,31.2,154

On voit bien que les colonnes n'ont pas été séparées comme il faut. Nous devons spécifier les caractères qui séparent ces variables.

```
In [18]: read.table("exemple_1.txt", header=T, sep = ",")
```

Segment	VitesseM	PuissanceEstim
Km1	31.9	130
Km2	33.3	165
Km3	28.1	130
Km4	30.8	133
Km5	27.7	103
Km6	31.2	154

1.3 En utilisant le package `RCurl`

Il est possible d'utiliser la bibliothèque `RCurl` qui offre plus d'options. Dans ce cours, nous nous limitons à l'utilisation de `read.csv`. Pour plus d'informations sur cette bibliothèque, vous pouvez lire plus de détails la documentation de ce *package*.

```
In [23]: install.packages("RCurl")

Updating HTML index of packages in '.Library'
Making 'packages.html' ... done


In [24]: library(RCurl)

Loading required package: bitops


In [26]: x <- getURL("https://raw.githubusercontent.com/aronlindberg/latent_growth_classes/maste
         y <- read.csv(text = x)

In [28]: head(y)
```

top100_repository_name	month	monthly_increase	monthly_begin_at	monthly_end_with
Bukkit	2012-03	9	431	440
Bukkit	2012-04	19	438	457
Bukkit	2012-05	19	455	474
Bukkit	2012-06	18	475	493
Bukkit	2012-07	15	492	507
Bukkit	2012-08	50	506	556

2 Traiter les valeurs manquantes

Afin d'illustrer le traitement des valeurs manquantes dans R, importons les données de l'exemple 2_2

```
In [7]: read.table("https://raw.githubusercontent.com/nmeraihi/data/master/exemple_2_2.txt", hea
```

km	temps	vitesseMoyenne	puissanceMoyenne	bpm
1.24	4:01	19.1	160	134
NA	9:42	30.2	133	146
1.02	1:57	30.8	141	139
17.61	36:11	29.2	NA	144
9.27	19:10	29.0	121	143

On peut appliquer un test booléen afin de vérifier l'existance des valeurs manquantes comme suit;

```
In [8]: df<-read.table("https://raw.githubusercontent.com/nmeraihi/data/master/exemple_2_2.txt",
        is.na(df)
```

km	temps	vitesseMoyenne	puissanceMoyenne	bpm
FALSE	FALSE	FALSE	FALSE	FALSE
TRUE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE

Cette fonction nous retourne un *data frame* du même format que le *df* test. Le résultat obtrenu sont valeurs `TRUE` sur les éléments manquants, et des `FALSE` sur les valeurs existantes.

On peut faire le test sur une partie précise du *df*;

```
In [9]: is.na(df[1,1])
```

FALSE

```
In [10]: is.na(df[2,1])
```

TRUE

Mais pourquoi préocupe t-on tant des valeurs manquantes? Eh bien, les valeurs manquantes sont le cauchemar #1 de toute personne qui manipule les données, que ce soit en entreprise ou pour un utilisation personnelle.

Essayons de faire un calcul de la moyenne du nombre de km;

```
In [11]: mean(df$km)
```

```
[1] NA
```

R nous retourne NA même si nous avons une seule observation qui est manquante
On peut régler ce problème avec la fonction na.omit()

```
In [12]: mean(na.omit(df$km))
```

7.285
Le calcul de la moyenne a été fait sur les variables;

```
In [13]: na.omit(df$km)
```

1. 1.24 2. 1.02 3. 17.61 4. 9.27
La fonction mean possède un argument optionnel appelé na.rm = qui ignore les valeurs manquantes;

```
In [14]: mean(df$km,na.rm = T)
```

7.285
Lorsque nous utilisons la fonction ns.omit, le df se réduit à un df qui ne possède aucune ligne contenant les valeurs manquantes;

```
In [15]: na.omit(df)
```

	km	temps	vitesseMoyenne	puissanceMoyenne	bpm
1	1.24	4:01	19.1	160	134
3	1.02	1:57	30.8	141	139
5	9.27	19:10	29.0	121	143

On peut aller modifier directement la valeur de cet élément

```
In [16]: df[2,1]<-4.84
          df[4,4]<-125
```

```
In [11]: df
```

km	temps	vitesseMoyenne	puissanceMoyenne	bpm
1.24	4:01	19.1	160	134
4.84	9:42	30.2	133	146
1.02	1:57	30.8	141	139
17.61	36:11	29.2	125	144
9.27	19:10	29.0	121	143

2.1 Remplacer toutes les valeurs manquantes:

Des fois, il peut être utile de remplacer toutes les valeurs manquantes par des 0. Je dis bien des fois, car les valeurs manquantes sont absentes et non des 0.

```
In [18]: df<-read.table("https://raw.githubusercontent.com/nmeraihi/data/master/exemple_2_2.txt"
         df
```

km	temps	vitesseMoyenne	puissanceMoyenne	bpm
1.24	4:01	19.1	160	134
NA	9:42	30.2	133	146
1.02	1:57	30.8	141	139
17.61	36:11	29.2	NA	144
9.27	19:10	29.0	121	143

Nous remplaçons alors les `NA` par `0` ou par toute autre valeur comme suit;

```
In [19]: df[is.na(df)] <- 0
```

```
In [20]: df
```

km	temps	vitesseMoyenne	puissanceMoyenne	bpm
1.24	4:01	19.1	160	134
0.00	9:42	30.2	133	146
1.02	1:57	30.8	141	139
17.61	36:11	29.2	0	144
9.27	19:10	29.0	121	143