Notes de cours R

Thibaut Marmey

October 19, 2018

Contents

1 Programmation R

1.1 Généralité

- *lien internet :* reprise cours OCR
- *lien internet :* doc d'installation de R et Rstudio
- Utilisation de la documentation "aide" : *help()* ou *help("nom de la fonction")* ou *?log*
- Documentation plus détaillée sur internet : *help.start()*
- Interaction avec l'environnement R :
  - liste toutes les variables crées : *ls()*
  - liste des variables avec une succession de lettre particulière : *ls(pattern="mot")*
  - supprimer des variables : *rm(var)*
  - quitter le travail : *q()* ou *quit()*
- Ecrire des commentaires avec "#"

- Utiliser la fonction *print()* pour afficher les informations, textes sur la console
- Enregistrer les variables que l'on veut :
  save(var1, var2, ..., file = "nameFile.extension").
  Enregistrement de ces fichiers dans le répertoires associés *Tools/Global Options*, ici c'est Documents/Notes-de-cours/Cours R/Test

1.2 Fonctions

- Tester le type d'une variable :
  - *is.character(var)*
  - *is.numeric(var)*
  - *is.logical(var)*
  - *is.vector(var)*
  - *is.factor(var)*
  - *is.list(var)*
- Spécifier le typage de la variable :
  - *as.numeric(var)*
  - *as.logical(var)*
  - *as.character(var)*
- Renvoyer l'entier inférieur : *floor(nb)*
- Renvoyer l'entier supérieur : *ceiling(nb)*
- Arrondir à l'entier le plus proche : *round(nb)*
- Racine carré : *sqrt()*
- Fonction trigo : *cos(angleRad), sin, ...*
- Fonction somme : *sum(vect)*
- Longueur vecteur : *length(vect)*

1.3 Manipulation de chaînes de caractères

- Saisir des données depuis le clavier : *scan()*. Pour valider l'input appuyer sur *entrée*, et un nouvel input appararaîtra. Pour arrêter la prise d'input appuyer directement sur *entrée*.
- Rentrer le nombre d'input avec l'appel de *scan(nmax=n)*
- La concaténation de texte et nombre avec *paste(texte, nombres, ...)*
- Combinaison de *scan()* et *paste()* : *paste('Bonjour j'ai ', scan(nmax=1), ' ans')*

- Permet de compter le nombre de lettres présentes dans la chaîne de caractères : *nchar("chaîne")*
- Mettre les caractères en majuscules ou minuscules : *toupper(), tolower()*
- Extraire une sous chaîne de caractères avec point de départ et d'arrivée inclus : *substr("chaine", start, stop).* La chaîne de caractères commence à 1.

1.4 Gestion de données complexes

- Le vecteur : élément de base du langage, c'est une liste d'éléments étant tous du même type. Il possède 3 attributs :
  - son type des éléments
  - sa longueur
  - les noms associés aux différents éléments
  - Spécifier deux attributs : le type de ces éléments et la longueur du vecteur (nb d'éléments)
  - Les fonctions prennent toutes des vecteurs en paramètre. Elles renverrons le résultat sous la forme d'un vecteur de longueur (généralement) égale à la longueur du vecteur d'entrée.
  - Différentes méthodes pour créer un vecteur :
    - la fonction *vector(type, length)*
      ex : vector("numeric", 10) crée un vecteur de 10 éléments tous égaux à 0. Les valeurs par défaut sont 0 (numeric), "" (character), FALSE (logical).
    - les fonctions *numeric(length), character(length), logical(length)*
  - Générer des séries de nombres
    - Suite d'un nombre à un autre : *nb1:nb2*
    - Suite d'un nombre répété : *rep(element, length)*
    - Séquence de nombres : *seq(start, stop, step)*
- Concaténer plusieurs vecteurs (ils doivent contenir les mêmes types de variables) : *c(vect1, vect2, ...)* (bonne méthode pour créer rapidement un vecteur avec des valeurs directement attribuées ex: *c(70, 50, 10, 0, 5)*)
- Nommer les éléments du vecteur :
  names("nom1", "nom2", rep(NA,2), "nom3", ...) (ici permet de nommer des éléments et d'en laisser 2 sans nom)
- Indexation numériques : travailler seulement sur une sous partie d'un vecteur. Les éléments des vecteurs sont indéxés de 1 jusqu'au dernier.
  - Accéder à un élément du vecteur via son index : *vect1[index]*
  - Accéder à plusieurs éléments du vecteur :
    vect1[nb1:nb2], vect1[c(index1, index2, ...)] il est en fait possible d'utiliser toutes les fonctions de séquences ou listes de nombres comme index.

  - Utiliser des booléens pour sélectionner les éléments à récupérer :
    vect1[vect1 ⟩ 7] (on récupère ici tous les éléments de vect1 qui sont supérieur à 7).
  - Utiliser les noms des éléments : *vect1[c("nom1", "nom2")]*

- Modifier un élément du vecteur : *vect1[index] = valeur* (sachant que index peut être des fonctions telles que *c()*, *nb1:nb2*, etc...

1.5 Opérations sur des vecteurs

- Addition sur vecteur : *vect1 = vect1 + 1* (+1 à tous les éléments)
  Dans ce cas là, R prend le plus grand vecteur pour y effectuer l'opération à partir du vecteur plus court.
  vect1 = vect1 + c(nb1, nb2,...) (revoie d'un avertissement si le grand vecteur n'est pas un multiple du petit vecteur)

- Connaître la longueur d'un objet : *length()*

- Sélectionner seulement certains éléments :
  - Les premiers éléments : *head(nb d'éléments)*
  - Les derniers éléments : *tail(nb d'éléments)*

- Triez les vecteurs :
  - *sort()* renvoie un nouveau vecteur contenant les mêmes éléments mais triés dans l'ordre.
    Dans l'ordre croissant : *sort(vect1)*
    Dans l'ordre décroissant : *sort(vect1, decreasing=True)*
    Dans le cas des caractères, la priorité est donnée aux majuscules, ainsi un "a" sera placé après un "Z". Pour contourner ce problème, utiliser les fonctions *toupper()* ou *tolower()*.
  - *order()* renvoie l'ordre des éléments via leur indice.
    ⟩ [1] 2 5 4 1 3 (ce résultat indique que l'élément d'index 2 est le plus petit, suivi de l'index 5, 4, 1 et 3)
  - *rank()* renvoie le rang de l'élément au sein de la distribution. Par défaut le *ties.method* est *average* ce qui implique que si plusieurs éléments sont de même rang la fonction attribuera ainsi la moyenne de ses rangs.

1.6 Analyses statistiques sur des vecteurs

- Analyser la distribution d'un vecteur consiste à calculer diférentes valeur sur celui ci : la moyenne, la médiane ou la variance par exemple.

- Calculer la moyenne : *mean(vect1)*

- Calculer la médiane : *median(vect1*
  La médiane permet de répartir deux parties en un même nombre d'élément. C'est en fait la valeur centrale de la distribution.

- En présence de valeurs NA dans la distribution, utiliser l'argument facultatif : *na.rm = TRUE.*

- Mesurer la dispersion d'une distribution :

  - les fonctions *min()* et *max()*. Utiliser *na.rm* si présence de NA.
  - Les quantiles sont les valeurs permettant de séparer une distribution ordonnée de valeurs en q sous-distributions.
    Généralement on calcule les quartiles c'est à dire on sépare la distribution en 4-quantiles.
    Utiliser la fonction *quantile(vect1)*
  - La fonction tout en un : *summary(vect1)* renvoie :
    * la valeur min
    * le premier quartile (25%)
    * la médiane (ou second quartile (50%)
    * la moyenne
    * le troisième quartile (75%)
    * la valeur max
  - La variance mesure la dispersion par rapport à la moyenne. Elle prend une valeur grande si les éléments de la distribution sont généréralement éloignés de la moyenne ou une valeur petite si ces éléments sont au contraire resserrés près de la moyenne.
    Calcul de la variance : *var(vect1)*
    Calcul de l'écart-type : *sd(vect1)*

1.7 Manipuler et comparer plusieurs vecteurs

- Faire l'union de deux deux ensembles : *union(vect1, vect2)*
  union(names(vect1), names(vect2)) (cette fonction retourne l'ensemble des éléments se trouvant dans au moins un de ces vecteurs)

- Faire l'intersection des ensmbles pour savoir quels sont les éléments présents dans les tous les vecteurs en argument : *intersect(vect1, vect2)*

- Isoler les éléments qui ne sont pas commun aux vecteurs placés en argument : *setdiff(x, y)* (renvoie les éléments de x qui ne sont pas dans y)
  Cette fonction est asymétrique c'est à dire que l'ordre des arguments modifie le résultat en sortie.

- L'opérateur *%in%* renvoie un vecteur d'éléments logiques de la même taille que le premier vecteur référencé à son appel : *x %in% y*
  Utilisation : *vect1[names(vect1) %in% vect2]*

- Ordonner les vecteurs de manière réciproque : certaines fonctions ont besoin de vecteurs qui sont ordonnées dans le même ordre.

  - Commencer par avoir un vecteur de nom obtenu par :
    commun = intersect(names(vect1), names(vect2))

  - Utiliser ce vecteur de nom commun pour récupérer dans le même ordre les éléments de vect1 et vect2
    vect1-2 = vect1[commun]
    vect2-2 = vect2[commun]
    Ainsi les deux vecteurs *vect1-2* et *vect2-2* sont ordonnés de la même manière avec le même nombre d'éléments.

- Trier ses données. Lorsque l'on veut trier un des vecteurs, on veut garder la réciprocité de l'ordre des éléments avec l'autre vecteur. Pour cela on veut récupérer les indices des éléments qui sont triés.

  - Utiliser la fonction *order()*. Utiliser le même vecteur avec order :
    vect1-tri = vect1-2[order(vect1-2)]
    vect2-tri = vect2-2[order(vect1-2)]
    On obtient alors deux vecteur dont les éléments sont classés et dans même ordre.

1.8 Corrélations et régression linéaire

- Calculer la corrélation entre deux variables revient à calculer l'intensité de leur liaison. Ce coefficient est compris entre -1 et 1. Plus le coefficient est proche de ces valeurs extrèmes, plus la liaison est forte. Si à l'inverse le coefficient est proche de 0, la liaison est faible ce qui veut aussi dire que les données sont linéairement indépendantes.

- Il est possible que ces données soient corrélées de manière non linéaire.

  - Un coefficient entre 0 et 1 indique que les deux séries de données évoluent dans le même sens.
  - Inversement, si le coefficient est compris entre -1 et 0.

- Calculer un **coefficient de corrélation linéaire** : *cor(x,y)*
  Attention : x et y doivent être de la même taille et contenir des éléments numériques
  Les deux données doivent être réciproques (premier élément de x correspond au premier élément de y et ainsi de suite)

- Calcul des coefficients de la **régrétion linéaire** $y = a*x + b$
  Utiliser la fonction : *lm()*
  Cette fonction prend comme premier argument une formule : *formula = y x* (étudier le comportement de y en fonction de x)
  Le premier nombre obtenu est b (*Intersect*), le second est a.

- Les tests statistiques ont pour but d'infirmer ou confirmer une hypothèse de départ aussi appelé hypothèse nulle (H0). L'hypothèse nulle consiste généralement à postuler une égalité entre les variables comparées.

- Le test statistique renverra une p-value, probabilité que la différence observée entre les deux obstervations soit due au hasard. Seuil pour que la différence observée n'est pas du au hasard : généralement 5%.
  Si la p-value est inférieure à 0.05, rejet de l'hypothèse nulle pour accepter hypothèse alternative : la différence entre les deux variables est significative.

- Test de corrélation : *cor.text(vect1, vect2)*
  - Lire les résultats de bas en haut
  - Calcul du coefficient de corélation *cor*
  - *95 percent confidence interval* : intervalle de confiance de ce coefficient de corrélation.
  - Calcul de la p-value
  - On accepte ou non l'hypothèse nulle
- Test de médianes avec le test de Wilcoxon-Mann-Whitney : *wilcox.test(vect1, vect2)*
  Connaître s'il y a une différence significative en comparant les médianes puisque la médiane est une mesure qui est moins affectés par d'éventuelles valeurs abbérantes.

1.9 Gestion de données avec les facteurs

- Les facteurs permettent de stocker une série d'éléments comme les vecteurs mais possèdent leurs propres attributs et propriétés.
- Ils sont généralement utilisés pour stocker des variables qualitatives (valeur descriptive et non numérique)
- Ils peuvent stocker des éléments de types différents
- Ils possèdent un attribut dit *levels* : vecteur qui contient l'univers des données (l'ensemble des valeurs que peut prendre les éléments)
- Créer un facteur : *factor()* :
  fact1 = factor(c("A","B","A","C"))
  Affichage : *[1] A B A C*, *Levels : A B C* (création automatique de l'attribut *levels*)
- Pour spécifier les valeurs de *levels* : *factor(c(...), levels=c("A","B",...,"F"))*
- Utilisation des facteurs (liste des manip identiques aux vecteurs) :
  - Nommer les éléments : *names()*
  - Obtenir la longueur du facteur : *length(fact1)*
  - Accéder ç un élément par l'index : *fact1[index]*
- Obtenir l'attribut *levels* d'un facteur : *levels(fact1)*
- Rajouter un élément dans *levels* : *levels(fact1) = c(levels, "G")*
- Pas possible de faire d'opérations arithmétiques sur les facteurs puisqu'ils contiennent des données de types qualitatives.
- Accéder à des éléments particuliers : *head()*, *tail()*
- Si le facteur ne possède que des valeurs numériques il est interessant de travailler sur un vecteur.
  Convertir un facteur en vecteur : *vect1 = as.vector(fact1)*

- Convertir un vecteur en facteur : *fact1 = as.factor(vect1)*
- Attention à la convertion d'un facteur d'éléments numériques en vecteur en utilisant la fonction *as.numeric()*.
  En utilisant *as.numeric()* renvoie alors la version numérique de l'identifiant interne des éléments du facteur.
  Utiliser : *vect1 = as.numeric(as.character(fact1))*

1.10 Les matrices

- Pour créer un matrice il faut que :
  - Les éléments soient du même type
  - Toutes les colonnes doivent avoir la même taille
  - Toutes les lignes doivent avoir la même taille
- Créer la matrice :
  matrix(nrow=nr, ncol=nc, data=vect1, byrow=TRUE/FALSE)
  matrix(c(vect1, vect2), nrow=nr, ncol=nc, byrow=T) (pas besoin de spécifier *data* si c'est le premier argument, raccourcir *TRUE* en *T*)
  - *nrow :* nombre de lignes
  - *ncol :* nombre de colonnes
  - *data :* vecteur des données de la matrice
  - *byrow :* méthode de remplissage de la matrice par data
- Il est préférable que le nombre d'éléments dans *data* soit égal à la capacité de la matrice (*nrow*ncol)*.
  - Si *data* possède moins d'éléments que la capacité de la matrice, les éléments sont répétés pour la compléter.
  - Si *data* possède plus d'éléments, les éléments en trop sont tronqués.
- Nommer les lignes : *rownames(mat1)*
- Nommer les colonnes : *colnames(mat1)*
- Indexation : *mat1[l,c]*
  Avec des noms : *mat1["nom1", "nom2"]*
  Attention, il ne faut pas avoir des noms identiques entre les lignes et les colonnes
- Sélection de plusieurs éléments : *matrice(c(...),c(...)*
  matrice(a:b, c:d)
  matrice[sort(rownames(matrice))[1:3], colnames(matrice)[c(1,3)]]
- Sélectionner toutes les lignes : *matrice[,y]*
  Sélectionner toutes les colonnes : *matrice[x,]*
  Sélectionner toutes les lignes et colonnes : *matrice[,]*

1.11 Les tableaux de données (dataframes)

- Les tableaux de données (dataframes en anglais) ressemblent beaucoup aux matrices. Ils ont pour vocation initiale de stocker différentes variables pour différents individus ou observations d'une population ou expérience.
- Les éléments d'une ligne ou d'une colonne peuvent être de types différents.

	age	sexe	observation
Raoul	21	M	BSTR
Fred	22	M	18
Dominique	20	F	85
Didier	22	M	PMF
Rachelle	21	F	TRUE

- Créer des tableaux de données : *data.frame()*, les arguments :
  - autant de vecteurs que de colonnes (le remplissage des dataframes se fait en colonne).
  - (optionnel) *row.names* pour spécifier le nom des lignes
  - (optionnel) *stringsAsFactors = TRUE/FASLE*, compostement à utiliser avec les vecteurs contenant des chaînes de caractères
    - *TRUE :* convertion auto de vecteurs en facteurs
    - *FALSE :* ces vecteurs restent des vecteurs
- Autre déclaration :
  data.fram(taille=vect1, poids=vect2, ..., row.names = c("A","B",...))
- Comme les éléments d'une même colonne peuvent avoir des types différents il faudra parfois utiliser des facteurs et non des vecteurs pour spécifier le contenu d'une colonne.
- Accéder aux éléments des dataframe : mêmes méthodes que les matrices
- Méthodes spécifiques aux dataframes :
  - Extraire une colonne : *dataframe$colonne*

1.12 Les listes

- Une liste est un ensemble d'éléments ordonnés les uns à la suite des autres dans une même structure indexée. Ces éléments peuvent être de types différents.
- Leur utilité est de regrouper dans un même objet une série d'autres objets appartenant par exemple à une même expérience ou observation. Il est aussi possible de traiter tous les éléments d'une même liste en une seule fois grâce à des fonctions adaptées.
- Les lignes ne sont pas nécessairement de la même longueur
- Traiter des jeux de données différents (nombre, type, etc) sans avoir à répéter la même procédure à chaque fois.

- Représentation d'une liste sous R (*ex :* Présence de vecteurs de type numérique qui sont associés à un index dont le nom commence par un dollar ($))

```
> athletes
$Didier
[1] 630 625 628 599 635 633 622

$Jules
[1] 610 590 595 582 601 603

$Pierre
[1] 644 638 639 627 642 633 639

$Matthieu
[1] 622 625 633 641 610
```

- Créer une liste : *list(nom1=vect1, nom2=vect2,...)*
- Longueur de la liste (nombre de lignes) : *length(liste)*
- Noms de la liste (nom des lignes) : *names(liste)*
- Indexation particulière puisque présence de double crochet : *liste[[index/nom]]*
  Les index peuvent être les indice des éléments ou les noms associés aux éléments.
- Accéder à une ligne : *liste$nomLigne*
- Appliquer une même fonction aux différents éléments de la liste :
  lapply(x, FUN, ...) (renvoie une liste)
  - *x :* nom de la liste à traiter
  - *FUN :* le nom de la fonction à appliquer à tous les éléments de *x*
  - *... :* arguements optionnels seront les arguments additionnels que peut prendre *FUN*, par exemple *na.rm*
- Même utilisation que *lapply()* mais qui renvoie les résultats sous la forme d'objet plus utilisable comme vecteurs, matrices ou dataframes : *sapply()*

- Grâce à *sapply()* il est alors possible d'utiliser des fonctions telles que *summary* :
  sapply(liste, summary)
  (possible avec *lapply()* mais les résultats ne sont pas bien visibles