# R studio

## Rappel

### Les types de variables

* **Qualitative nominale :** Ce type de variable représente un nom ou une catégorie comme par exemple la variable Variete dans l’ensemble de données sur les melons.
* **Qualitative ordinale :** Ce type de variable représente des valeurs numériques ordonnées comme par exemple la variable Creneau dans l’ensemble de données sur les melons.
* **Quantitative discrète :** Ce type de variable représente une quantité avec des valeurs isolées comme par exemple la variable Plantation dans l’ensemble de données sur les melons.
* **Quantitative continue :** Ce type de variable représente une quantité avec des valeurs non isolées comme par exemple la variable Poids dans l’ensemble de données sur les melons.

## Bases de R

### Installer un package directement depuis internet

**install.packages("nom du package")**

### Corriger la nature d’une variable

| **Rôle** | **Commande R** |
| --- | --- |
| Convertir en variable qualitative nominale | as.factor() |
| Convertir en variable qualitative ordinale | as.ordered() |
| Convertir en variable quantitative discrète | as.integer() |
| Convertir en variable quantitative continue | as.numeric() |

Rmq : une variable quantitative est reconnue soit en type integer (s’il n’y aucun nombre à virgules dans les valeurs de cette variable), soit en type numeric (s’il y a au moins un nombre à virgules), mais dans les deux cas la variable sera bien considérée comme une variable quantitative.

### Transformation d’un jeu de données : transform()

Ajouter une variable à une base de données par exemple :

« base de données » <- transform (« base de données », « variable à ajouter »)

### Extraire un sous ensemble de données

Accéder à un sous-ensemble de variables et/ou d’individus : [,] utilisation avant la virgule pour les individus et après la virgule pour les variables.

| **Rôle** | **Commande R** |
| --- | --- |
| Extraire un sous-ensemble d’individus | base[n°lignes, ] |
| Extraire un sous-ensemble de variables | base[, n°colonnes] |
| Extraire un sous-ensemble d’individus et de variables | base[n°lignes, n°colonnes] |
| Extraire une variable | base$nom\_variable |
| Supprimer un sous-ensemble d’individus | base[-n°lignes, ] |
| Supprimer un sous-ensemble de variables | base[, -n°colonnes] |
| Supprimer un sous-ensemble d’individus et de variables | base[-n°lignes, -n°colonnes] |

### Sauvegarder un jeu de données

La fonction **write.csv2()** permet de sauvegarder une base de données dans un fichier CSV.

Comme pour la fonction read.csv2(), l’argument file indique le nom du fichier. L’argument row.names permet de préciser si l’on souhaite que le numéro des lignes soit présent dans le fichier.

Par ex : write.csv2(Melons, file = "DonneesMelons2.csv", row.names = FALSE)

### Ce qu’il faut toujours faire quand on importe ou crée un jeu de données

Une fois qu’un jeu de données est importé, la bonne démarche pour l’analyser se résume en 4 étapes :

* Vérifier les types de variables (quantitatives/qualitatives) à l’aide de la fonction str().
* Si besoin, corriger la nature des variables à l’aide des fonctions as.factor(), as.ordered(), …
* Analyser le jeu de données (extraction d’individus, de variables, calcul de statistiques).
* Si besoin, sauvegarder le jeu de données modifié à l’aide la fonction write.csv2()

## Calculer le coefficient de corrélation entre deux variables

**cor(x, y, method = c("pearson", "kendall", "spearman"))**

Avec x et y qui sont 2 variables. La méthode de corrélation de pearson calcule un coefficient de corrélation appelé paramétrique. Les méthodes de test de corrélation de kendall et de spearman sont non paramétriques. Ce sont des tests de corrélation basés sur le rang.

*Ex d’utilisation de la méthode de pearson : cor(x,y, method="pearson") / Résultat : [1] 0.5712*

Dans le cas où la méthode utilisée est de type “kendall” ou “spearman”, les statistiques tau de kendall et rho de Spearman sont respectivement utilisées pour estimer le coefficient de corrélation basé sur le rang. Ce sont des tests statistiques dits robustes car ils ne dépendent pas de la distribution des données. Le test de corrélation de Kendall et celui de Spearman est recommandé lorsque les variables ne suivent pas une loi normale.

Comment savoir si une variable suit une loi normale ? Test de normalité sur R

* **Le test de Shapiro-Wilk :** c’est un test permettant de savoir si une série de données suit une loi normale.

**shapiro.test(x)**

*Ex : shapiro.test(rnorm(100, mean = 5, sd = 3)) / Résultat : W = 0.9895, p-value = 0.6211*

L'exemple ci-dessus renvoie une p-value non significative (P-value>0,05). L'échantillon suit donc une loi normale.

* **Test de Kendall :** Le calcul du coefficient de corrélation de Kendall, basé sur un test de rang, pourrait être utilisé lorsque les données ne proviennent pas normalement d’une distribution normale.

**res<-cor.test(x,y, method="kendall")**

*ex de résultat :*

*Kendall's rank correlation tau*

*data: x and y*

*T = 26, p-value = 0.1194*

*alternative hypothesis: true tau is not equal to 0*

*sample estimates:*

*tau*

*0.4444*

tau est le coefficient de corrélation de Kendall.

* **Test de Spearman :** La statistique rho de Spearman peut être aussi utilisée pour estimer une association, basée sur un test de rang, entre deux variables. Comme le test de kendall, le test de spearman pourrait être utilisé lorsque les données ne proviennent pas d’une distribution normale.

**res<-cor.test(x,y, method="spearman")**

*ex de résultat :*

*Spearman's rank correlation rho*

*data: x and y*

*S = 48, p-value = 0.0968*

*alternative hypothesis: true rho is not equal to 0*

*sample estimates:*

*rho*

*0.6*

rho est le coefficient de corrélation de Spearman.

## Fonctions diverses

*Source :* [*http://adv-r.had.co.nz/Vocabulary.html*](http://adv-r.had.co.nz/Vocabulary.html)

**apply(X, MARGIN, FUN, ...)** Renvoie un vecteur ou un tableau ou une liste de valeurs obtenues en appliquant une fonction aux marges d'un tableau ou d'une matrice. x, le tableau ou la matrice. margin, un vecteur donnant les indices sur lesquels la fonction sera appliquée. Fun, la fonction appliquée.

**cbind()** Mettre de séquence d’un vecteur, d’une matrice ou d’un cadre de donner et les associer en colonne.

**colnames()<- value** Remplacer le nom des colonnes par le vecteur voulu. Pareil pour rownames

**count(x$y)** Trouver le nombre de fois que se répète la valeur d’ne variable (package plyr)

**crop(x,y)** rend un sous-ensemble géographique d'un objet comme spécifié par un objet de Mesure (ou l'objet dans lequel un objet de mesure peut être extrait ou créé). Si x est un objet Raster, la mesure est alignée à x. Les zones incluses dans y mais à l'extérieur de la mesure de x sont ignorées (voir s'étendent si vous voulez une plus grande zone).

**dir=** Ouverture d’un répertoire.

**duplicated ()** détermine quels éléments d'un vecteur ou d'une trame de données sont des doublons d'éléments avec des indices plus petits, et renvoie un vecteur logique indiquant les éléments (lignes) dupliqués.

**Function ()** "fonctions anonymes" qui sont de vrais objets de fonction mais qui n’ont été attribués à aucun symbole avant leur utilisation. Généralement, les fonctions sont attribuées à des symboles, mais ce n’est pas nécessaire. La valeur renvoyée par l'appel à function est une fonction. Si on ne lui donne pas un nom, on parle de fonction anonyme. Les fonctions anonymes sont le plus souvent utilisées comme arguments à d'autres fonctions telles que la famille apply ou outer.

*Ex : (function(x,y){x\*y + x/y})(2,5)*

*# [1] 10.4*

**grep(pattern,x)** Permet d’aller chercher dans le vecteur x, les valeurs ou les noms correspondant au pattern, c’est-à-dire à la variable que nous voulons associer à x.

**gsub** remplacent respectivement le premier match (correspondances) et tous les matchs.

**is.na ()** indique les éléments manquants.

**last (x,)** Renvoie le dernier élément d'un vecteur ou d'une liste, ou la dernière ligne d'un fichier data.frame ou data.table.

**list.files()** Ces fonctions produisent un vecteur de caractère des noms de fichiers ou de répertoires dans le répertoire nommé.

**melt(data, id.vars, measure.vars)** Fonction provenant du package reshape qui permet de passer facilement d'un format de données «long» (une ligne par individu et par variable) à un format «large» (une seule ligne par individu). id.vars est une liste de variable d'identifiants qui sont les variables qui vont rester en « long ». measure.vars est une liste de variables de mesure qui sont les variables qui vont passer en « large ».

**merge(x, y, by = intersect(names(x), names(y))** Permet de fusionner 2 data frames qui ont des variables communes. x et y sont les data frames à fusionner et « by = » spécifie la colonne en commune qui va permettre la fusion. La fonction intersect cherche toutes les colonnes communes entre les 2 tables pour les fusionner. L’option **all = TRUE** permet de garder les lignes de chaque table qui ne sont pas présentes dans l’autre ajoutant NA dans les cellules des variables de l’autre tableau.

**na.omit(object, ...)** Permet de supprimer les lignes avec des NA.

**Object$variable <- «»** Permet d’ajouter une colonne vide

**paste()** Enchainé des vecteurs après les avoir converti en caractères. Cela permet de créer la direction et un nom de fichier. *Ex : INDIC\_fp=paste(IndicN,INDIC,sep="/")*

**paste0()** Pareil que la fonction précédente mais le 0 permet d’enlever tous les espaces présent dans le nom du fichier. *Ex : fic= paste("D:/Mairi/RastersLSf\_Ind\_zones/LSfIND\_",zone,".tif")*

**pattern= « »** permet de sélectionner un format de fichier.

**Plot(« nom axe des ordonnées » ~ « nom axe des abscisses », mean= « nom du jeux de données)** Permet de choisir les axes dans son graphique

**print()** Montre dans la console les étapes d’actions du programme normalement invisibles quand on il lit le sript.

**rbind()** Mettre de séquence d’un vecteur, d’une matrice ou d’un cadre de donner et les associer en ligne.

**Rownames()<- value** Remplacer le nom des lignes par le vecteur voulu.

**setdiff(x,y)** Permet de voir les différences entre 2 vecteurs portant le même nom. Il va montrer les lignes apparaissant dans x mais pas dans y. Si on change l’ordre x,y, il va montrer les lignes apparaissant dans y mais pas dans x.

**setnames(raster,vecteur de noms)** Permet de renommer les couches du raster avec les noms du vecteur. Il faut que le raster et le vecteur soient égales numériquement.

**setwd()** Est utilisé pour mettre le répertoire de travail au dir.

**shift(object, x=, y=)** Permet de décaler l’objet sur de nouvelles coordonnées.

**str()** verification du type de variable :

| **Type de variable** | **Résultat de str()** |
| --- | --- |
| Qualitative nominale | Factor |
| Qualitative ordinale | Ord.factor (ordered.factor) |
| Quantitative discrète | int (integer) |
| Quantitative continue | num (numeric) |

**stringsAsFactors = FALSE** Éviter des problèmes retardent le re-codage de cordes (séries) quand on crée des data.frames.

**strsplit** Diviser les éléments d’un vecteur pour en prendre qu’une partie.

**subset(x,subset)** Renvoie des sous-ensembles de vecteurs, matrices ou trames de données répondant à des conditions. x est l’objet à modifier. Subset, expression logique indiquant les éléments ou les lignes à conserver: les valeurs manquantes sont considérées comme fausses.

**Substr (x, start, stop)** Extraire ou remplacer une partie des caractères d’un vecteur.

**toupper(variable nominale)** Permet de retrouver les lettres de cette variable qu’elles soient en majuscule ou en minuscule.

**Tstrsplit ()** Cela équivaut à transposer (strsplit (...)). Il s'agit d'une fonction d'encapsulation commode permettant de fractionner une colonne à l'aide de strsplit et d'affecter le résultat transposé à des colonnes individuelles.

**unique()** Trouver les différentes modalités (valeurs) que peut prendre la variable

**which(), wich.max(), which.min()** permettent de faire apparaitre la colonne ou la ligne souhaitée

## Fonctions Boucles

Les boucles doivent être évité dès qu’une syntaxe, impliquant des calculs matriciels ou les commandes de type apply, peut se substituer.

### Structures conditionnelles « if »

* **if(condition){instructions}** est la syntaxe permettant de calculer les instructions uniquement si la condition est vraie.
* **if(condition){ A }else{ B }** calcule les instructions A si la condition est vraie et les instructions B sinon.

### Structures itératives

Ces commandes définissent des boucles pour exécuter plusieurs fois une instruction ou un bloc d’instructions. Les trois types de boucle sont :

* **for var in seq) {commandes}** (seq=vecteur)
* **while (condition) {commandes}**
* **repeat {commandes ; if (condition) break }**

Dans une boucle for, le nombre d’itérations est fixe alors qu’il peut être infini pour les boucles while et repeat ! La condition est évaluée avant toute exécution dans while alors que repeat exécute au moins une fois les commandes.

### Boucle avec une matrice

Le problème de la matrice est qu’elle est réversible. Il va falloir choisir si on veut prend en compte les lignes ou les colonnes pour faire la boucle :

Soit : **for(row in 1:nrow(my\_matrix)) {**

Soit : **for(col in 1:ncol(my\_matrix)) {**

## Package Raster

Il faut avoir le package Raster. Un **Rasterlayer** (veut dire que toutes les bandes sont fusionées en une seule couche) peut être créée à partir de rien, d’un dossier, d’une matrice, … Dans des nombreux cas, par exemple quand un RasterLayer est créé d'un fichier (dossier), il ne contient pas initialement des valeurs de cellule (le pixel) dans (la RAM) la mémoire (le souvenir), il a seulement les paramètres qui décrivent le RasterLayer.

* RasterLayer -> raster(x) -> il y a une seule couche. Une seule bande spectrale
* RasterStack -> raster(x, layer=0) avec stack=pile donc c’est un raster à plusieurs couches. Les bandes spectrales sont distinctes.
* RasterBrick -> raster(x, layer=0) avec brick=brique donc les bandes sont distinctes mais collées.

### Enregistrer un raster

**writeRaster(nom du raster à enregistrer, bylayer=T/F, format= « », filename=« », overwrite=T/F)** Enregistrer un raster. « **bylayer**» signifie enregistrer un raster pour chaque couche, si on choisit faux, ça va enregistrer toutes les couches sur le même raster. « **filename** » représente la localisation du raster et son nom dans le dossier, *par ex : "D:/Mairi/RastersLSf\_Ind\_zones/LSfIND\_",zone,".tif".* « **overwrite** » permet d’enregistrer le raster en écrasant le raster précédent du même nom.

Quand on enregistre un raster sous format ENVI, le fichier format hdr s’ajoute automatiquement dans le dossier. C’est le fichier qui permet de lire les informations ENVI sous forme de texte.

### Différence entre Raster Brick et Raster Stack

Ce sont deux classes de raster multicouches. Le raster stack peut être facilement formé d’un assemblage de raster layer càd de raster avec une unique couche venant de dossiers différents. Le raster brick est plus efficient que le raster stack, c’est pourquoi il vaut mieux enregistrer les rasters sous format brick. Cependant ce type de raster peut seulement fusionner des rasters monocouche d’un seul dossier à la différence du raster stack.

### Mettre un raster au format stack ou brick dans R

**Nom de la variable(value) = stack(nom du raster)**

**Nom de la variable(value) = brick(nom du raster)**

### Modification de la résolution d’un raster

Si l'un de vos rasters présente une résolution plus élevée que les autres, pensez peut-être à ré-échantillonner ce raster selon la résolution des rasters plus grossiers, de manière à obtenir des jeux de données raster avec une résolution uniforme. Cela permet d'accélérer le traitement et de réduire la taille des données. Contrairement à la définition de la taille de cellule dans l'environnement d'analyse, les outils de modification de la résolution sont appliqués uniquement au raster obtenu. Les deux principales méthodes de changement de résolution sont :

* L’interpolation
* L’agrégation : permet de former une homogénéité. La méthode d'agrégation repose sur une agrégation de statistiques désignée dans un voisinage. Elle permet de dériver des valeurs dans le raster en sortie avec une résolution différente.

**aggregate(« raster à agréger », fact=, fun=)** Cette fonction associe les valeurs individuelles d'un groupe de cellules et permet d'obtenir une cellule unique avec une résolution grossière. Les types de statistiques que vous pouvez utiliser sont les suivants : somme, minimum, maximum, moyenne et médiane. « fact » est le facteur d’accumulation exprimé en nombre exprimé en nombre de cellules dans chaque direction (horizontale et verticale). « fun » est la fonction utilisée pour agréger les variables. *Ex : ind\_zra=aggregate(ind\_zr,fact=10,fun=mean) Ici par exemple on utilise la fonction moyenne.*

**resample(raster agrégé qui doit être rééchantilloné, raster dont les paramètres permettent le rééchantillonage)** Permet le re-échantillonnage càd le rééchantillon transfère des valeurs entre la non correspondance Raster-Objet en terme d’origine et de résolution.

### ACP à partir d’un raster

Faire un tableau avec en colonne les variables et en ligne les valeurs des pixels pour chaque variable et ajouter le package RStoolbox :

library(RStoolbox)

# Choisir les bandes/variables à prendre en compte pour l'ACP:

layers <- c(raster$B1\_blue,raster$B2\_green,raster$B3\_red,raster$B4\_nir,raster$B5\_swir1,raster$B6\_swir2, raster$ABS5med75)

newraster <- stack(layers)

#Effectuer l'ACP sur le raster:

pca <- **rasterPCA(**newraster**)**

# Voir les composantes et la proportion de variance sur chaque composante:

summary(pca$model)

pca$map

summary(pca$map)

plot(pca$map)

# Enregistrer l'ACP sous format raster:

fic<-paste0("D:/Mairi/RastersLSf\_Ind\_zones/ACPvers1.envi")

writeRaster(**pca$map**, bylayer=F, format="ENVI",filename = fic, overwrite=TRUE)

## Découper une variable numérique en classes

Le premier type de recodage consiste à découper une variable de type numérique en un certain nombre de classes. On utilise pour cela la fonction cut. Celle-ci prend, outre la variable à découper, un certain nombre d’arguments : **breaks** indique soit le nombre de classes souhaité, soit, si on lui fournit un vecteur, les limites des classes ; **labels** permet de modifier les noms de modalités attribués aux classes ; **include.lowest** et **right** influent sur la manière dont les valeurs situées à la frontière des classes seront inclues ou exclues ; **dig.lab** indique le nombre de chiffres après la virgule à conserver dans les noms de modalités.

Prenons tout de suite un exemple et tentons de découper notre variable age en cinq classes et de placer le résultat dans une nouvelle variable nommée age5cl :

d$age5cl <- **cut**(d$age, 5)

**table**(d$age5cl)

(17.9,33.8] (33.8,49.6] (49.6,65.4] (65.4,81.2] (81.2,97.1]

454 628 556 319 43

Par défaut R nous a bien créé cinq classes d’amplitudes égales. La première classe va de 16,9 à 32,2 ans (en fait de 17 à 32), etc.

Les frontières de classe seraient plus présentables si elles utilisaient des nombres ronds. On va donc spécifier manuellement le découpage souhaité, par tranches de 20 ans :

d$age20 <- **cut**(d$age, **c**(0, 20, 40, 60, 80, 100))

**table**(d$age20)

(0,20] (20,40] (40,60] (60,80] (80,100]

72 660 780 436 52

## Packages de validation croisée

La validation croisée désigne le processus qui permet tester la précision prédictive d'un modèle dans un échantillon test (parfois aussi appelé échantillon de validation croisée) par rapport à la précision prédictive de l'échantillon d'apprentissage à partir duquel le modèle a été développé. Dans l'idéal, avec un échantillon suffisamment grand, une certaine proportion d'observations (disons la moitié ou les deux-tiers) peut être affectée à l'échantillon d'apprentissage et les observations restantes sont affectées à l'échantillon test. Le modèle peut être construit en utilisant les observations de l'échantillon d'apprentissage, et la puissance prédictive peut être testée en utilisant les observations de l'échantillon test. Si le modèle s'exécute aussi bien sur l'échantillon test que sur l'échantillon d'apprentissage, nous pouvons dire que la validation croisée est bonne ou plus simplement, qu'il y a validation croisée.

### Package Random Forest

*Source :*

[*http://perso.ens-lyon.fr/lise.vaudor/classification-par-forets-aleatoires/*](http://perso.ens-lyon.fr/lise.vaudor/classification-par-forets-aleatoires/)

[*http://mehdikhaneboubi.free.fr/random\_forest\_r.html*](http://mehdikhaneboubi.free.fr/random_forest_r.html)

Ce package permet de repérer des liens entre une variable à expliquer et des variables explicatives en faisant une régression linéaire.

Rappel : un modèle de régression linéaire est un modèle de régression qui cherche à établir une relation linéaire entre une variable, dite expliquée, et une ou plusieurs variables, dites explicatives.

Il va classer les variables explicatives en fonction de leurs liens avec la variable à expliquer. Les forêts aléatoires sont composées (comme le terme "forêt" l'indique) d'un ensemble d'arbres décisionnels. Ces arbres se distinguent les uns des autres par le sous-échantillon de données sur lequel ils sont entraînés. Ces sous-échantillons sont tirés au hasard (d'où le terme "aléatoire") dans le jeu de données initial. Chaque arbre de la forêt est construit sur une fraction ("in bag") des données (c'est la fraction qui sert à l'entraînement de l'algorithme. Alors pour chacun des individus de la fraction restante ("out of bag") l'arbre peut prédire une classe.

Ce modèle doit expliquer la meilleure variance possible. Moins il peut expliquer la variance, plus il y a d’erreur.

**rf=randomForest(object, x=, y=, ntree=500)**

Avec :

Object = Data frame ou matrice contenant x et y ce qui est optionnel si le chemin vers l’objet « $ » est comrpis dans x et y.

x = le data frame prédicteur ou la matrice prédictrice

y = un vecteur de réponse càd la variable qu’on veut prédire. Si un facteur, la classification est assumée, autrement la régression est assumée.

ntree=500 = paramètre par défaut : la forêt est composée de 500 arbres.

Par défaut, la fonction randomForest tire au hasard n individus parmi les n (avec remplacement, ce qui devrait correspondre, en moyenne, à l'échantillonnage au hasard de 63.2% des individus). Ainsi sur les n = 500 arbres, chaque individu fera partie de la fraction "in bag" et "out of bag" en moyenne 0.632  500 = 316 et 0.368  500 = 184 fois (en moyenne, mais pas forcément exactement, du fait de l'aléa d'échantillonnage).

La fonction print nous permet d'afficher quelques caractéristiques de l'objet produit. On apprend ainsi que la forêt est composée de 500 arbres, qu'à chaque noeud l'algorithme fait un essai sur 2 variables, que le taux d'erreur est « Mean of squared residuals » et après on a le % de variance expliquée (R²). Si on regarde les infos avec la fonction str, mtry correspond au nombre de variables testées à chaque division qui est 2 par défaut.

### Package Cvtools

Permet de prendre aléatoirement 80% des données pour prédire les 20% restantes.

### Package PLS (partial least square regression)

Modèle très utile quand on a beaucoup de bandes. C’est un modèle qui peut prédire plusieurs grandeurs pour faire une prédiction globale et pas locale comparé aux 2 modèles Random Forest et Cv tools.

## Glossaire

|  |  |
| --- | --- |
| tmp | répertoire où on stocke les fichiers de façon temporaire |
| == | exactement égal à |
| «» | signifie vide |
| != | différent |
| length | taille d’un vecteur peut être obtenue par l’instruction length() |
| $ | synonyme de « prend/compte » *ex : melon$logpoid veut dire logpoid est compris dans melon* |
| ~ | « en fonction » *ex : plot(rf$y~rf$predicted, main="rf") veut dire rf$predicted en fonction de rf$y* |
| na.rm = TRUE | Enlever les NA d’une colonne |
| | |  |
| %in% | Pour voir les correspondances entre 2 objets -> Le résultat sera false s’il n’y a pas de correspondance et true s’il y en a. |

## ANOVA : Analyse de la variance

**L’ANOVA** est une méthode d’analyse qui permet d’envisager l’étude de la dépendance d’une variable quantitative continue à une ou deux variables qualitatives. Les variables qualitatives impliquées sont appelées facteurs. On peut s’intéresser aux effets principaux de ces facteurs ainsi qu’à l’effet de leur interaction sur la variable quantitative étudiée. On parle alors d’ANOVA a 1 ou 2 facteurs et l’analyse est :

* Bivariée lorsqu’un seul facteur est considéré
* Multivariée lorsque 2 facteurs sont considérés

La variable quantitative est appelée variable dépendante et est notée VD. Les facteurs sont les variables indépendantes notées VI.

Pour savoir si la VD a une dépendance significative au facteur considéré, on va étudier l’homogénéité de la moyenne de la VD sur l’ensemble des modalités de la VI pour lesquelles elle a été calculée. On espère rejeter l’hyp nulle H0 qui est l’égalité des moyennes par l’analyse de la variance. Le test d’hyp réalisé est un test d’hypothèse de fisher comparant la variance inter et intra échantillon. Le rapport de ces variances devra s’éloigner assez fortement de 1 pour rejeter l’hypothèse => lorsque F>>1, , H0 est rejetée. F = Vinter/Vintra.

* On essaie d’expliquer la cause de la diversité des informations recueillies par l’analyse de leur variance.

**La variance** est une mesure de la dispersion des valeurs d'un échantillon ou d'une distribution de probabilité. Elle exprime la moyenne des carrés des écarts à la moyenne, aussi égale à la différence entre la moyenne des carrés et le carré de la moyenne. La variance est toujours positive, et ne s’annule que si la distribution de valeurs est constante. Sa racine carrée définit l’écart type σ, d’où la notation σ² = V. L’écart type correspond du coup à la dispersion des valeurs d’un échantillon.

## XI. Package Biomass

Contient des fonctions permettant d'estimer la biomasse / le carbone en surface et son incertitude dans les forêts tropicales. Ces fonctions permettent :

* de récupérer et de corriger la taxonomie
* d’estimer la densité du bois et son incertitude
* de construire des modèles hauteur-diamètre
* de gérer les coordonnées des arbres et des parcelles
* d’estimer la biomasse / carbone en surface au niveau du peuplement avec incertitude associée

Fonctions du package :

**correctTaxo ()** Cette fonction corrige les fautes de frappe pour un nom taxonomique donné à l'aide du service de résolution de noms taxonomiques (TNRS) via l'interface Taxosaurus.

## XII. Mettre à jour R (télécharger la nouvelle version)

Package installr

Commandes :

*# installing/loading the package:*

if(!**require**(installr)) {

**install.packages**("installr");

**require**(installr)

} *#load / install+load installr*

## XIII. Changer la library R de place pour régler les pbs d’accents dans le nom d’utilisateur

> .libPaths()

[1] "C:/Users/Souza Oliveira Maïri/Documents/R/win-library/3.5"

[2] "C:/Program Files/R/R-3.5.1/library"

> myPaths <- .libPaths() # get the paths

> myPaths <- c(myPaths[2], myPaths[1]) # switch them

> .libPaths(myPaths) # reassign them

> .libPaths()