**Installation R et RStudio**

1. **Installer R**

**Windows**

1. Pour installer R il faut aller sur :

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

une fois vous êtes sur cette page il faut cliquer sur :

 après vous lancez le fichier téléchargé (.exe) comme tous les logiciels Windows

**Linux (Debian/Ubuntu)**

1. Il suffit de lancer la commande suivante en sudo :

sudo apt-get update

sudo apt-get install r-base r-dev

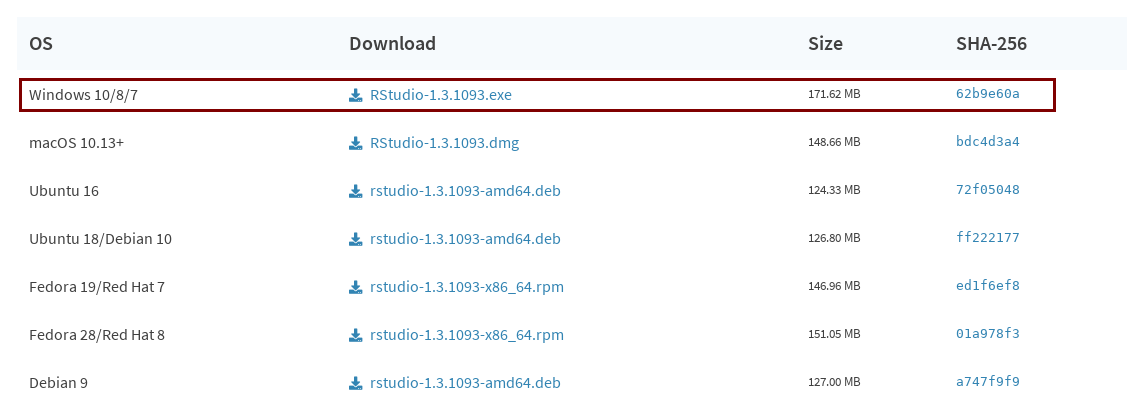
1. **Installer RStudio**

**Windows**

1. Pour installer RStudio il faut aller sur :

[https://rstudio.com/products/rstudio/download/#download](https://rstudio.com/products/rstudio/download/" \l "download)

une fois vous êtes sur cette page il faut cliquer sur :



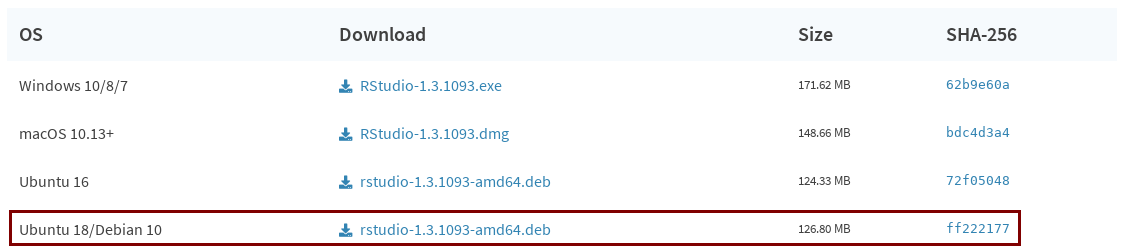
après vous lancez le fichier téléchargé (.exe) comme tous les logiciels Windows

**Linux (Debian 10 /Ubuntu 18)**

1. On va sur le même lien

[https://rstudio.com/products/rstudio/download/#download](https://rstudio.com/products/rstudio/download/" \l "download)

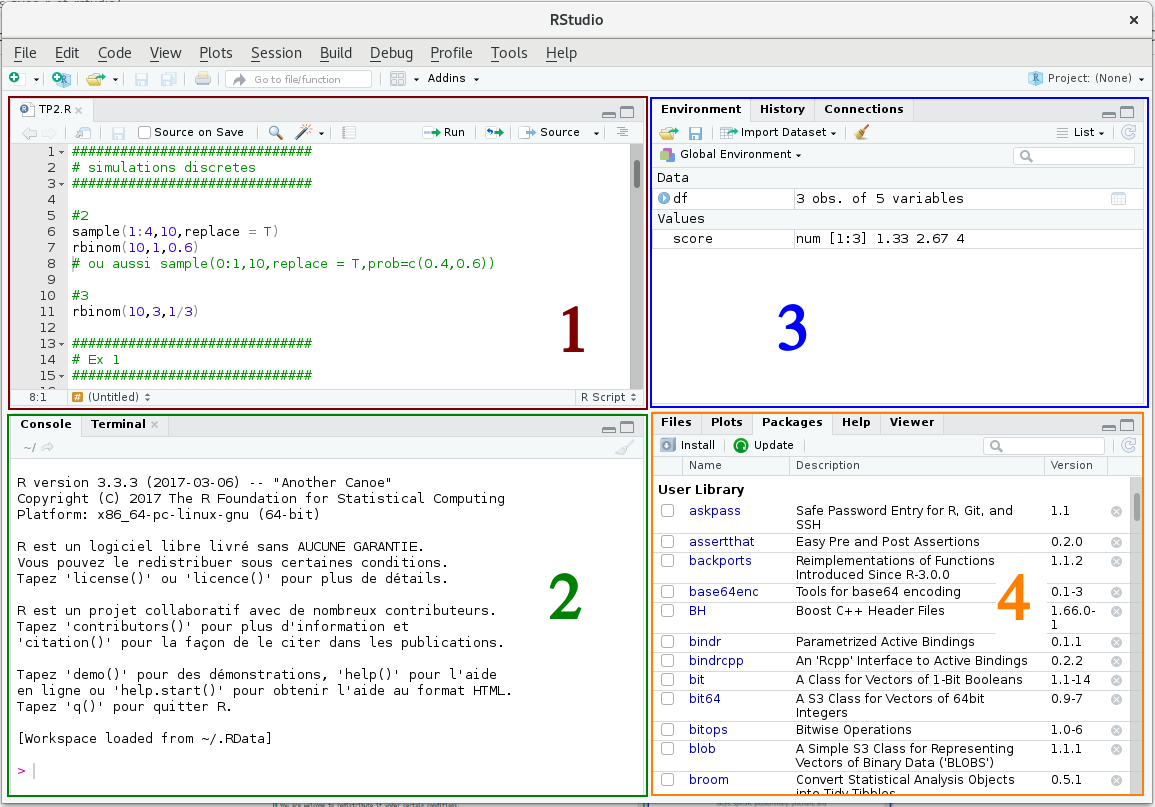
une fois vous êtes sur cette page il faut cliquer sur (selon votre version linux):



après il faut aller où se trouve le fichier téléchargé, une fois dans le dossier (normalement le dossier est Téléchargement ou Downloads) on lance la commande :

sudo dpkg -i [rstudio-1.3.1093-amd64.deb](https://download1.rstudio.org/desktop/bionic/amd64/rstudio-1.3.1093-amd64.deb)

**Interface RStudio**

****

**1.** Fenêtre d’édition des fichiers sources. Ici vous pouvez notamment éditer votre fichier de script, sélectionner des parties et les exécuter avec *Ctrl- Enter* (*Cmd-*⏎ sur Mac) où avec le button run.

**2.** LeConsole. Ici, vous pouvez écrire des lignes de commandes directement. Appuyez sur Enter pour exécuter une ligne que vous venez d’écrire. C’est aussi là que s’affichent les éventuels messages d’erreur.

**3.** Fenêtre *environnement*. Ici s’affiche la liste des objets que vous avez créés. Ces objets peuvent être de simples variables, des tableaux, des fonctions, des objets graphiques etc, c’est bien de supprimer les objet crées à chaque nouveau exercice (on peut faire ça en appuyant sur le button avec la photo de **balaie**.

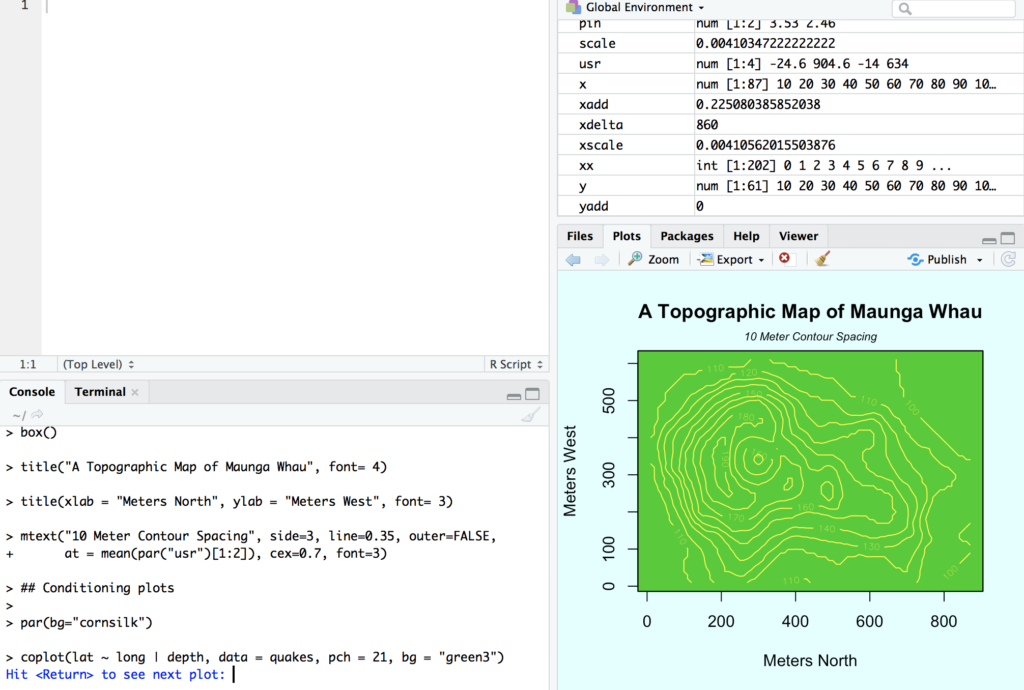
**4.** Fenêtre multi-onglets: ***Files*** sert à naviguer dans le système de fichiers de votre ordinateur, ***Plots*** affiche les graphiques, ***Packages*** affiche les paquets disponibles et chargés (lorsque cochés). ***Help*** donne de l’aide sur les fonctions.

## Exécutez des lignes de codes dans la console (B)

Dans la zone B, écrivez

demo(graphics)

Puis appuyez sur la touche ⏎ (Enter). Vous devriez voir une démonstration des capacités graphiques de R:



Regardez à la fois le résultat graphique dans la zone D et les messages du programme dans la console B. En bleu apparaissent parfois des indications, par exemple Hit <Return> to see next plot. Suivez ces instructions. Continuez à appuyer sur la touche Enter jusqu’à la fin de la démonstration. Vous saurez que la démonstration est finie lorsque vous verrez le “curseur” clignoter dans une nouvelle ligne vide > de l’invite de commandes.



À l’endroit du curseur, dans la console (B) écrivez :

**a** <- 2

et pressez ***Enter***. Vous venez d’attribuer la valeur 2 à la variable *a*. Notez l’utilisation du signe *<-* . Ce dernier s’appelle *assignement operator*. Il sert à attribuer une valeur à une variable. Son usage est quasiment identique à l’usage de l’opérateur = dans d’autres langages, comme *JavaScript* ou *Python*. Vous n’avez pas besoin de [comprendre les subtilités de la différence entre = et <-](https://renkun.me/2014/01/28/difference-between-assignment-operators-in-r/) pour l’instant. Dans la console (B) écrivez :

a

La console devrait afficher la valeur assignée à a:

[1] 2

À présent, appliquez une fonction simple à a:

**a** + 2

La console affichera le résultat (4). À présent, nous allons créer deux vecteurs. Vous pouvez soit copier/coller les deux lignes suivantes ligne par ligne dans la console, ou les deux à la fois puis presser Enter.

b <- c(3,4,5,6)

d <- c(1,2,1,2)

À présent, multipliez les vecteurs:

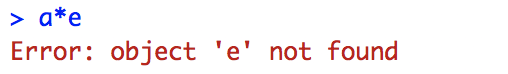
b\*d

Que voyez-vous? Et qu’arrive-t-il lorsque vous multipliez a\*b ou a\*d ?

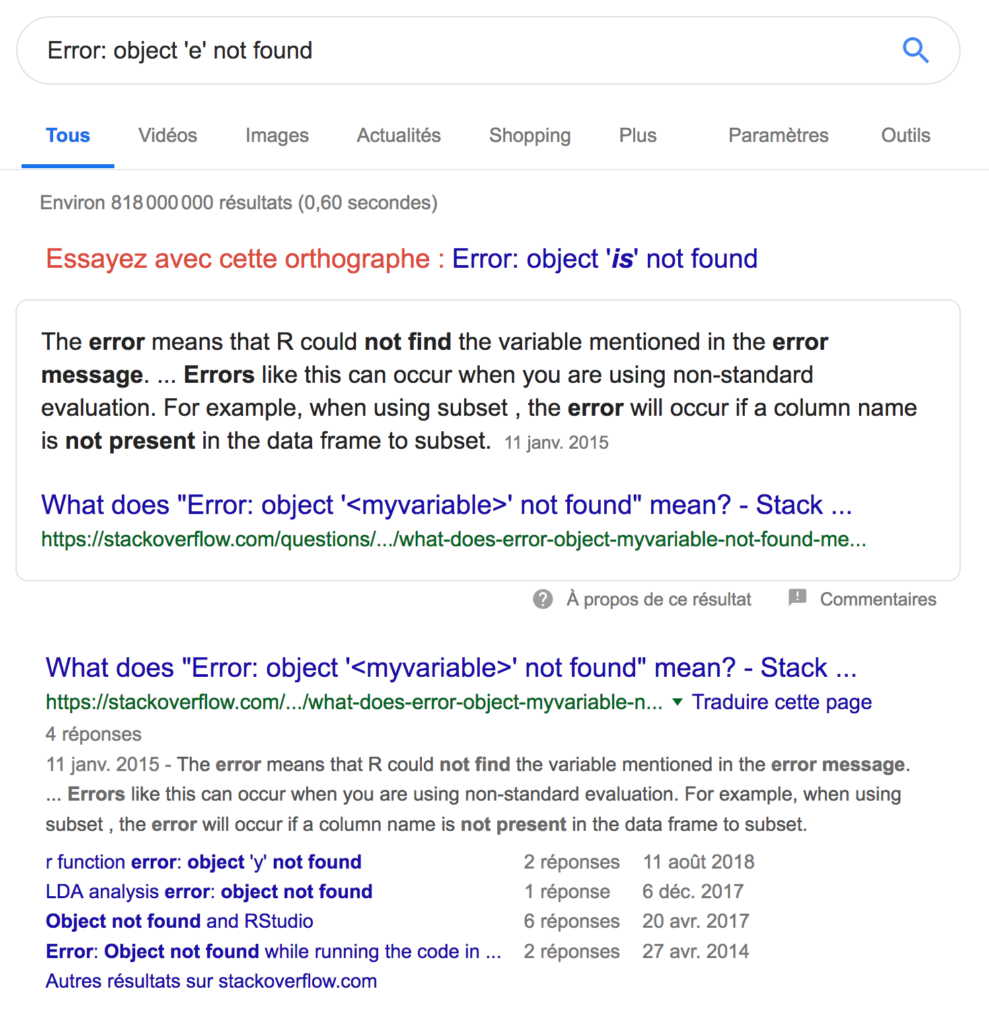
À présent, essayez d’exécuter la commande suivante:

a\*e

Vous verrez ceci:



Vous venez de créer votre première erreur de code. En programmation, il est extrêmement important d’accorder de l’attention à de tels messages. Ils vous permettent de corriger votre code. Ici, la source de l’erreur est évidente: nous n’avons, en aucun moment, défini de variable e; il est donc impossible de multiplier a par e. D’autres messages d’erreur que vous obtiendrez par la suite seront plus énigmatiques. Il est improbable, cependant, que vous soyez la première personne à voir un message d’erreur. La meilleure chose à faire, toujours, est de copier la totalité du message et de chercher s’il existe une explication de l’erreur – et une solution – à l’aide d’un moteur de recherche Internet. Essayez:



## **Créez votre premier fichier de script**

S’il est pratique d’exécuter des commandes simples dans la console, le but de la programmation est d’écrire des scripts réutilisables contenant chacun une série de commandes que R exécute dans l’ordre d’apparition. Créez un nouveau fichier de script R. (*File>New File>R Script*). Si elle n’est pas déjà ouverte, la fenêtre A s’ouvre. Elle est vide pour l’heure.

Sauvegardez votre nouveau fichier de script en pressant *Ctrl+S* (ou avec *File > Save*) et en lui donnant un nom de votre choix. *Souvenez-vous où, c’est-à-dire dans quel dossier, vous l’avez mis.*

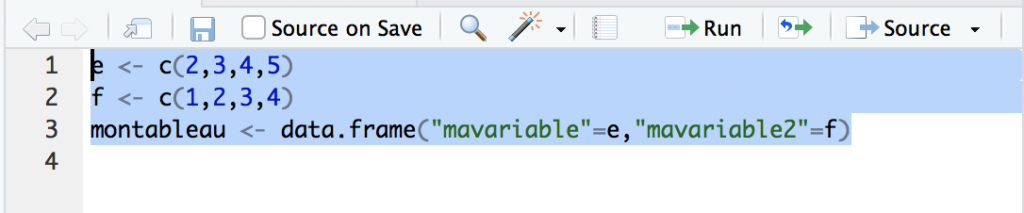
Copiez le code suivant et collez-le dans la fenêtre A:

**e** <- c(2,3,4,5)

f <- c(1,2,3,4)

montableau <- data.frame("mavariable"=e,"mavariable2"=f)

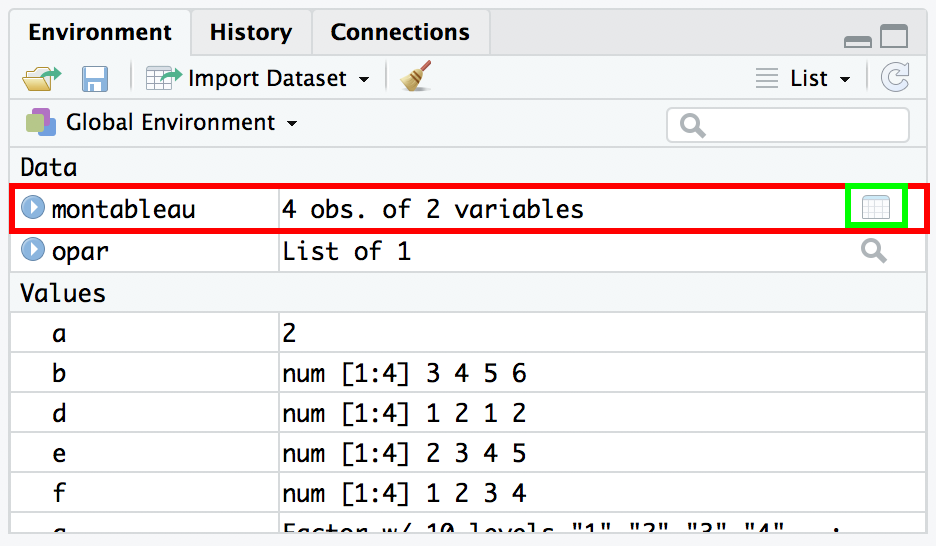
Vous venez de commencer à écrire script. Sauvegardez votre travail en pressant *Ctrl+S* (ou avec *File > Save*).



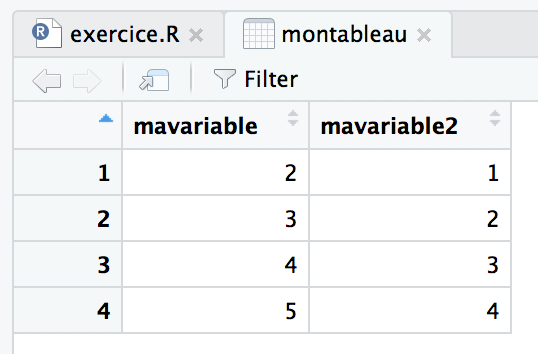
Sélectionnez l’ensemble du code et pressez control-⏎ (Ctrl-Enter). Vous venez d’exécuter le code à partir de votre fichier de script. La suite des commandes a créé un tableau de données nommé montableau, et contenant deux colonnes:

* mavariable qui contient les valeurs du vecteur e.
* mavariable2 qui contient les valeurs du vecteur f.

Ce tableau est désormais listé dans l’environnement d’exécution (C):

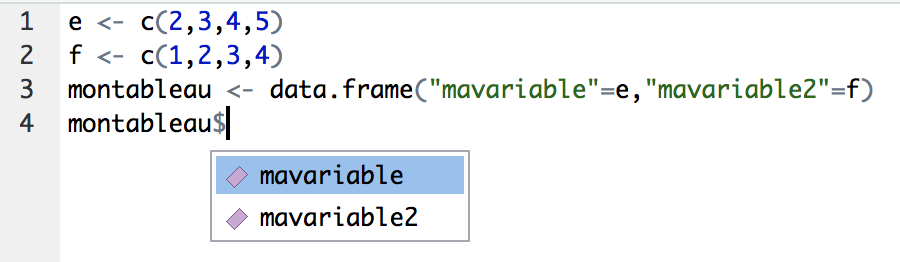


Pour voir le contenu du tableau appuyez sur le symbole de tableau encadré en vert sur l’image ci-dessus. Un nouvel onglet s’ouvre dans la zone A, montrant son contenu:

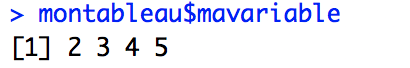


Les tableaux sont au cœur de l’usage de *R*. Ce langage est conçu spécifiquement pour les manipuler et pour accéder facilement à leur contenu. *R* ressemble, en cela, au [module *pandas* pour le langage *Python*](https://pandas.pydata.org/pandas-docs/stable/getting_started/comparison/comparison_with_r.html). Les tableaux dans R peuvent être stockés sous forme de matrices de divers types. Le plus utilisé de ceux-ci est désigné par le terme *data.frame*. Il y a plusieurs possibilités pour accéder à une colonne d’un *data.frame*. La plus simple est l’usage du signe $.

Écrivez montableau$ dans la suite de votre script. Quand vous utilisez *RStudio* pour programmer, il vous offre l’avantage de “l’autocomplétion” de code: vous n’avez pas besoin d’écrire tout le code. *RStudio* devine la suite et vous propose des options. Quand elles s’affichent, comme ci-dessous, il suffit alors d’appuyer Enter ou Tab ↹ pour que la suite du code s’écrive toute seule.



Exécutez la ligne sur laquelle vous vous trouvez en pressant ***Ctrl+Enter***. (Utilisez cette méthode pour exécuter chaque bout de code proposé dans le suite de ce tutoriels, après l’avoir ajouté à votre fichier de script.) Vous devriez voir ceci dans la console:



### **Alternatives et commentaires**

Il existe d’autres manières d’accéder aux valeurs d’une colonne d’un *data.frame*:

montableau[,"mavariable"] # cette ligne équivaut à la précédente

De nouveau, ajoutez cette ligne à la suite de votre code et exécutez-la. Notez l’usage du signe #. Les lignes ou parties de lignes précédées d’un # sont des commentaires du code, que *R* n’exécute pas. Vous pouvez ajouter autant de commentaires que vous voulez à votre code. Les commentaires sont essentiels dans la programmation:

* Pour vous, lorsque vous allez réutiliser votre code dans quelques semaines ou mois. Vous comprendrez alors beaucoup plus facilement ce que vous avez cherché à faire aujourd’hui.
* Pour travailler en équipe, lorsque vous transmettez votre code à un tiers.

Il existe des styles de codage, et chaque problème informatique possède une quantité de solutions différentes. Les commentaires aident à les comprendre.

Poursuivez en exécutant les lignes suivantes une par une et en observant les commentaires:

montableau[,1] # encore une autre façon d'accéder à la colonne mavariable. Ici, on dit à R: "montre-moi la première colonne de montableau"

# Accédez à plusieurs variables:

montableau[,c("mavariable","mavariable2")] # Vous pouvez accéder à plusieurs variables, en indiquant leurs noms dans un vecteur

montableau[,c(1,2)] # Identique à la ligne précédente

montableau[,c(1,2,3)] # Ceci retourne une erreur "undefined columns selected". Trouvez ce qu'elle signifie

g <- c("premier","deuxième","troisième","quatrième") # créons un vecteur de chaînes de charactères

montableau$num\_observation <- g # ajoutons le vecteur g comme nouvelle colonne de montableau

View(montableau) # Jetons un coup d'oeil à montableau. La fonction View(nom\_du\_tableau) équivaut à l'acte d'appuyer sur le bouton correspondant dans RStudio

montableau[,c(1,2,3)] # Cette fois, cette ligne ne retourne pas d'erreur. Pourquoi?

montableau$ef <- montableau$mavariable + montableau$mavariable2 # nous venons de créer une colonne par une opération sur deux colonnes

montableau[1,] # Accédez à la première ligne de montableau

montableau[1,"mavariable"] # Accédez à la valeur de mavariable de la première ligne de montableau. Notez la syntaxe tableau[ligne,colonne] pour accéder à une valeur particulière

montableau[1,]$mavariable # Équivaut à la ligne précédente

montableau[1,1] # Équivaut à la ligne précédente

montableau[1,"mavariable"] <- 10 # changez la valeur de mavariable dans votre premier enregistrement

View(montableau) # Vérifions le changement

montableau[c(2,3,4),] # affichez les lignes 2, 3 et 4

montableau[2:4,] # Équivaut à la ligne précédente. Parce que la syntaxe 2:4 est juste une manière abrégée de la création d'un vecteur c(2,3,4)

# Essayez:

c(2,3,4)

2:4 # Équivaut à la ligne précédente. Oui, le signe ":" ne signifie pas une division mais une séquence (de 2 à 4 dans le cas présent)

seq(2,4) # Équivaut à la ligne précédente. seq(début,fin) est une fonction de R

2/4 # Pour diviser, on utilise le signe "/"

Les data.frames de R offrent en outre la possibilité importante de sélection de données correspondant à des critères plus complexes:

montableau[montableau$mavariable>4,] # Sélectionne les entrées (individus) pour qui la valeur de mavarible est supérieure 4

montableau[montableau$mavariable2 \* 2 < montableau$mavariable,] # Sélectionne les entrées pour qui le double de la valeur de mavarible2 est supérieur à mavariable

# De telles sélections s'appuyent sur un "vecteur logique". Essayez

montableau$mavariable>4

# Donne: TRUE FALSE FALSE TRUE

# Cela signifie que le premier eregistrement de montableau remplit le critère (TRUE), le second pas, ni le troisième, mais le quatrième enregistrement le satisfait de nouveau. La sélection retoure toutes les lignes pour qui le résultat est TRUE

# Essayez ceci:

montableau[c(**TRUE**,**FALSE**,**FALSE**,**TRUE**),] # le résultat est identique. Pourquoi?

# Contrôlez un autre vecteur logique

montableau$mavariable2 \* 2 < montableau$mavariable

# Donne TRUE FALSE FALSE FALSE

montableau[c(**TRUE**, **FALSE**, **FALSE**, **FALSE**),] # Ceci correspond à montableau[montableau$mavariable2 \* 2 < montableau$mavariable,] . Pourquoi?

Essayez de faire des sélections avec d’autres critères.

## **Fonctions**

Dans le code précédent, nous avons utilisé des fonctions prédéfinies de R. Vous pouvez créer vos propres fonctions

# Définissions une nouvelle fonction.

mafonction <- **function**(q,w) {

variable1 <- q / 10

variable2 <- w ^ 3 / q

**return**(variable1 + variable2)

}

# Notez l'usage de l'opérateur "^" qui signifie la puissance. Ici w^3 veut dire w à la puissance 3, càd, w au cube. Notez aussi l'usage de la commande return(quelque\_chose). La commande return() à l'intérieur d'une fonction définit ce que la fonction va renvoyer, càd quel résultat elle donnera

mafonction(3) # Ceci vous retourne une erreur "argument "w" is missing, with no default". Que signifie-t-elle?

mafonction(3,5) # Ici, nous fournissons deux paramètres à mafonction. Les paramètres sont pris dans l'ordre d'apparition. 3 sera donné à q, 5 sera donné à w. Notez que vous pouvez nommer vos paramètres comme vous le souhaitez:

# redéfinissons la fonction en changeant les noms de paramètres

mafonction <- **function**(blabla,blablou) {

variable1 <- blabla / 10

variable2 <- blablou ^ 3 / blabla

**return**(variable1 + variable2)

}

mafonction(3,5) # Le résultat demeure identique

# Vous pouvez appliquer une fonction à une série de valeurs:

mafonction(c(3,5),c(5,12))

mafonction(c(3,5),12) # Ceci retourne aussi un résultat. Mais attention: que signifie-t-il

mafonction(c(1,1),1) # En essayant ceci, vous comprendrez mieux.

# Mais l'usage le plus intéressant d'une est bien sûr de l'appliquer aux colonnes d'un tableau:

mafonction(montableau$mavariable,montableau$mavariable2)

# Enregistrons le résultat dans une nouvelle colonne:

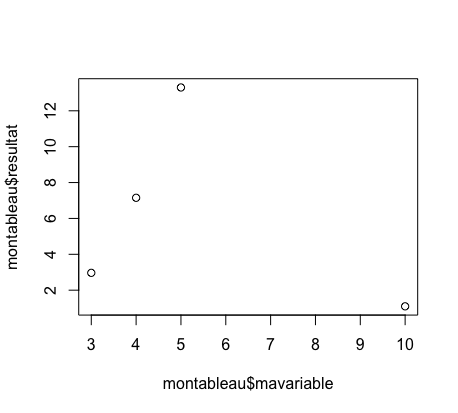
montableau$resultat <- mafonction(montableau$mavariable,montableau$mavariable2)

View(montableau) # Vérifions

## **Les modules complémentaires (*packages*)**

Les fonctions prédéfinies de R vous permettent de générer, entre autres, des graphiques. Essayez:

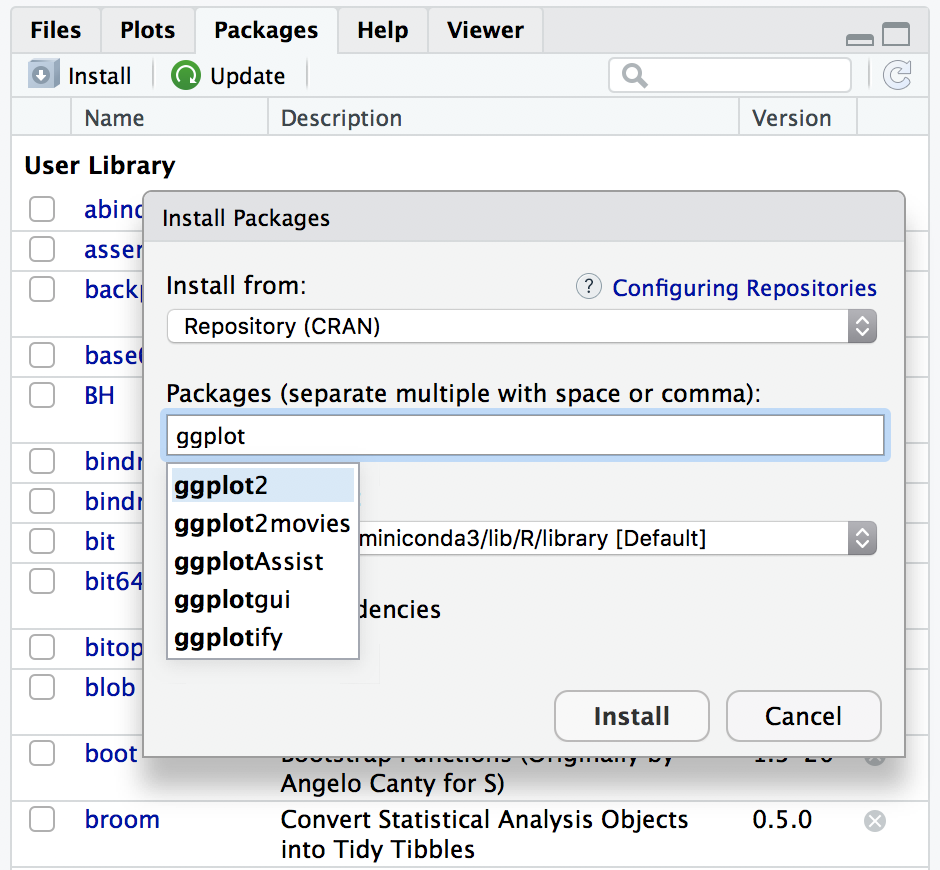
plot(montableau$mavariable,montableau$resultat)



Comme vous le voyez, ce graphique est compréhensible mais il pourrait être plus beau. Les fonctions graphiques de base de R permettent en théorie de faire tout ce que vous voulez, mais il existe des modules complémentaires qui rendent la tâche plus aisée. Le vaste écosystème de modules complémentaires est en fait l’un des plus grands avantages de R. Installons le plus célèbre parmi eux: [ggplot2](http://docs.ggplot2.org/current/).

### **Installation d’un ou de plusieurs modules**

Dans l’onglet “Packages” de la zone D, cliquez sur “Install” et tapez *ggplot2*:



Appuyez “Install”. Pendant le processus d’installation, notez que des messages s’affichent dans la console:



De manière générale, appuyer sur un bouton dans l’interface *RStudio* envoie une commande à la console. *RStudio* peut être considéré comme un générateur convivial de commandes *R*. Vous auriez, en effet, tout aussi bien pu installer le module *ggplot2* en ajoutant la commande suivante dans votre script et en l’exécutant:

**install**.packages("ggplot2")

Cette manière de faire peut s’avérer très utile pour installer plusieurs modules sans cliquer à répétition. Essayez en ajoutant les commandes suivantes à votre code et en les exécutant. Notez que l’exécution peut prendre un certain temps. En attendant, renseignez vous sur les modules installés en cherchant leur nom sur Internet:

**install**.packages("reshape2")

**install**.packages("splines")

**install**.packages("car")

**install**.packages("sandwich")

**install**.packages("RcmdrMisc")

**install**.packages("scatterplot3d")

**install**.packages("plot3D")

**install**.packages("rgl")

**install**.packages("gtable")

**install**.packages("gridExtra")

**install**.packages("MASS")

**install**.packages("fmsb")

**install**.packages("plyr")

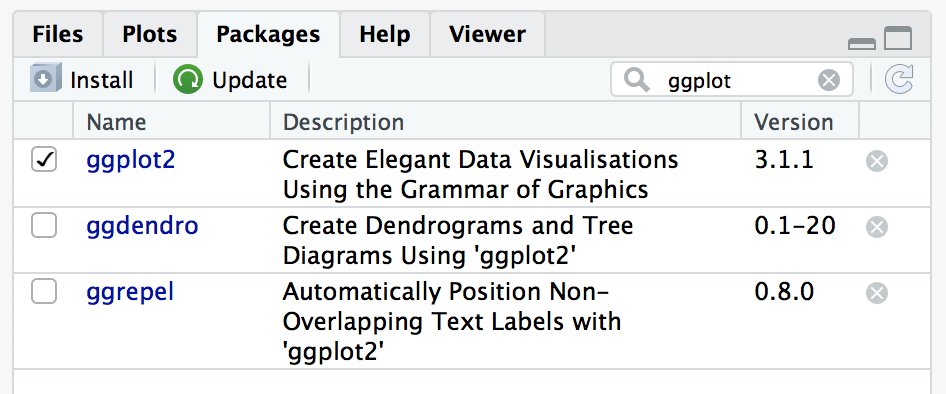
**install**.packages("ggdendro")

**install**.packages("readxl")

En fonction de la configuration de votre machine, certaines installations généreront peut-être des avertissements (*warnings*). Un avertissement est généralement moins grave qu’une erreur en informatique; il signifie “j’ai pu exécuter la commande indiquée mais pas tout à fait comme prévu, il se pourrait qu’il y ait des erreurs par la suite”. Prenez note des avertissements sans vous en inquiéter davantage pour l’heure.

### **Activez les modules (*packages*) installés**

Afin d’utiliser un module, il faut l’activer. Pour ce faire, cochez la case correspondante  dans le navigateur des modules (*packages*) de *Rstudio*. Activons *ggplot2*.



À nouveau, un coup d’œil sur la console vous permet de voir que l’activation d’un module se fait à l’aide de la commande library("nom\_du\_paquet"). Vous pouvez ajouter les lignes suivantes à votre code et les exécuter:

**library**("ggplot2")

**library**("reshape2")

**library**("splines")

**library**("car")

**library**("sandwich")

**library**("RcmdrMisc")

**library**("scatterplot3d")

**library**("plot3D")

**library**("rgl")

**library**("gtable")

**library**("gridExtra")

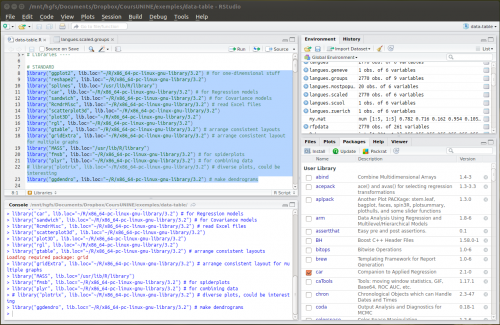
**library**("MASS")

**library**("fmsb")

**library**("plyr")

**library**("ggdendro")

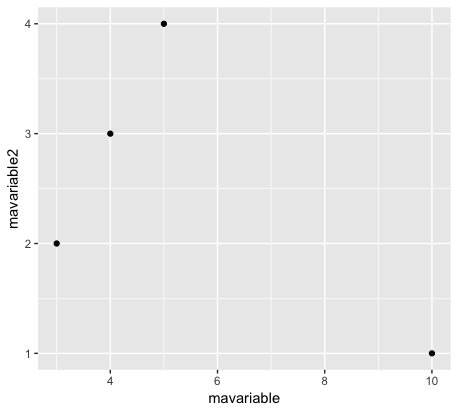
**library**("readxl")



Revenons à présent à notre graphique. Avec *ggplot2* installé, vous avez accès à la commande ggplot():

ggplot(data = montableau) +

geom\_point(aes(x=mavariable,y=mavariable2))



L’affichage est plus esthétique. *Ggplot2* vous donne accès à un grand nombre de paramètres:

ggplot(data = montableau) +

geom\_point(aes(x=mavariable,y=mavariable2,size=resultat),color="blue") +

xlab("Le nom long de ma variable") +

ylab("Le nom long de mon autre variable") +

labs(size = "Ce que donne\nma fonction", title = "Le titre de mon graphique")

