R pour les nuls

Centre d'analyse des politiques publique
Université Laval

Table des matières

1 C'est quoi R? 2

2 R vs. RStudio 2

3 RStudio – La base 3
3.1 Télécharger les logiciels . . . 3
3.2 Interface . . . 5

4 HelloWorld 8
4.1 Workflow . . . 8
4.1.1 Arborescence . . . 9
4.1.2 Commentaires . . . 9
4.1.3 Raccourcis clavier . . . 9

5 Intro à la programmation en R 10
5.1 Ça va bien aller . . . 10
5.2 Opérateurs . . . 11
5.2.1 Opérateurs de calcul . . . 11
5.2.2 Opérateurs d'assignement . . . 12
5.2.3 Opérateurs logiques . . . 12
5.2.4 Instructions de contrôle . . . 12
5.3 Structure de données . . . 14
5.3.1 Types de données et constantes . . . 14
5.3.2 Vecteurs . . . 15
5.3.3 Data frames . . . 17

6 Importation de données 18
6.1 Importation de données . . . 18

7 Analyse - Description 19
7.1 Univariée . . . 19
7.2 Bivariée . . . 19

8 Analyse - Régression linéaire simple et multiple 21

9 Visualisation 22

1 C'est quoi R?

Bonjour et bienvenue! Ce tutoriel sera votre guide de démarrage ainsi qu'un document de référence pour débuter la programmation en R[1]. Dans l'objectif d'introduire les étudiants de sciences politiques aux méthodes quantitatives et à l'analyse causale en sciences sociales, nous avons cru bon de vous initier au langage de programmation R. N'ayez crainte, c'est plus simple qu'il n'y paraît et vous en tirerez beaucoup d'avantages.

Pour la petite histoire, la première version de **R** a été publiée en 1995 par Ross Ihaka et Robert Gentleman, mais le langage s'inspire des travaux de John Chambers aux laboratoires Bell dans les années 1970. Aujourd'hui, **R** est un outil d'analyse statistique populaire, tant dans le secteur privé que dans le monde universitaire. **R** est ce que l'on appelle un *logiciel libre*, ce qui signifie que son code source est ouvert. Ceci permet à des utilisateurs bénévoles de développer des *packages* (micrologiciel ou librairie de fonctions) qui sont ensuite rendus disponibles à la communauté (pour la plupart gratuitement). Ceci fait de **R** un outil puissant, flexible, public, et donc particulièrement adapté à la méthode scientifique.

Le reste du document vous permettra de vous familiariser avec le langage de programmation **R** et avec son environnement de travail **RStudio**. Nous encourageons donc sa lecture attentive.

2 R vs. RStudio

Une distinction importante à effectuer est la différence entre le langage **R** et l'*IDE*[2] **RStudio**. L'*IDE* a pour fonction principale de recevoir le code et de le compiler. En d'autres mots, **R** c'est la langue que l'on écrit et le papier c'est l'*IDE*. Plusieurs *IDE* existent et il est facile de se perdre dans leurs différents paramètres et fonctionnalités. Dans ce cours, nous imposons l'utilisation de **RStudio**. De cette façon, tous les étudiants ont accès au même environnement de travail, peu importe l'ordinateur utilisé. **Si vous avez un Chromebook, il se peut que l'installation soit plus difficile, car il est important de connaître quelques détails de votre version Chromebook afin de savoir quelle version de RStudio. télécharger. Venez nous voir pour des précisions.**

Ainsi, dans le cadre du cours, les étudiants utiliseront le langage de programmation **R** à partir de l'environnement **RStudio**. Cette distinction s'affinera au cours de la session, n'ayez crainte. Pour les plus curieux d'entre vous, de nombreuses ressources, francophones et anglophones, existent sur internet :

- ChatGPT
  - ChatGPT est un outil d'intelligence artificielle générative. Des outils de ce type peuvent être très puissants pour vous aider à comprendre et rédiger des lignes de code. Néanmoins, cet outil n'est pas utile pour écrire un code du début à la fin. Si vous l'utilisez de telle manière, vous risquez d'avoir plusieurs pépins dans votre code que vous ne comprendrez pas – donc qui seront encore plus difficiles à corriger que simplement l'écrire par vous-même! Pour bien utiliser cet outil, il faut également savoir comment écrire des *prompts* pertinents. Voici une suggestion :
    - *En tant qu'analyste de données programmant dans le langage R, [décrire le problème ou la ligne de code que vous souhaitez mieux comprendre et /ou insérer un exemple de code ici]. Fournis un exemple.*
- Datacamp
  - Ceci est un site de formation en ligne. Datacamp se concentre sur des exercices pratiques en plusieurs langages de programmation incluant **R**, **Python** et **SQL**. Avec une approche très pratique, c'est un bon moyen d'accélérer l'intégration de connaissances techniques.
- Stackoverflow
  - Google était le meilleur ami des programmeurs avant les outils d'intelligence artificielle génératives. Si vous rencontrez un problème, Google vous fournira sans doute la solution sous la forme d'un *post* sur Stackoverflow. Il s'agit d'un site répertoriant les questions d'utilisateurs concernant la plupart des langages de programmation, incluant **R**. À la manière du logiciel libre, ce sont les autres utilisateurs du site qui se chargent de répondre avec grande précision. C'est vraiment un outil important.
- r-bloggers

[1] Ce tutoriel fondé sur le travail de Vincent Arel-Bundock, Yannick Dufresne, et Florence Vallée-Dubois

[2] *Integrated development environment* ou environnement de développement intégré.

  - Pour être informé sur les nouveaux développements de **R** et de **RStudio**. Encore une fois, il s'agit d'un point de rencontre de la communauté.

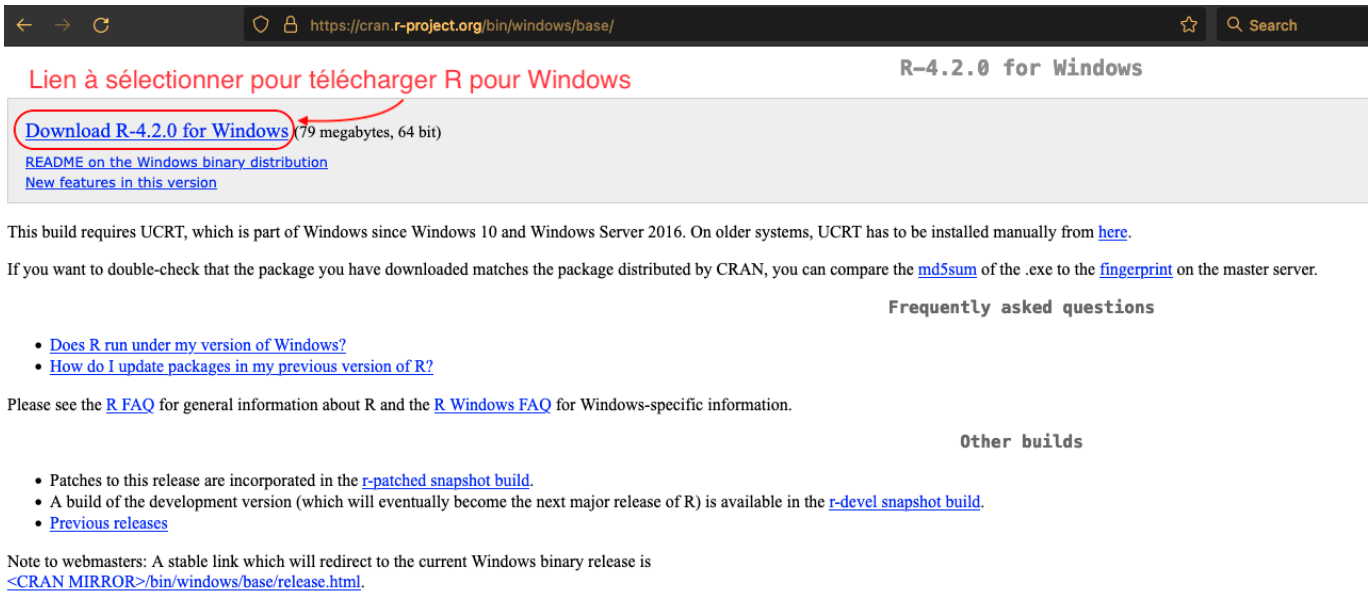
- Datanovia
  - Similaire à Datacamp, Datanovia est une excellente source d'information bilingue sous forme de tutoriel et de blogue. Ce site web est particulièrement utile lorsqu'il s'agit de la visualisation de données.
- Coursera
  - Site de formation en ligne. Les cours ont le format de cours universitaires, mais avec la version gratuite : pas besoin de suivre l'entièreté des plans de cours (ni de remettre les travaux).

3 RStudio – La base

3.1 Télécharger les logiciels

Dans le cadre du cours, nous utiliserons le logiciel **R** qui est gratuit et de plus en plus utilisé en science politique. Vous devez télécharger la version adaptée pour votre système d'exploitation. Mac et Windows sont les deux systèmes les plus communs, mais il est aussi possible avec Chromebook. Vous pouvez télécharger la version pour Mac en cliquant **ici** et la version pour Windows en cliquant **ici**. Si vous avez un Chrombook, venez nous voir afin de télécharger la bonne version pour votre ordinateur.

(a) Télécharger R pour Mac



(b) Télécharger R pour Windows

Figure 1: Comment télécharger R sur votre ordinateur personnel

Une fois que vous avez téléchargé **R**, vous devez aussi télécharger **RStudio**. **RStudio** n'est pas un logiciel qui fonctionne seul, il nécessite que la base de **R** soit installée sur votre ordinateur. **RStudio** est une interface qui va « par-dessus » R et facilite un peu son utilisation. Il faut donc préalablement avoir installé **R**, puis on installe **RStudio**. Une fois cela fait, on peut simplement utiliser **RStudio**. Vous pouvez télécharger RStudio Desktop (la version Open Source) pour Mac ou Windows **ici**.

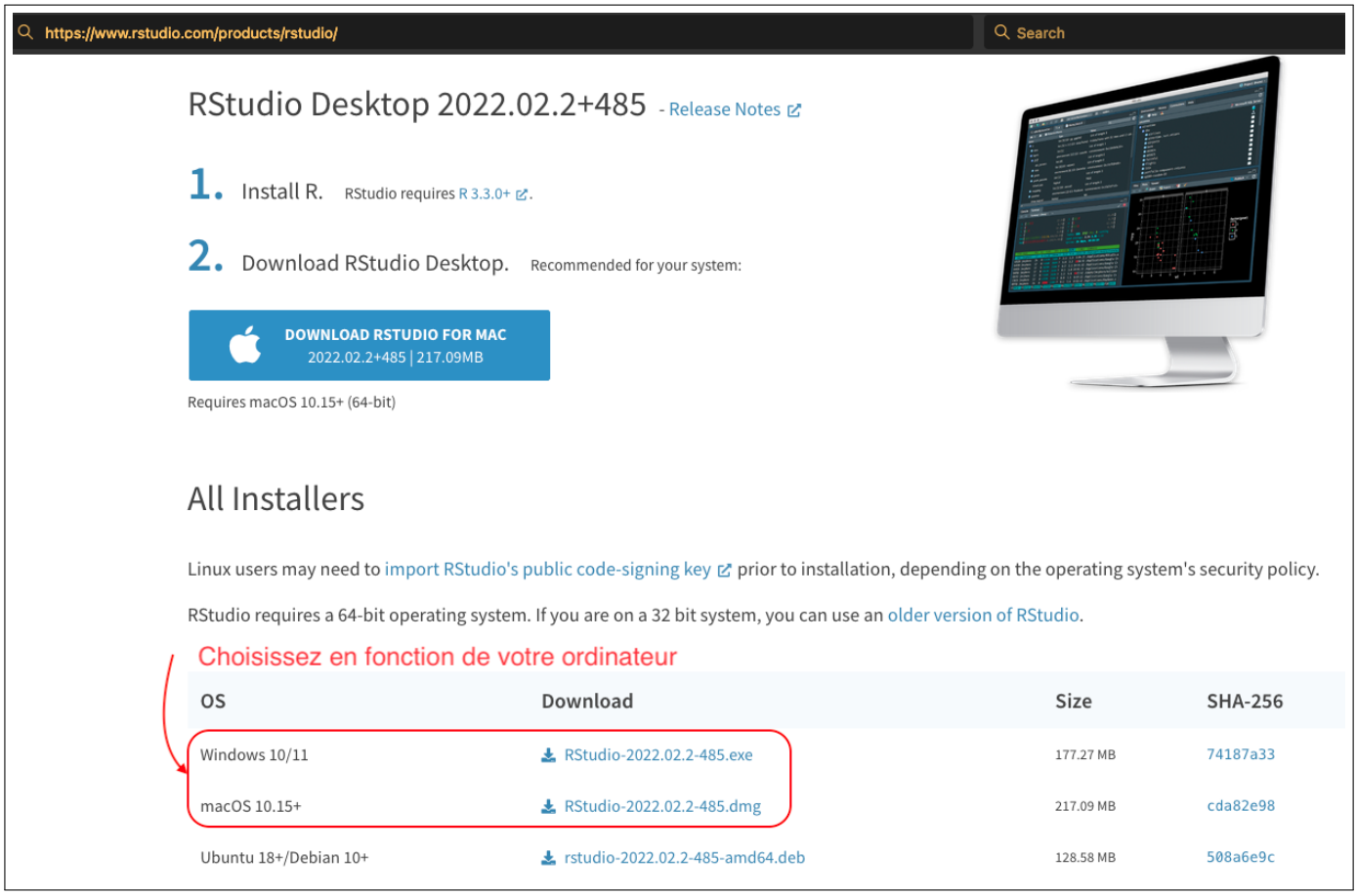


Figure 2: Comment télécharger RStudio sur votre ordinateur personnel

3.2 Interface

Lorsque vous ouvrez **RStudio**, vous verrez l'interface. Ceci est l'endroit où vous écrirez et exécuterez votre code. La figure 3 présente les points d'intérêt les plus importants. L'*environnement* contiendra les bases de données que vous importerez ainsi que tous les objets créés lors de la session. Nous aborderons les bases de données dans une section ultérieure. En dessous de l'*environnement*, vous avez accès à vos dossiers, les librairies utilisées, un aperçu des graphiques produits ainsi qu'à de l'aide dans la fenêtre en bas à droite. La plus grande fenêtre par défaut est celle contenant la console. La **console** est un environnement d'exécution directe. En d'autres mots, vous pouvez y écrire des commandes qui seront exécutées immédiatement. C'est utile pour faire des tests et exécuter de petites manipulations. Or, la **console** n'enregistre pas la suite de commandes que vous lui demandez de faire, ce n'est pas son rôle. Il est donc déconseillé d'utiliser la console pour écrire le code. Au lieu, vous devez ouvrir un nouveau R Script ce qui vous permettrait de sauvegarder vos lignes de code dans l'**éditeur** tel que spécifié par la figure 4 et les sauvegarder. L'**éditeur**, c'est l'endroit où l'on écrit un code (une suite de commandes permettant d'atteindre un but quelconque, comme calculer une moyenne, faire une régression simple ou produire un graphique) afin qu'il puisse être enregistré et réutilisé. Dans le cadre de ce cours, vous allez principalement travailler dans l'**éditeur**.[3]

[3] Parfois, lorsque vous avez un R Script ouvert, la console peut le cacher. Dans ce cas, vous pouvez simplement cliquer là ou indiqué dans la figure 3 pour le rapetisser. Vous verrez ensuite votre éditeur.

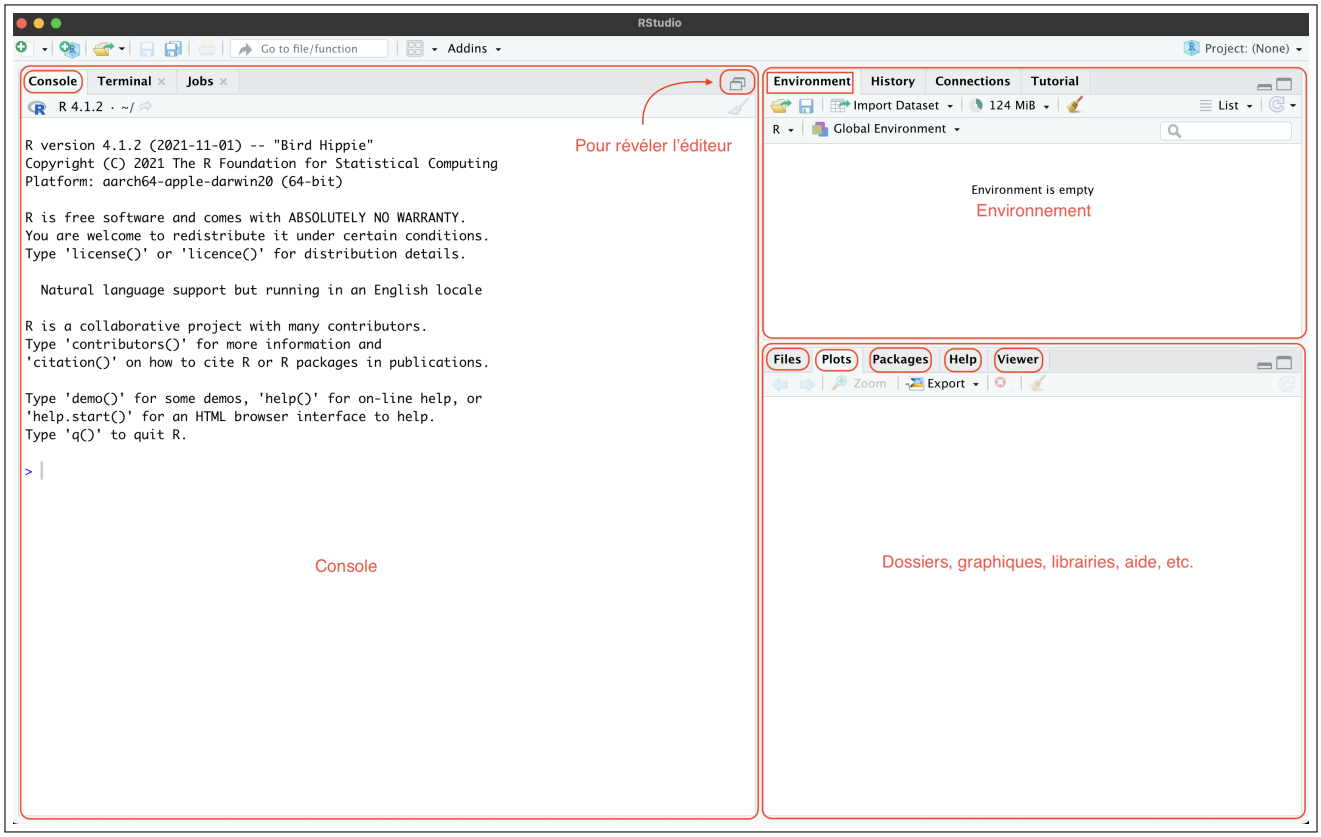


Figure 3: État par défaut

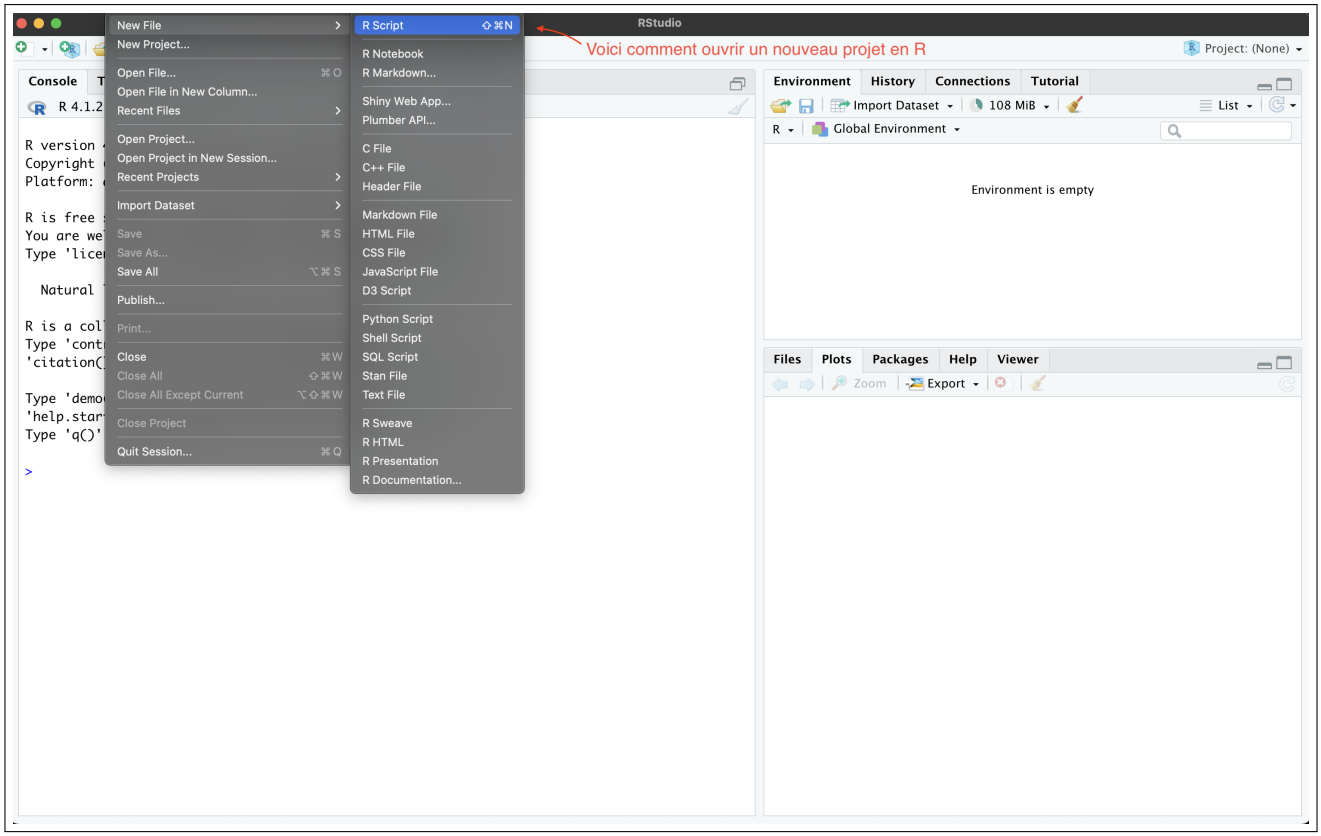


Figure 4: Comment ouvrir un nouveau R Script

Avant d'entrer dans la syntaxe d'utilisation de **R**, je recommande fortement la personnalisation de votre environnement. Non seulement cela vous permettra d'avoir une expérience esthétique plus agréable, cela vous aidera à long terme à reconnaître les structures du langage. Les figures 5 et 6 montrent comment accéder aux options de l'environnement global. Vous y trouverez ma mise en place préférée. Je tends à préférer coder sur fond plus foncé, mais je vous encourage à explorer les options et à trouver ce qui vous correspond le mieux. C'est pour vos yeux, pas ceux de l'équipe d'enseignement.

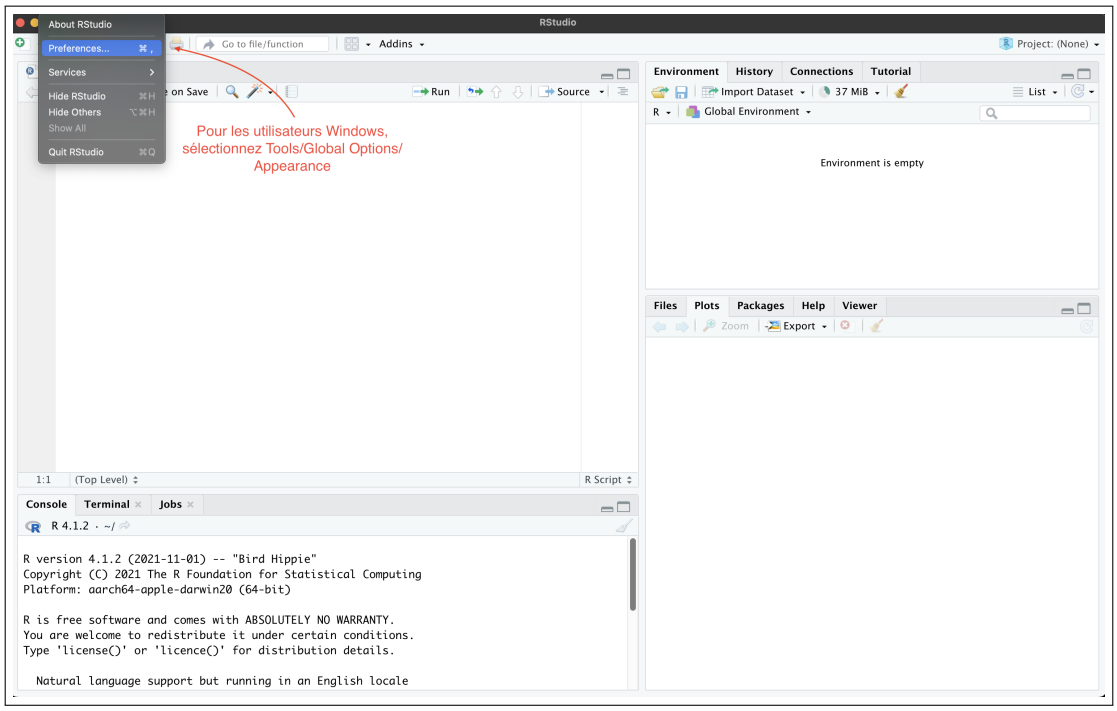


Figure 5: Début de la personnalisation

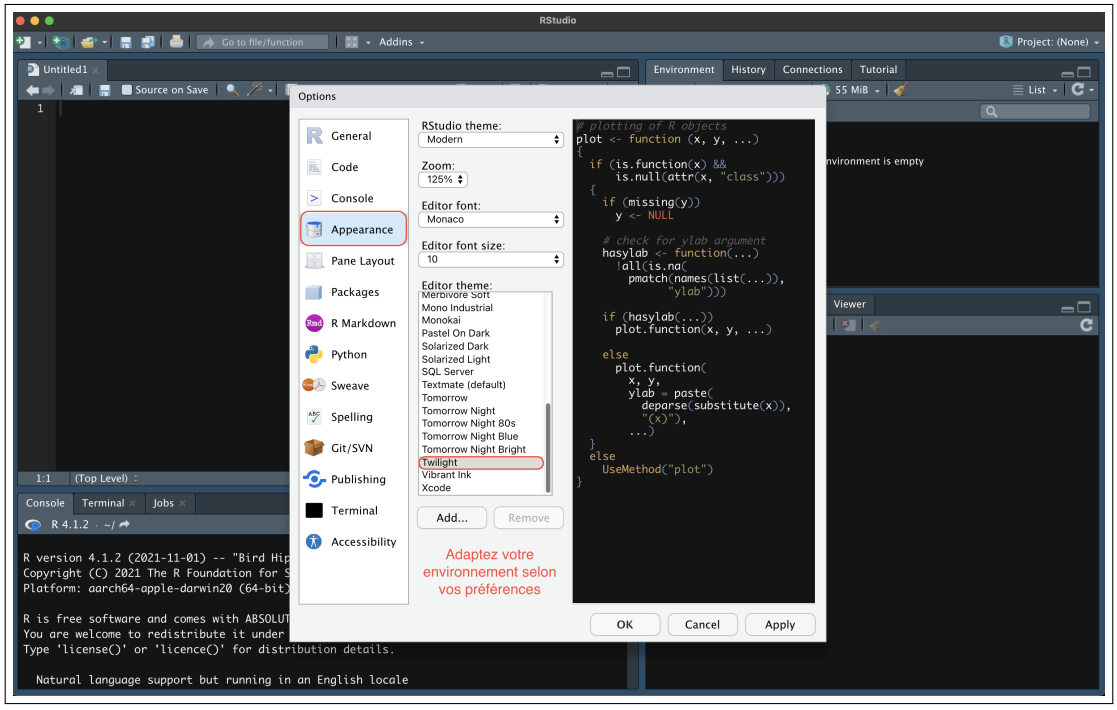


Figure 6: Fin de la personnalisation

4 HelloWorld

Vous en connaissez maintenant assez à propos de **R** pour rédiger votre première ligne de code! Allez dans l'éditeur, et tapez la commande suivante :

```
> print("HelloWorld!")
```

Comme vous pouvez voir, **R** est un gentil programme qui fait une partie du travail pour vous. Dès que vous ouvrez une parenthèse dans l'éditeur, le logiciel ajoute déjà la seconde parenthèse pour s'assurer qu'il n'y ait pas de message d'erreur. **R** fait la même chose pour les guillemets.

À présent, il faut exécuter votre ligne de code. Pour ce faire, sélectionnez la ligne, puis cliquez sur le bouton ***RUN*** en haut à droite. La figure 7 vous montre un exemple. Une fois exécuté, ce simple code imprimera n'importe quelle suite de caractères que vous aurez choisi. Amusez-vous!

Remarquez que le résultat s'affiche dans la console en bas. C'est sa deuxième utilité. La console vous indique les résultats des commandes que vous exécutez à partir de l'éditeur. Félicitations, vous êtes désormais des programmeurs, le fun peut commencer!

ATTENTION! Pour la suite du tutoriel, je vous invite tout de suite à repérer ces symboles sur votre clavier d'ordinateur : <, >, =, ", ', $, |, &, %, [,], {, }.

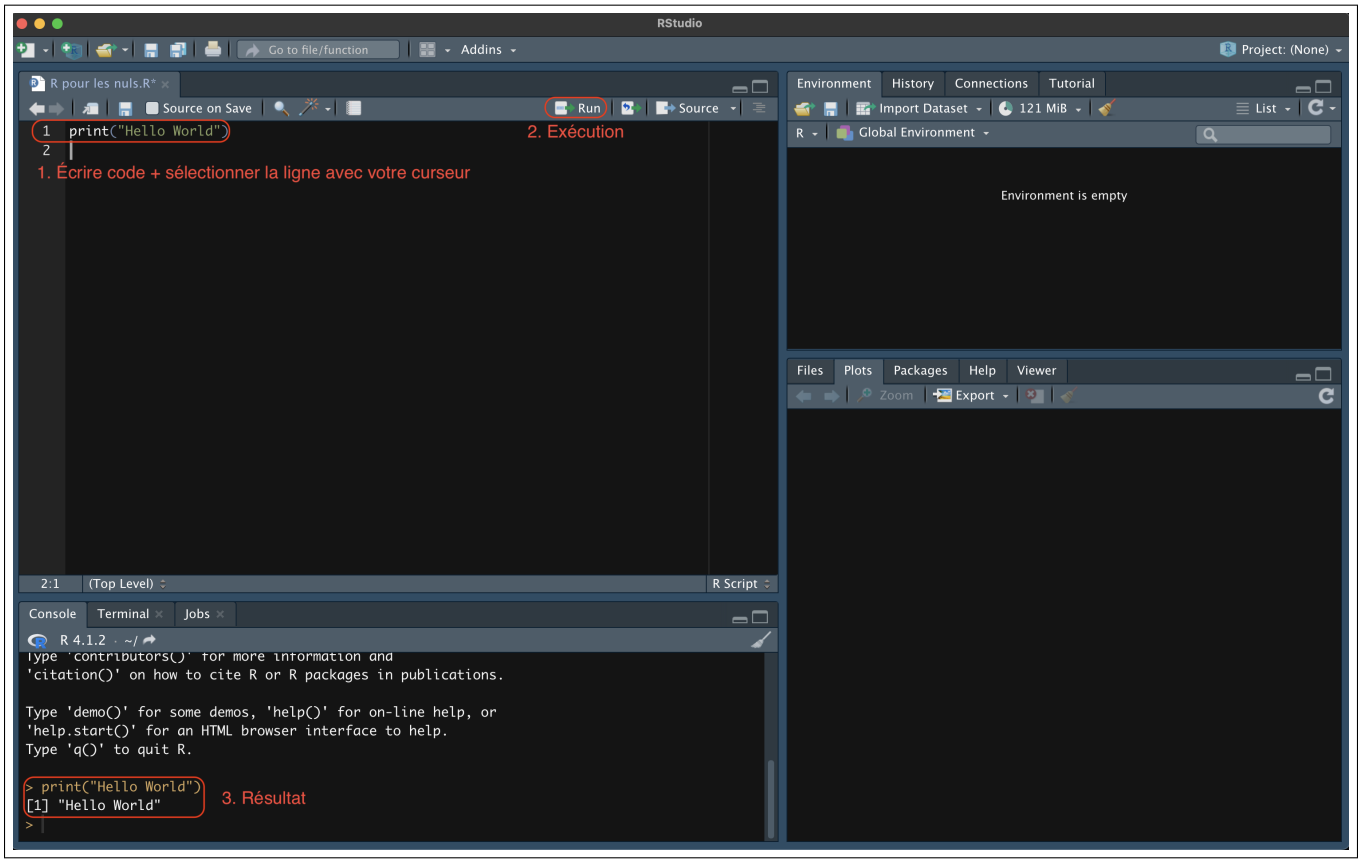


Figure 7: Votre première ligne de code!

4.1 Workflow

Un facteur important à prendre en considération lorsque l'on programme ou que l'on gère des bases de données, c'est l'organisation de notre « *flux de travail* ». Plusieurs éléments sont à considérer lorsque l'on veut optimiser son travail. Nous aborderons les principaux dans cette section. Ces habitudes peuvent vous aider à éviter des situations fâcheuses.

4.1.1 Arborescence

Si vous lisez le titre de cette section sans savoir ce à quoi nous faisons référence, ne vous inquiétiez pas. La plupart des gens utilisent l'arborescence de leur ordinateur sans même s'en rendre compte. C'est tout simplement le chemin par lequel vous (ou votre ordinateur) devez passer pour accéder à un fichier. Comme quand vous voulez retrouver un vieux travail, votre ordinateur doit savoir où se trouvent les fichiers que vous mobilisez dans votre code. Pour ce faire, il faut établir le « répertoire de travail » (*working directory*) de la session **R**. On utilise alors la fonction `setwd()` et on y inscrit l'arborescence (le chemin) :

```
# Pour les Mac
> setwd("/Users/nomDelaSession/nomDuDossier/nomDuSousDossier")
# Pour les PC
> setwd("C:/Users/nomDelaSession/nomDuSousDossier")
```

Remarquez la façon dont l'arborescence est écrite. Chaque dossier est suivi d'une barre oblique (/) et d'un sous-dossier. C'est donc important de bien organiser les dossiers sur son ordinateur de sorte à éviter de devoir changer de répertoire de travail trop souvent. L'idéal c'est de référer à un dossier général contenant les dossiers spécifiques. Par exemple, si vous travaillez sur un projet de recherche à propos des élections fédérales en 2021, vous pourriez avoir un dossier `ElectionFed21` qui contient un dossier `code`, un dossier `visualisations` et un dossier `data`. De cette façon, vous pouvez indiquer un répertoire de travail général au début de votre code pour ensuite utiliser des arborescences relatives. Ne vous inquiétez pas, tout ceci deviendra très simple au cours de la session.

4.1.2 Commentaires

Commenter un code c'est l'habitude la plus importante à prendre, surtout lorsqu'on est en apprentissage. Que voulons-nous dire par là? Imaginez que vous êtes en train d'explorer une forêt. Vous pourrez sans doute retrouver votre chemin de mémoire, mais l'idéal c'est de laisser des repères tout au long du chemin. Surtout si vous n'y retournerez pas avant un moment. Coder, c'est la même chose. Vous allez développer une suite d'instructions logiques qui permettra de produire un résultat spécifique. Il faut donc vous assurer de pouvoir bien comprendre les différentes étapes de votre code, et ce, même après plusieurs mois ou années. La mémoire étant une faculté qui oublie, ce n'est peut-être pas l'outil le plus fiable pour cette tâche. Donc, pour commenter, il vous suffit de précéder ce que vous écrivez par un `#`. Voici un exemple dans lequel nous avons divisé les réponses à une variable (qui variait entre 0 et 1) en trois catégories distinctes :

```
# Separation en trois niveaux de la variable ses_health
> Data$ses_health[Data$ses_health==0.75 | Data$ses_health==1] <- 1
> Data$ses_health[Data$ses_health==0.5] <- 0.5
> Data$ses_health[Data$ses_health==0.25 | Data$ses_health==0] <- 0
```

4.1.3 Raccourcis clavier

Écrire des lignes de code peut être frustrant, surtout lorsque vous n'êtes pas habitué à repérer certains symboles comme [ ] et { }. Comme dans toute chose, avec de l'expérience, vous allez acquérir de la rapidité. Par ailleurs, certains raccourcis clavier de **RStudio** pourront vous éviter de perdre de précieuses secondes. Puisque ces raccourcis vont varier en fonction du système d'opération de votre ordinateur, les deux prochaines sections s'adressent aux utilisateurs Mac et PC respectivement.

4.1.3.1 Raccourcis clavier – Utilisateurs Mac

1. `cmd + enter`, permet de rouler la partie du code sélectionné.
2. `option + sélection`, permet de sélectionner une portion de plusieurs lignes en même temps. Voir la figure 8.
3. `shift + flèches`, permet de sélectionner à l'aide des flèches (gauche, droite, bas, haut) du clavier.
4. `cmd + flèches`, permet d'accéder au début (avec la flèche du haut) et à la fin (avec la flèche du bas) d'un document.
5. `option + flèche`, permet d'inverser une ligne avec celle du haut (avec la flèche du haut) ou avec celle du bas (avec la flèche du bas).

4.1.3.2 Raccourcis clavier – Utilisateurs PC

1. `ctrl + enter`, permet de rouler la partie du code sélectionné.
2. `alt + sélection`, permet de sélectionner une portion de plusieurs lignes en même temps. Voir la figure 8.
3. `shift + flèches`, permet de sélectionner à l'aide des flèches (gauche, droite, bas, haut) du clavier.
4. `ctrl + flèches`, permet d'accéder au début (avec la flèche du haut) et à la fin (avec la flèche du bas) d'un élément du code.
5. `alt + flèche`, permet d'inverser une ligne avec celle du haut (avec la flèche du haut) ou avec celle du bas (avec la flèche du bas).

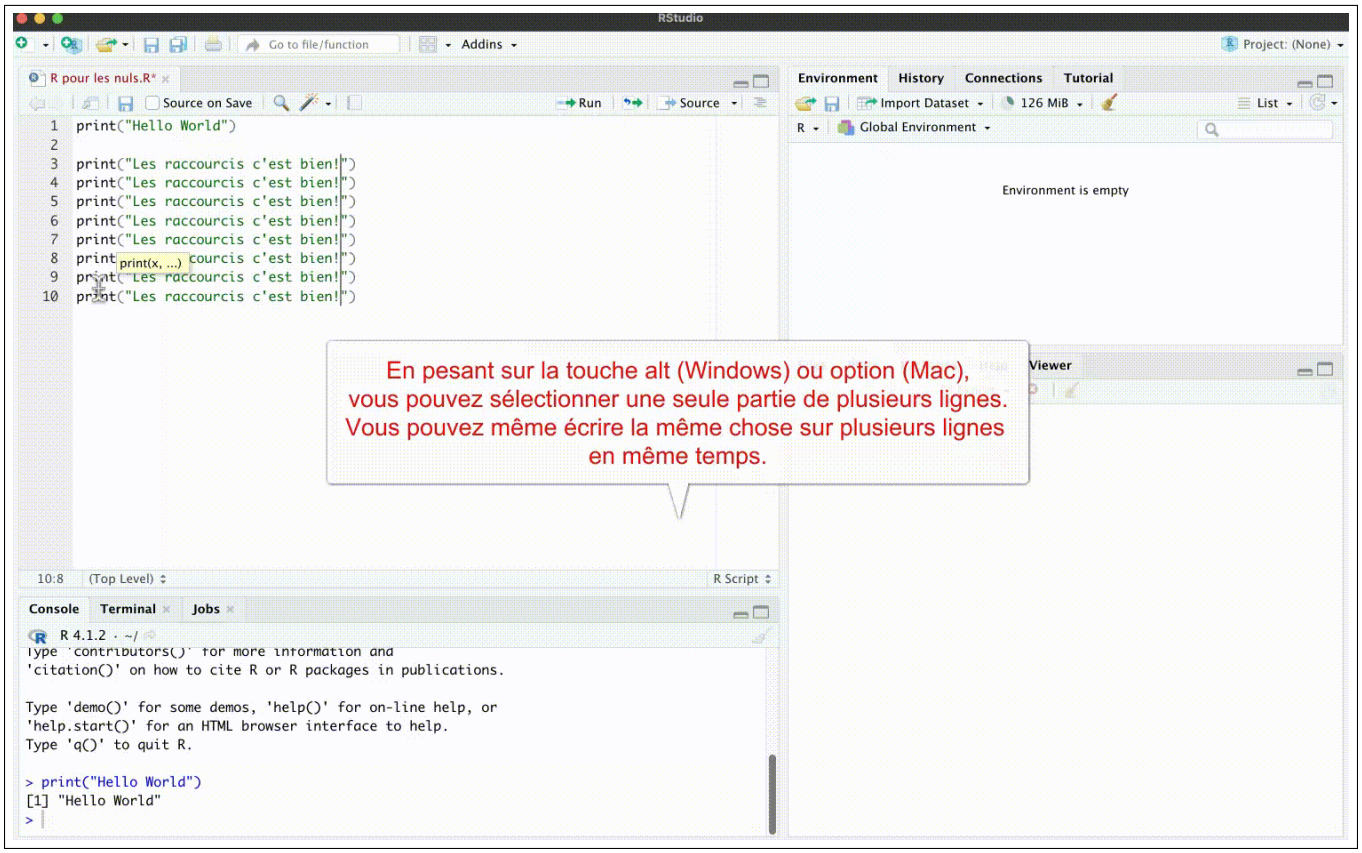


Figure 8: Sélection multiple de lignes

5 Intro à la programmation en R

Vous connaissez le logiciel **R** et vous avez de bonnes habitudes de codage. Nous pouvons désormais entrer dans le vif du sujet, la programmation en **R**.

5.1 Ça va bien aller

Plusieurs étudiants sont anxieux face à la tâche d'apprendre un langage de programmation. C'est tout à fait normal. La programmation vient avec une aura cryptique, un jargon et des programmeurs qui ne sont pas toujours à même d'expliquer ce qu'ils font. Or, pour ce cours, nous considérerons **R** comme une calculatrice +, c'est-à-dire comme un outil permettant d'effectuer des opérations mathématiques plus ou moins complexes. Vous ne deviendrez pas des développeurs à la fin de ce cours et ce n'est pas l'objectif. Le but est plutôt de vous former pour que vous soyez en mesure d'effectuer des analyses de base, une compétence qui vous distinguera sur le marché du travail.

Ceci étant dit, vous allez avoir besoin d'aide à un moment, c'est certain. L'équipe d'enseignement est, bien entendu, disponible pour répondre à vos questions. Cependant, nous préférons que vous développiez votre autonomie. Pour ce faire, nous vous encourageons à consulter les ressources en ligne mentionnée à la section 2. D'autres ressources sont également offertes par **R** lui-même. En précédant toute fonction d'un `?` dans la console [4]:

```
# Dans la console, pour obtenir de l'aide pour la fonction sample()
> ?sample()
```

5.2 Opérateurs

Comme vous avez certainement déjà saisi, au lieu d'avoir une interface qui vous permet de cliquer sur les boutons et onglets, la plupart des actions en **R** se font à l'aide d'*opérateurs*. Ceux-ci sont simplement des symboles utilisés pour diriger le programme à faire des actions quelconques. C'est donc en manipulant ces opérateurs qu'il soit possible de calculer une moyenne, de faire un graphique, ou même de faire de l'apprentissage machine. Ainsi, les différents types d'opérateurs sont la base de la programmation en **R**. Nous pouvons sous-diviser les opérateurs et différents types. Les prochaines sections vous présenteront aux opérateurs de bases auxquels vous allez avoir affaire dans le cadre du cours.

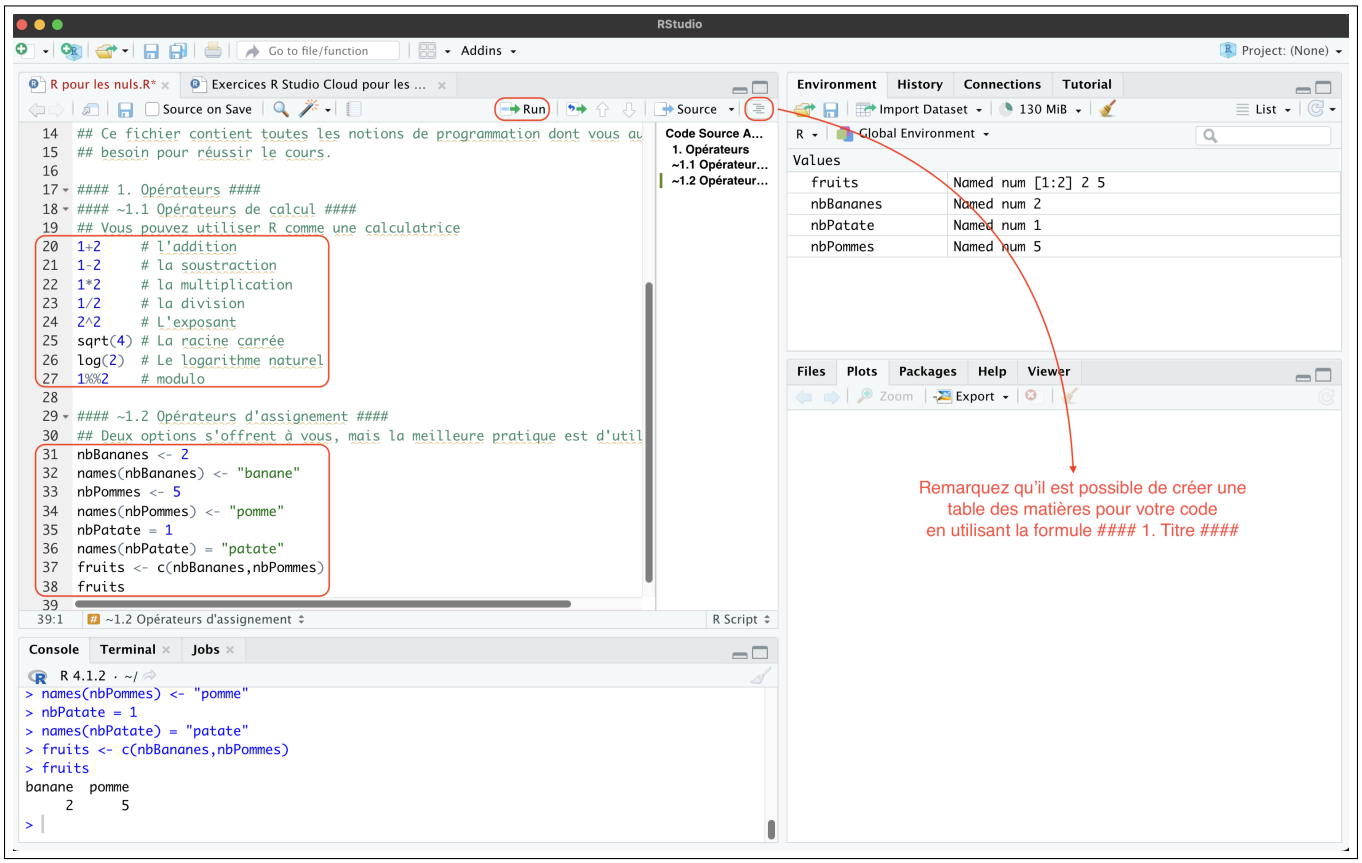


Figure 9: Opérateurs de calcul et d'assignement

5.2.1 Opérateurs de calcul

Les opérateurs de calcul sont les opérateurs mathématiques que vous connaissez (`+`,`-`,`*`,`/`).[5] Ces opérateurs sont aussi visibles dans l'éditeur à la figure 9. Tout comme une calculatrice, vous pouvez utiliser ces opérateurs pour faire des calculs. Essayez :

[4] Lorsqu'une fonction n'est pas dans votre environnement de travail et que vous utilisez `?ggplot()`, **R** retournera une erreur du genre « `Error in .helpForCall(topicExpr, parent.frame()) : no methods for 'ggplot' and no documentation for it as a function` ». Lorsque c'est le cas, vous pouvez précéder la fonction d'un double `?`. Ceci indiquera à **R** de chercher dans toute la documentation des serveurs de CRAN.

[5] Il existe aussi l'opérateur modulo (`%%`) qui permet de calculer le reste d'une division euclidienne. C'est pratique pour identifier si un nombre est un diviseur d'un autre nombre.

```
# Calcul pour obtenir la moyenne de cet échantillon fictif de 5 cas : 8, 12, 15, 19, 20
> (8 + 12 + 15 + 19 + 20) / 5
```

Mais cette façon de procéder est peu productive. Nous verrons les commandes pour les mesures de la centralité et de dispersion à la section 5.3.2.

5.2.2 Opérateurs d'assignement

Les opérateurs d'assignement, comme leur nom l'indique, permettent d'assigner des valeurs à des objets. Voici un exemple avec la moyenne que nous venons de calculer. Une fois exécuté, la console retournera la réponse « 14.8 » lorsque nous écrivons la moyenne dans l'éditeur et que l'on clique sur « Run ». Vous pouvez voir d'autres exemples de l'éditeur dans la figure 9.

```
# Assigner le calcul précédent à l'objet "moyenne"
> moyenne <- (8 + 12 + 15 + 19 + 20) / 5
```

Lorsque vous utilisez des opérateurs d'assignement, deux options s'offrent à vous. Vous pouvez utiliser le signe d'égalité (`=`) ou utiliser la flèche d'assignement (`<-`). Même si utiliser le signe d'égalité est plus intuitif pour la plupart d'entre vous, il est fortement recommandé d'utiliser la flèche d'assignement (`<-`). Pourquoi? Il s'agit d'un standard de bonne pratique. Souvent en programmation, il y a plusieurs façons d'arriver au même résultat. Les standards de bonne pratique nous permettent d'identifier les pratiques optimales sans avoir à comprendre pourquoi une pratique est meilleure qu'une autre dans toutes ses nuances. Pour ce cours, utilisez la flèche d'assignement (`<-`).

5.2.3 Opérateurs logiques

Les opérateurs logiques retournent l'une de deux valeurs, soit « vrai » ou « faux ». Ils sont donc utilisés pour comparer deux objets. Voici les principaux :

- A `<` B : A est plus petit que B
- A `>` B : A est plus grand que B
- A `<=` B : A est plus petit ou égal à B
- A `>=` B : A est plus grand ou égal à B
- A `==` B : A est égal à B
- A `!=` B : A n'est pas égal à B
- `&` : Une condition **ET** une autre condition
- `|` : Une condition **OU** une autre condition
- A`%in%`B : A est contenu dans B

La figure 10 contient une série de tests logique (lignes 41 à 49) qui peuvent servir d'exemple. Cependant, nous vous suggérons fortement de tenter d'y répondre dans votre tête avant d'exécuter chaque ligne.

- `if` (*condition*) {*suite d'opération*}
- `else` {*suite d'opération*}
- `for` (*facteur d'itération*) {*suite d'opération*}

5.2.4 Instructions de contrôle

Jusqu'à présent, **R** ne se distingue pas beaucoup d'une calculatrice. Les instructions de contrôle sont ce qui rend **R** beaucoup plus puissant que votre calculatrice. En effet, elles permettent d'automatiser des procédures et des calculs. Trois instructions de contrôle principales sont à retenir :

Les instructions de contrôle sont souvent ce qu'il y a de plus difficile à apprendre lorsqu'on commence à programmer. *Leur maîtrise n'est donc pas essentielle pour la réussite de ce cours*. Il est cependant important que vous ayez connaissance de leur existence. Vous avez un exemple à la figure 10.

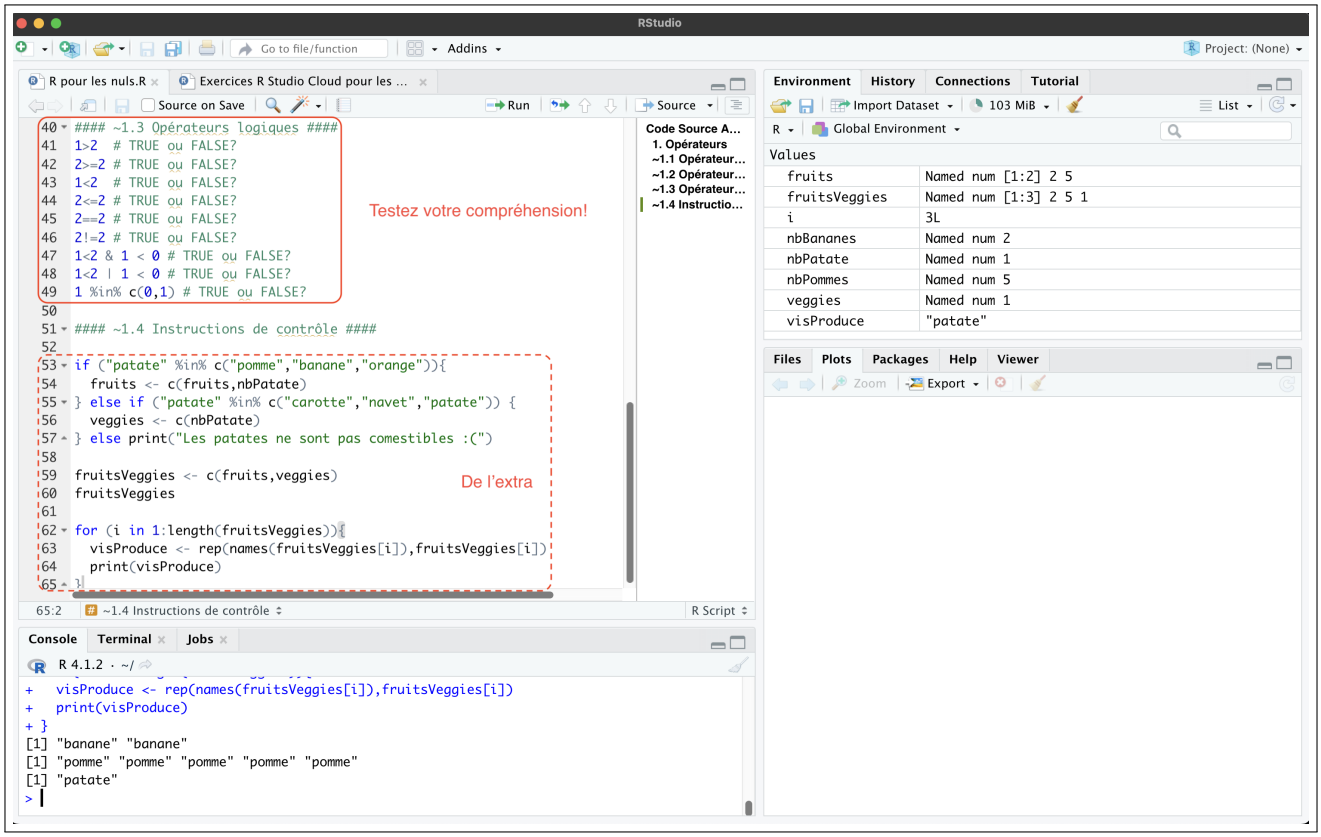


Figure 10: Opérateurs logiques

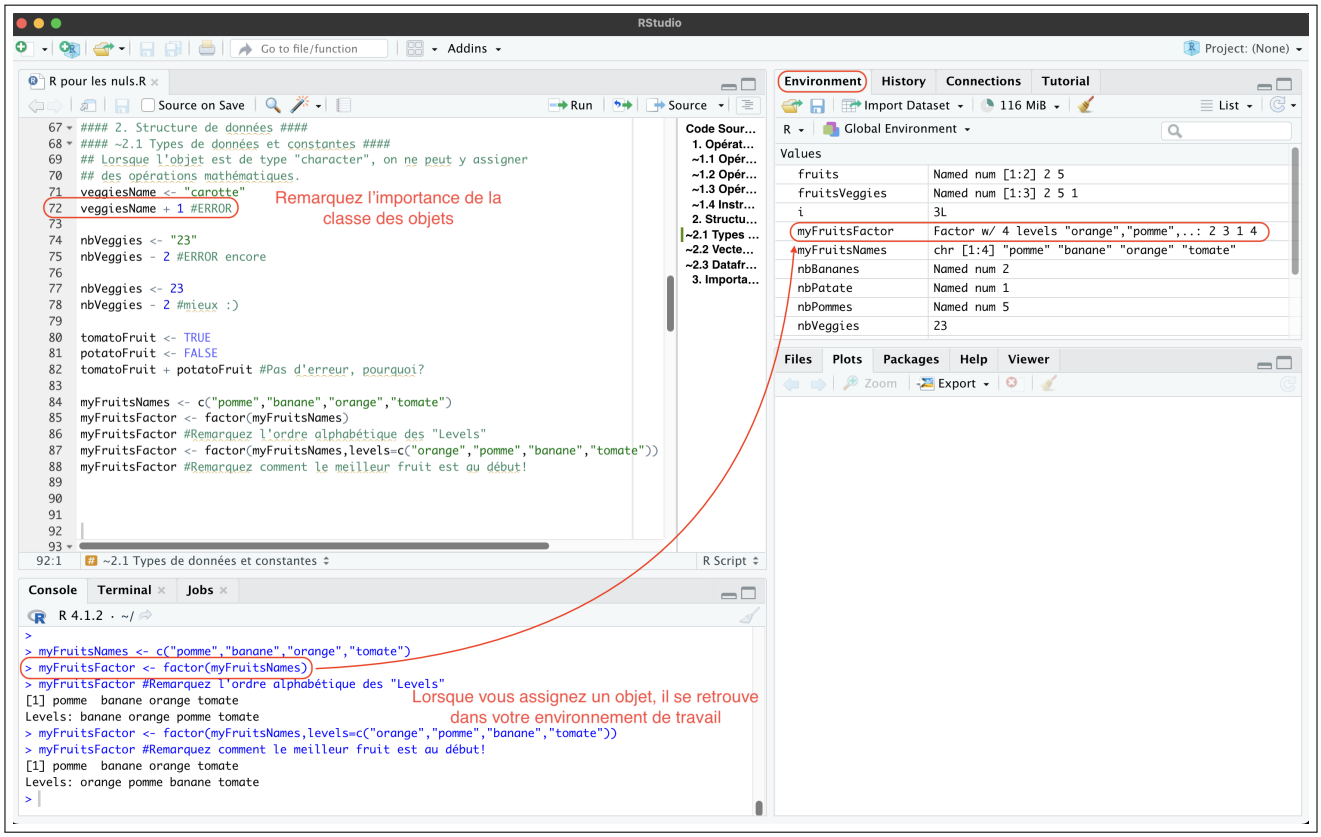


Figure 11: Assigner un objet

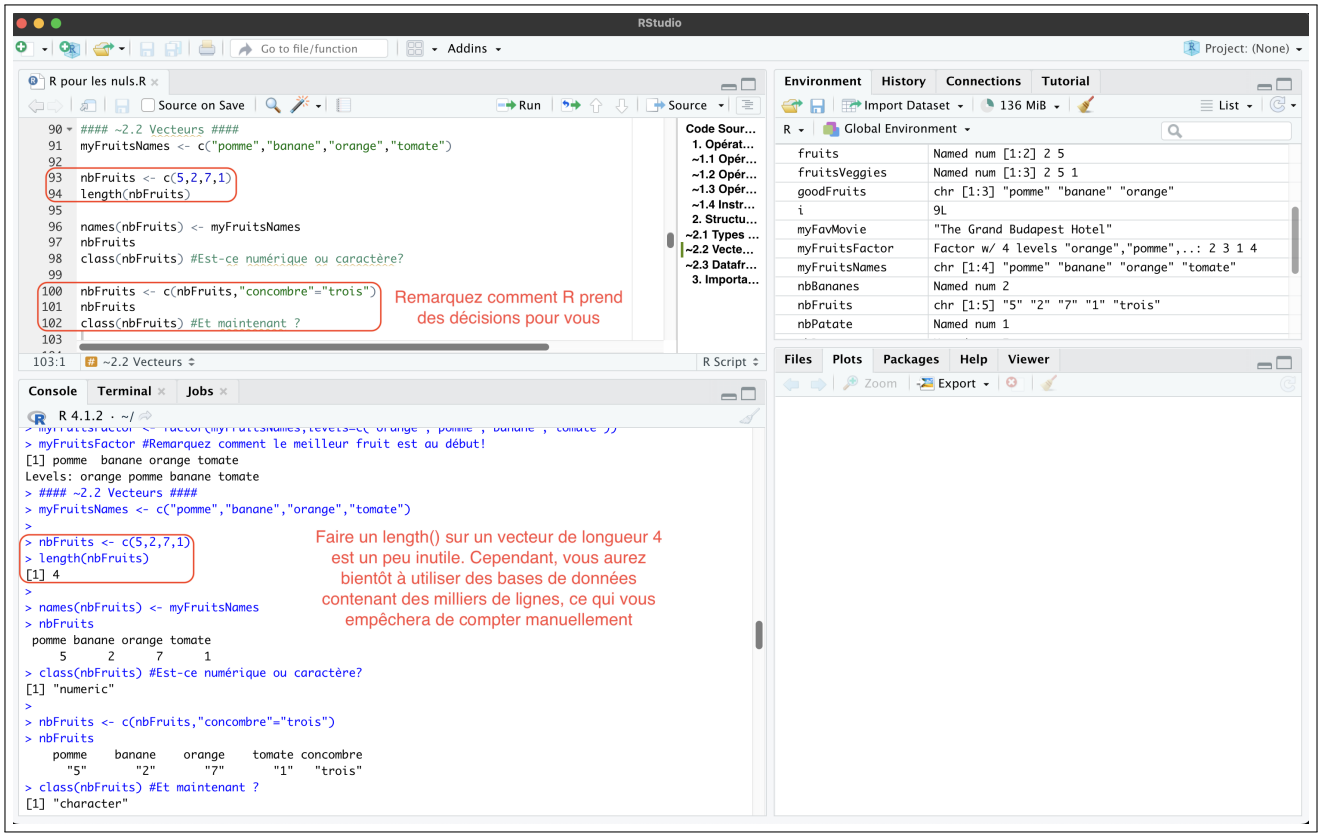


Figure 12: Le vecteur

5.3 Structure de données

Vous savez désormais comment effectuer des opérations de base sur des objets. Or, pour ce faire, il vous faut des objets à exploiter. Ces objets peuvent prendre plusieurs formes ayant chacune des caractéristiques et fonctionnalités différentes qui les rendent plus ou moins adaptés à différent type d'utilisation. Vous verrez ici les trois formats principaux qui seront utilisés dans le cours.

5.3.1 Types de données et constantes

Quatre types de données principales sont à maîtriser. Les données de classe *caractère* (ou «*character*») sont considérées comme du texte par **R**. *Elles ne peuvent donc pas être sujettes aux opérations mathématiques standards*. Par exemple, la figure 11 présente l'objet `veggiesName` qui contient la valeur textuelle *carotte*. Pour assigner une donnée textuelle, il faut l'encadrer de guillemets « "" ». Si vous exécutez la ligne 72, vous obtiendrez une erreur puisque l'opération demandée ne peut pas s'appliquer à la classe de l'objet. Chose importante, même s'il s'agit d'un chiffre, si vous l'encadrez de guillemets comme à la ligne 74, **R** considérera l'objet comme du texte et la même erreur se produira.

Pour faire des opérations mathématiques, il faut utiliser des valeurs numériques ou booléennes. Les valeurs numériques sont des chiffres. Ceci inclut les nombres entiers, rationnels (π ou e), et décimaux. Les valeurs booléennes ont deux valeurs possibles qui correspondent aux réponses possibles des opérateurs logiques, soit `FALSE` ou `TRUE`. Bien qu'à première vue les valeurs booléennes ressemblent à des données textuelles, **R** les comprend comme 0 et 1 respectivement. C'est pourquoi si on additionne `tomatoFruit` et `potatoFruit`, on obtient 1. Pour ceux et celles d'entre vous un peu confus par ceci, rappelez-vous du processus de la codification des données et des guides de référence pour expliquer les bases de données.

Un format alternatif et particulier des données textuelles et numériques est le *facteur*. Conceptuellement, un facteur en **R** est un objet pouvant prendre un nombre limité de valeurs. En d'autres mots, il s'agit d'une variable catégorielle. À ces valeurs il est possible d'imposer un ordre (pensez ici aux variables ordinales, intervalles et ratio). Par défaut, **R** ordonnera les valeurs numériques en ordre croissant et les valeurs textuelles en ordre alphabétique.

En utilisant la fonction `factor()`, il est possible de créer un objet de classe facteur et de lui imposer un ordre avec l'argument *levels*. Si les facteurs vous semblent compliqués pour rien, ils vous seront très utiles lorsque vous voudrez faire de la visualisation de données plus avancée. Il s'agit également d'une source d'erreur importante. Assurez-vous donc de bien connaître la classe des objets avec laquelle vous travaillez.

Il est possible de connaître la classe d'un objet en regardant ce qui se trouve à l'intérieur. Or, ce n'est pas toujours possible ou évident, soit à cause de la taille de l'objet (la quantité de choses qui s'y trouve), soit parce qu'on peut confondre un 1 caractère et un 1 numérique ou une "pomme" caractère et une "pomme" facteur. Pour être absolument certain, il est toujours préférable de demander à **R** en utilisant la fonction `class()`.

Les constantes sont des objets avec une seule donnée d'associée. À la figure 11, `veggiesName`, `nbVeggies`, et `tomatoFruit` sont des constantes, un objet avec un seul élément. Ces constantes n'ont cependant pas la même classe – soit, respectivement, caractère, numérique et booléen. Les constantes forment l'unité de base des structures de données possibles en **R**.

5.3.2 Vecteurs

Vous avez sans doute déjà entendu parler de vecteurs dans vos cours de mathématiques au secondaire. N'ayez crainte, il n'est pas question de la même chose ici. Un vecteur, pour **R**, c'est une série de données du même type. En d'autres mots, c'est une suite de constantes de la même classe mises bout à bout. Un vecteur **R** peut également être compris comme contenant les valeurs d'une variable – des notes d'examen, le positionnement idéologique, ou des noms de fruits. Sans vecteur, il n'est pas possible pour **R** d'effectuer des opérations aussi simples que le calcul d'une moyenne. Autant dire que **R** ne sert pas à grand chose sans vecteur. Heureusement, vous en avez déjà rencontré sans l'avoir remarqué. À la figure 12, `myFruitsNames` est un vecteur de classe « caractère » et de longueur 4. Pour déterminer combien d'éléments contient un vecteur (ou quelle est sa longueur, vous pouvez les compter vous-même ou demander à **R** de le faire pour vous avec la fonction `length()` comme c'est fait à la ligne 94 de la figure 12.

Pour créer un vecteur, il faut utiliser la fonction `c()` et l'assigner à un objet, comme à la ligne 91 de la figure 12. Le `c` de la fonction représente le verbe anglais «concatenate» qui signifie relier beaucoup de choses ensemble dans une chaîne ou une série. Vous pouvez également ajouter des noms aux éléments de votre vecteur. Ces noms n'auront aucune influence sur le traitement des valeurs associées. Par exemple, à la figure 12, j'ai un vecteur avec le nombre de chaque type de fruits dans le bon ordre (`nbFruits`). Si je désire leur associer un nom pour ne pas oublier ce que chaque chiffre représente, je peux utiliser la fonction `names()` comme à la ligne 96 de la figure 12. Cette ligne se lit comme suit : aux noms des éléments de `nbFruits`, assigne les valeurs qui se trouvent dans `myFruitsNames`. Vous pouvez également associer directement les noms de chaque élément à la création du vecteur comme à la ligne 100 de la figure 12.

À la ligne 100 de la figure 12, j'amende le vecteur `nbFruits` en y ajoutant un élément. Remarquez comment je réutilise l'objet `nbFruits` dans l'opération. La ligne 100 se lit : associe à l'objet `nbFruits` un vecteur des valeurs déjà contenues dans `nbFruits` et la valeur "trois" au nom "concombre". Or, n'ai-je pas dit plus haut qu'un vecteur se devait d'avoir le même type de valeurs? **R** est gentil, il vous laisse le faire sans retourner d'erreur. À la place, il prend une décision pour vous en assignant la classe caractère à **TOUS** les éléments du nouveau vecteur. C'est donc important d'être attentif à ce que fait **R**; il prend parfois de mauvaises décisions pour vous.

Lorsque le vecteur est de classe numérique, il est possible d'effectuer des opérations mathématiques plus ou moins complexes. Vous pouvez bien sûr additionner, soustraire, multiplier et diviser par la même valeur tous les éléments du vecteur. Or, **R** vous permet aussi d'effectuer les opérations statistiques de base sans appel à d'autres librairies. La moyenne, la médiane, l'écart-type, la variance et plusieurs autres opérations ont des fonctions associées. Vous pouvez les consulter à la figure 13. Je vous encourage à explorer le code et à expérimenter.

Le dernier élément important à aborder au sujet des vecteurs est la façon de sélectionner des éléments. Pour ce faire, on utilise les crochets "[]". Ceux-ci nous permettent d'indiquer la position dans le vecteur des éléments à sélectionner. La figure 13 montre, dans l'ordre à partir de la ligne 126, comment sélectionner un élément, une suite d'éléments et plusieurs éléments distincts. Il est également possible d'apposer un `-` devant le chiffre pour retirer l'élément.

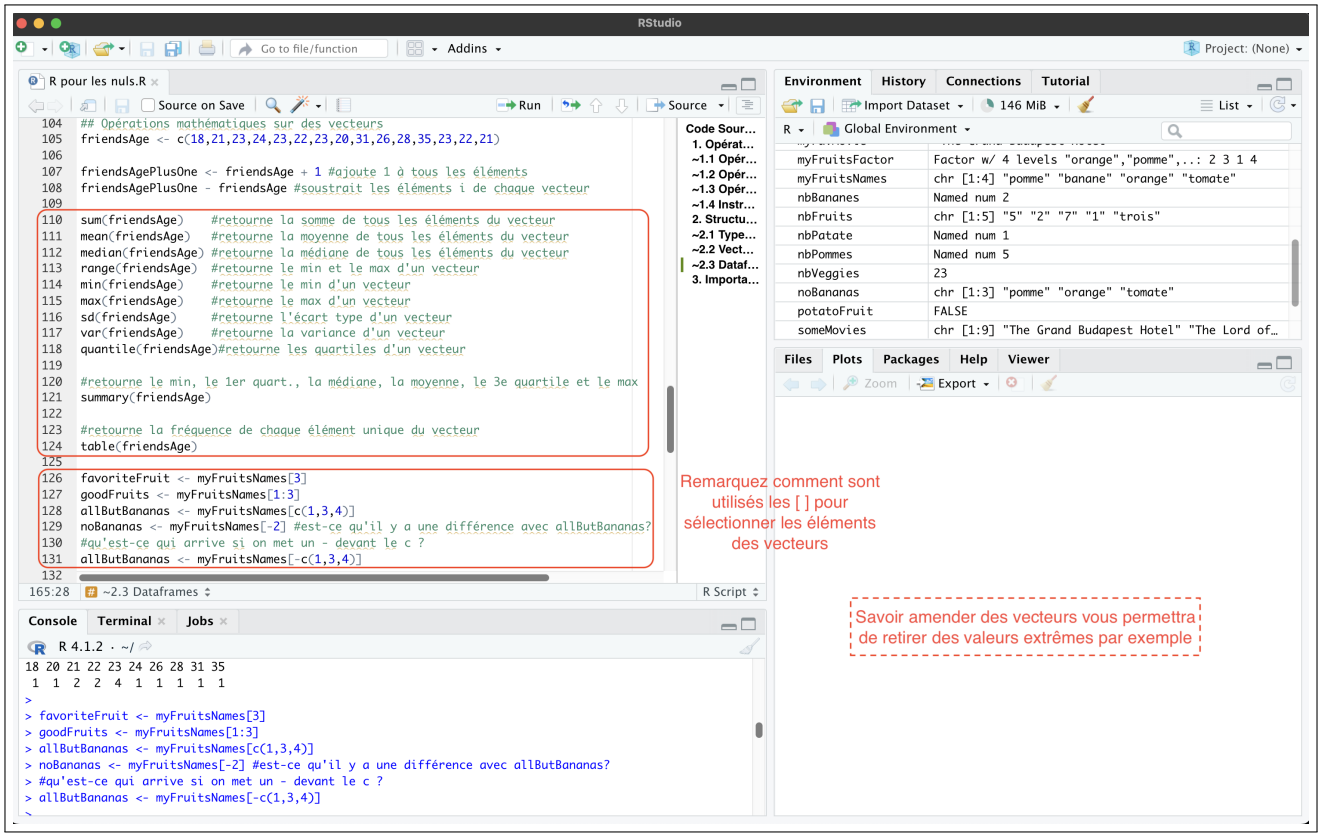


Figure 13: Opérations mathématiques de base

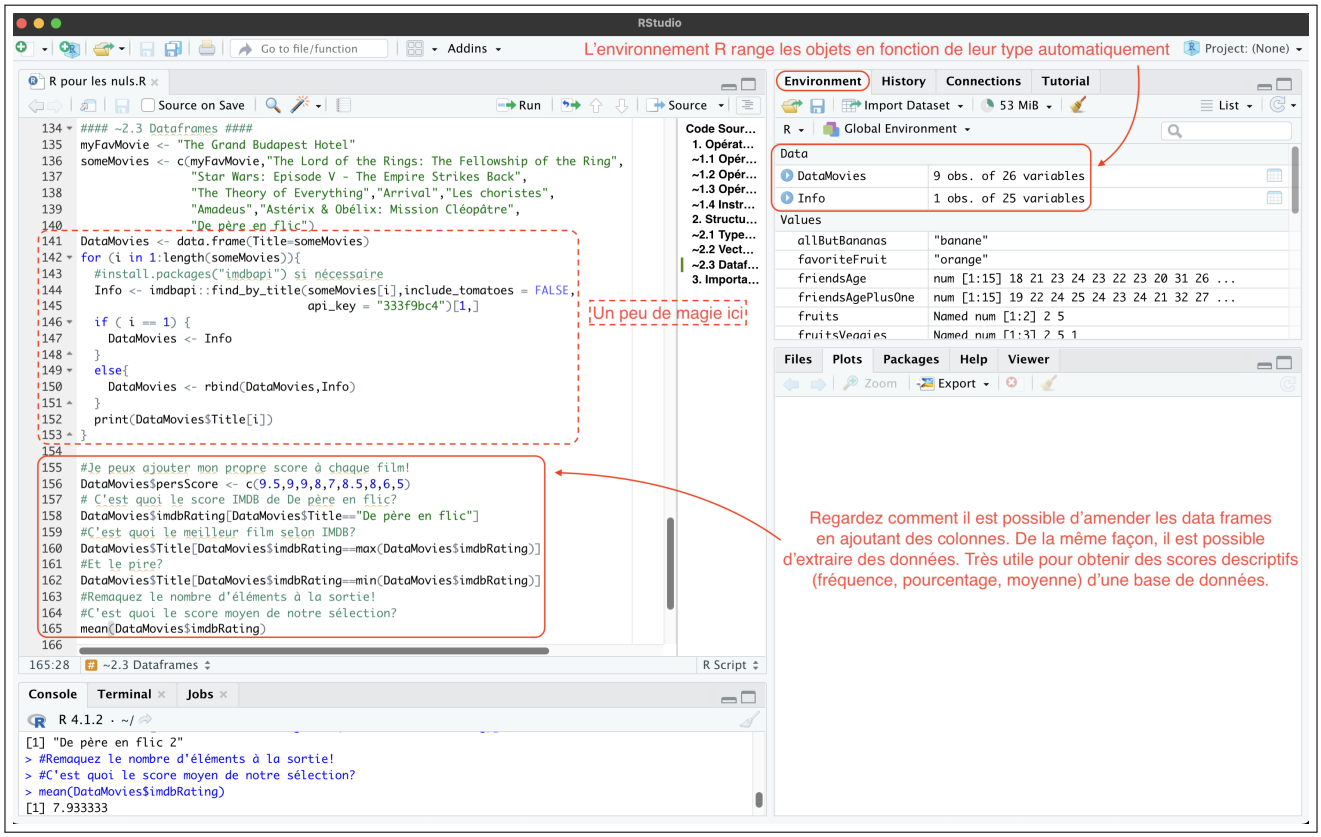


Figure 14: Le *data frame*

5.3.3 Data frames

Un *data frame* (il n'y a pas vraiment de traduction satisfaisante, mais le terme base de données est aussi communément utilisé) contient des données organisées en rangées et en colonnes, comme un tableau Excel. Chaque colonne peut avoir une classe différente, mais les données au sein d'une même colonne doivent être de la même classe. *Généralement, les colonnes seront associées aux variables, tandis que chaque rangée correspondra à une observation.*

Les *data frames* sont composés de vecteurs de la même façon que les vecteurs sont composés de constantes. On peut donc avoir un film préféré (une constante), une liste de films que nous avons vus (un vecteur) ainsi qu'un fichier Excel regroupant ces films, leur année de publication, leur genre, leur score IMDB, etc.[6] Bien qu'il soit possible de créer des *data frames* par vous-même, il est rare qu'un chercheur construise manuellement sa base de données directement dans **R**. Le plus souvent, une base de données existante est téléchargée ou créée dans d'autres logiciels comme un fichier Excel.

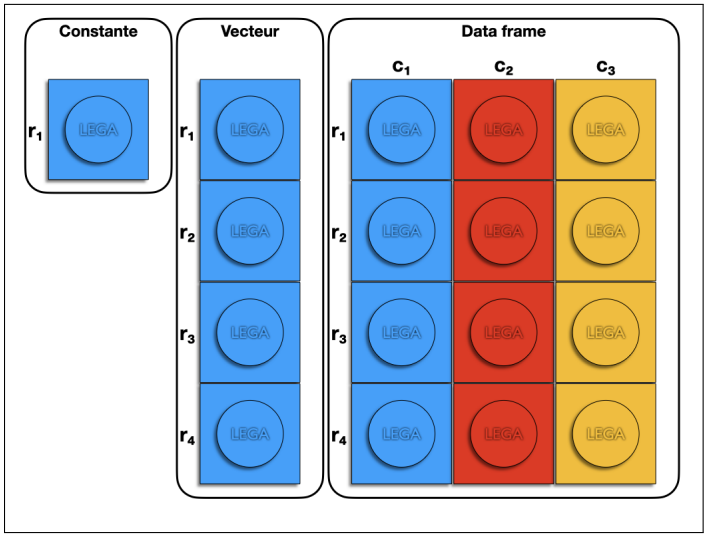


Figure 15: Représentation des types d'objets **R**

Comme les constantes et les vecteurs sont à la base du fonctionnement de **R**, le *data frame* est l'objet le plus important pour le travail des analystes de données. En effet, les données que vous analyserez ne viendront pas sous forme de vecteur, ce serait un cauchemar d'organisation. Imaginez avoir à calculer la moyenne d'un.e étudiant.e pour un cours avec 20 évaluations. Créer un vecteur avec les notes de tous les étudiant.es pour chaque évaluation serait beaucoup trop inefficace. La solution serait de créer un fichier Excel avec une colonne pour les noms des étudiant.es et une colonne par évaluation. Ainsi, au lieu d'avoir 20 vecteurs nommés (rappelez-vous de la fonction `names()` à la figure 12), vous auriez 1 fichier avec 21 colonnes. Ceci vous permet non seulement de calculer la moyenne par étudiant, mais également de croiser vos données. Par exemple, vous pourriez ajouter l'âge des étudiant.es ou leur nombre de sessions universitaires complétées afin d'évaluer si l'une de ces variables semble avoir un effet sur la réussite du cours. C'est ici que la puissance de **R** peu vraiment être exploitée, et c'est pourquoi il est très important de bien comprendre comment manipuler le *data frame*. Il en va de votre capacité à analyser les données. Les prochains paragraphes pourront paraître plus obtus, mais il faut faire un effort pour bien les comprendre.

L'aspect important à reconnaître, c'est qu'un *data frame* a deux dimensions au lieu d'une (les rangées et les colonnes). Pour se déplacer dans un *data frame*, il faut donc ajouter un élément à nos crochets (`[]`) habituel. Dans **R**, le premier élément retrouvé dans les crochets correspond au numéro de la rangé, le second au numéro de la colonne. Essentiellement les r et les c de la figure 15. Bien que **R** soit un outil puissant, il nous demande souvent de faire des abstractions, comme d'indiquer exactement la rangée et la colonne qui nous intéresse. Une manière de visualiser ceci est d'imaginer un fichier Excel ou un jeu d'échecs qui vous indique les colonnes avec des lettres et les rangées avec des chiffres. Ainsi, si je m'intéresse au deuxième élément de la troisième colonne d'un *data frame* nommé `Data`, je peux l'extraire avec `Data[2,3]`. Une erreur que font plusieurs codeurs est d'inverser rangées et colonnes. Il faut se souvenir: rangé, colonne, ou `Data[r,c]`!

Il est cependant rare que nous nous intéressions à des colonnes ou à des cellules particulières d'un *data frame* en termes de numéros de rangée et de colonne. Le plus souvent, nous voulons faire une opération sur une colonne comme une moyenne afin d'avoir des statistiques descriptives. À la figure 14, nous nous intéressons à la moyenne des scores IMDB d'une sélection de films. Pour ce faire, nous utilisons le `$` pour dire à **R** que nous nous intéressons à la colonne `imdbRating` de `DataMovies` (`Data$nomDeLaColonne`).

De la même façon, il est également possible de combiner le `$` avec les crochets (`[]`) pour sélectionner des éléments d'un *data frame* sous certaines conditions. Ceci est un outil puissant pour explorer vos données. Dans le cas du

[6] Vous remarquerez que je fais appel au *package* `imdbapi` à la figure 14. Il s'agit d'une librairie contenant des fonctions qui me permettent de communiquer avec l'API d'IMDB et de lui faire des requêtes. Dans le code, je fournis une clé pour y accéder, mais pour celles et ceux qui désireraient leur propre clé c'est ici ↦ http://www.omdbapi.com/apikey.aspx. Pour de la documentation sur le package c'est là ↦ https://cran.r-project.org/web/packages/imdbapi/imdbapi.pdf.

score IMDB de « De père en flic », la ligne se lit : dans la colonne `imdbRating` du *data frame* `DataMovies`, sélectionne les éléments dont la valeur à la même rangée dans la colonne `Title` du *data frame* `DataMovies` est égale à "De père en flic". Les possibilités de conditionnalité sont donc infinies. Dans l'exemple on s'intéresse aux scores maximum et minimum, mais il est possible de combiner des tests booléens (vrai ou faux, `TRUE`/`FALSE`) avec les arguments conditionnels " `|` " (OR) et " `&` " (AND).

Enfin, les mêmes opérations mathématiques décrites à la figure 13 pour les vecteurs s'appliquent aux colonnes des *data frames*. C'est normal, rappelez-vous, une colonne d'un *data frame* c'est comme un vecteur. Il n'y a qu'à choisir la colonne avec le `$` et on peut appliquer les fonctions de moyenne, d'écart-type, de variance, etc.

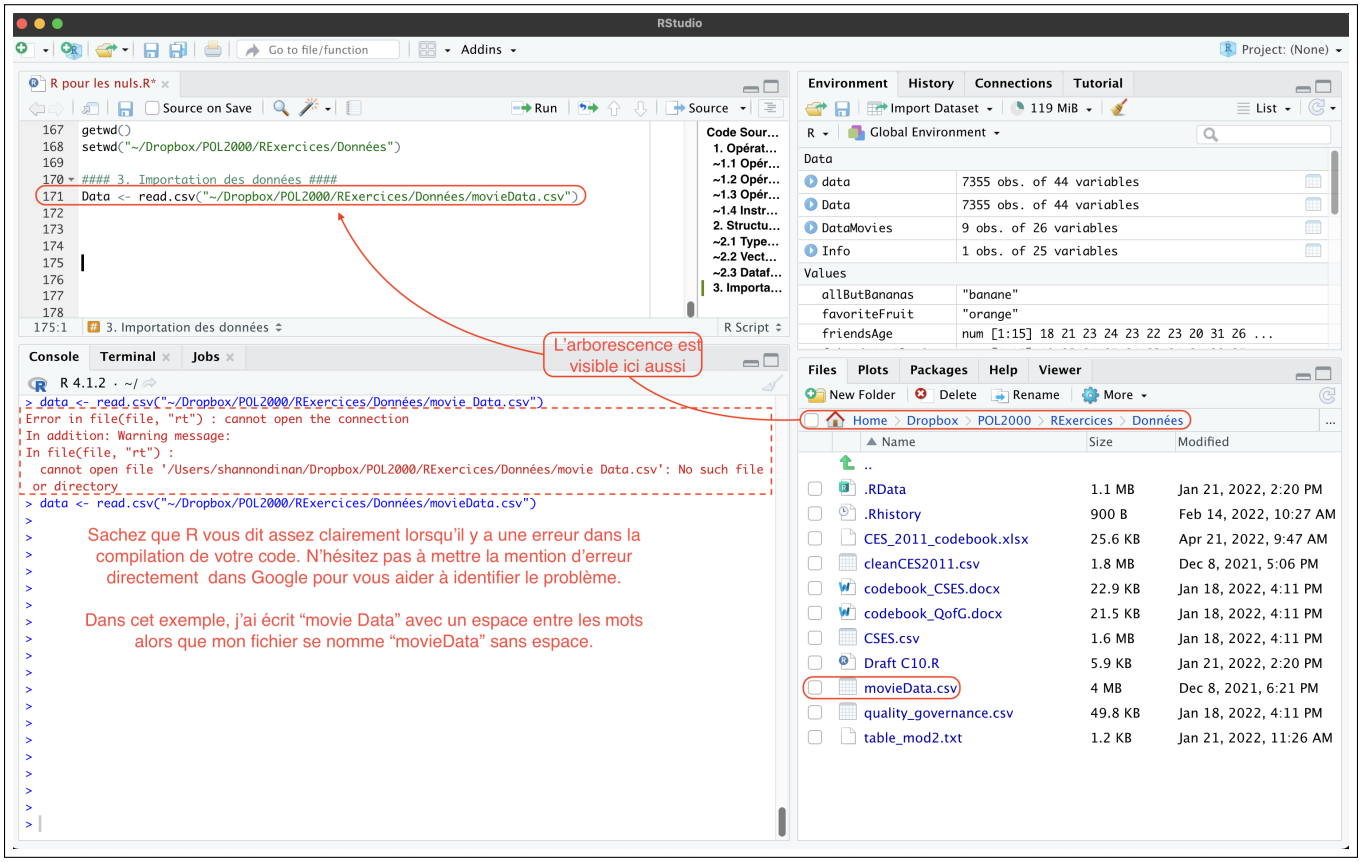


Figure 16: Importer des données à un environnement de travail

6 Importation de données

6.1 Importation de données

Avant de débuter l'analyse, il faut des données à analyser. En science politique, les bases de données auxquelles vous ferez face seront typiquement rectangulaires. Elles auront donc la même structure qu'un *data frame* tel que compris par **R** : un cas par ligne, une variable par colonne. Si vous êtes familier avec Excel, vous reconnaîtrez certainement cette structure.

R étant un langage *open source*, la communauté de développement donne accès à plusieurs librairies (*packages*) dédiées à la lecture de différents formats de données. Dans le cadre du cours, vous n'aurez uniquement qu'à traiter des données enregistrées en format `.csv`, valeurs séparées par des virgules (*comma-separated values*). C'est ce qui se cache derrière les tableaux Excel. Or, les formats `.sav`, `.spss`, `.rds`, et `.tsv` sont également populaires. Si vous rencontrez un jour l'un de ces formats, il est certain qu'une fonction pour les ouvrir existe déjà. À ce moment, **Chat GPT** et **Google** seront vos meilleurs amis! Une fois la bonne fonction identifiée, la procédure est assez simple si l'on comprend bien le concept d'arborescence. Je vous invite donc à réviser la section **??** avant de continuer.

Pour ouvrir un fichier `.csv`, il faut exécuter la fonction `read.csv()` et assigner le résultat à un objet (typique-

ment appelé *Data*) comme le montre la figure 16. Il est important de constater que le seul argument requis est l'arborescence menant au fichier `.csv` que vous voulez ouvrir. Dans le cas de la figure 16 à la ligne 171, il s'agit d'une arborescence d'ordinateur Mac typique. Pour un PC, un `C:` aurait été ajouté devant `/Users`.

Afin d'importer vos données avec RStudio, vous devez vous souvenir de l'arborescence que vous avez initialement établie (voir la section 4.1.1). Si vous ne vous souvenez pas, vous pouvez écrire le code ci-dessous. R vous retournera une ligne dans la console vous indiquant votre « répertoire de travail » ou *working directory*. Vous pouvez aussi consulter la partie de l'interface avec dossiers, graphiques, librairies, aide, etc. comme visible dans la figure 16.

```
# Comment trouver son répertoire de travail
> getwd()
```

Une fois que vous vous souvenez de ceci, vous pouvez indiquer à R où aller chercher vos données. Une bonne pratique est de télécharger toutes les bases de données du cours (disponibles sur le Portail) et les mettre dans un fichier commun sur votre ordinateur nommé « données » ou « data ». De cette manière, il sera facile de réutiliser la même ligne de code en changeant le nom de la base de données nécessaire. Par exemple :

```
# Importation des données
> data <- read.csv("~/POL2000/R/data/movieData.csv")
```

7 Analyse - Description

L'analyse descriptive des données est l'outil de base du chercheur voulant explorer une base de données. Cette section présentera les principaux tests, visualisations et mesures d'analyse descriptive.

7.1 Univariée

Une analyse univariée est, comme son nom l'indique, une analyse s'effectuant sur UNE SEULE variable. Vous avez déjà rencontré les principales mesures lui étant associé à la figure 13. En effet, ce qui nous intéresse dans ce genre d'analyse est la distribution d'une variable. Est-elle distribuée normalement ou est-elle asymétrique? Quel est son mode? Sa variance? Les **mesures de dispersion** permettent d'identifier le positionnement des observations sur la distribution et de répondre à ces questions. Ainsi, pour obtenir la moyenne d'une variable, il faut utiliser la même fonction qu'à la figure 13 en indiquant cette fois la bonne colonne du *data frame* où se trouve la variable d'intérêt (`mean(Data$nomDeLaVariable)`). Rappelez-vous: les colonnes d'un *data frame* ne sont que des vecteurs déguisés!

7.2 Bivariée

Vous vous en doutez certainement, si une analyse univariée se concentre sur une seule variable, une analyse bivariée en compare deux. Étonnant, n'est-ce pas?! Plusieurs tests existent pour comparer deux variables, nous vous en présenterons deux : le test-t de Student et le r de Pearson.

Le test-t de Student permet d'effectuer un test d'hypothèse statistique. Il est souvent utilisé pour déterminer si les distributions (ou les moyennes) de deux variables sont significativement différentes. Par exemple, si on s'intéresse à la différence de revenu collecté pour différents types de films, nous pouvons tester les moyennes grâce à un test-t. La figure 17 montre comment utiliser la fonction `t.test()`. Celle-ci prend comme arguments principaux deux variables. Dans notre cas, un vecteur du revenu des films romantiques et un vecteur du revenu des films d'action. Remarquez comment sont spécifiés les vecteurs d'intérêt. Si nous utilisions uniquement `Data$revenue` en x et en y, le test-t ne remarquerait aucune différence puisque c'est la même variable. Il est également possible de spécifier l'intervalle de confiance désiré. N'oubliez pas, si vous souhaitez plus d'informations sur l'utilisation d'une fonction, inscrivez et exécutez `?maFonction()` dans la console. Une fenêtre d'information s'ouvrira.

Le r de Pearson est un test de corrélation simple et pratique qui vous permettra de tester la relation entre deux variables. Distinguez bien les deux concepts. Le test-t permet de tester si deux distributions sont différentes. La corrélation permet, elle, de tester s'il y a une relation entre deux distributions. Attention: une corrélation entre deux variables ne veut pas dire qu'il y a causation! La fonction **R** pour ce test est utilisée de la même manière que

pour le test-t. Il suffit d'identifier les arguments x et y (les deux variables) de la fonction `cor.test()` comme le montre la figure 18 et le tour est joué.[7]

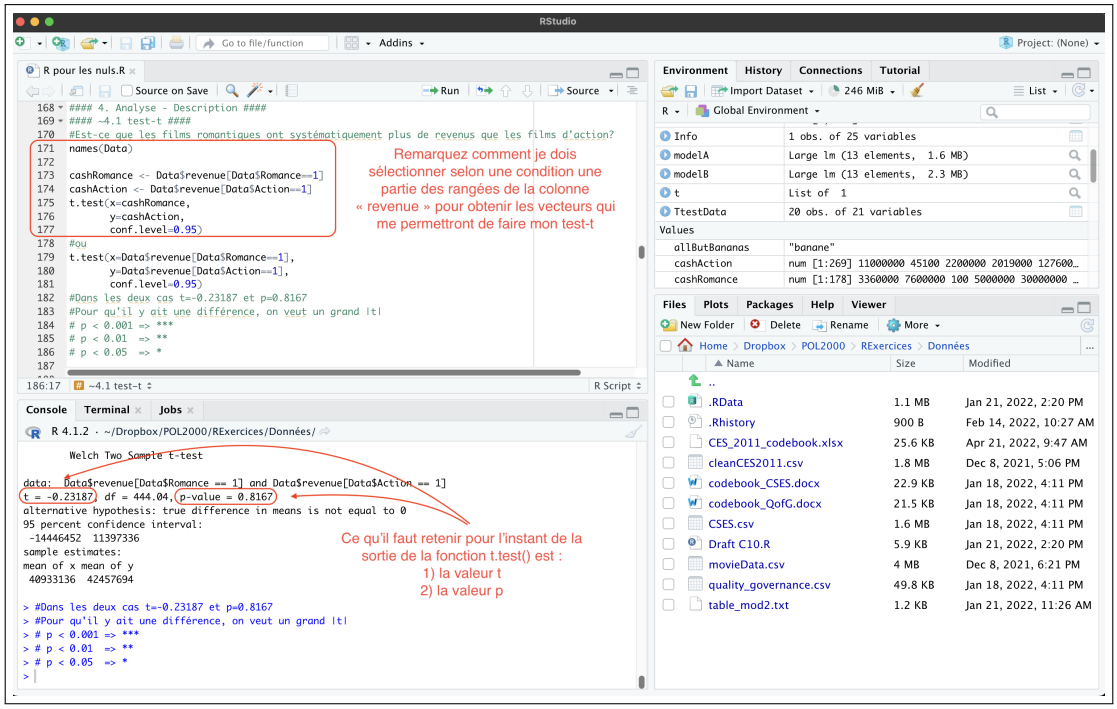


Figure 17: Test t de Student

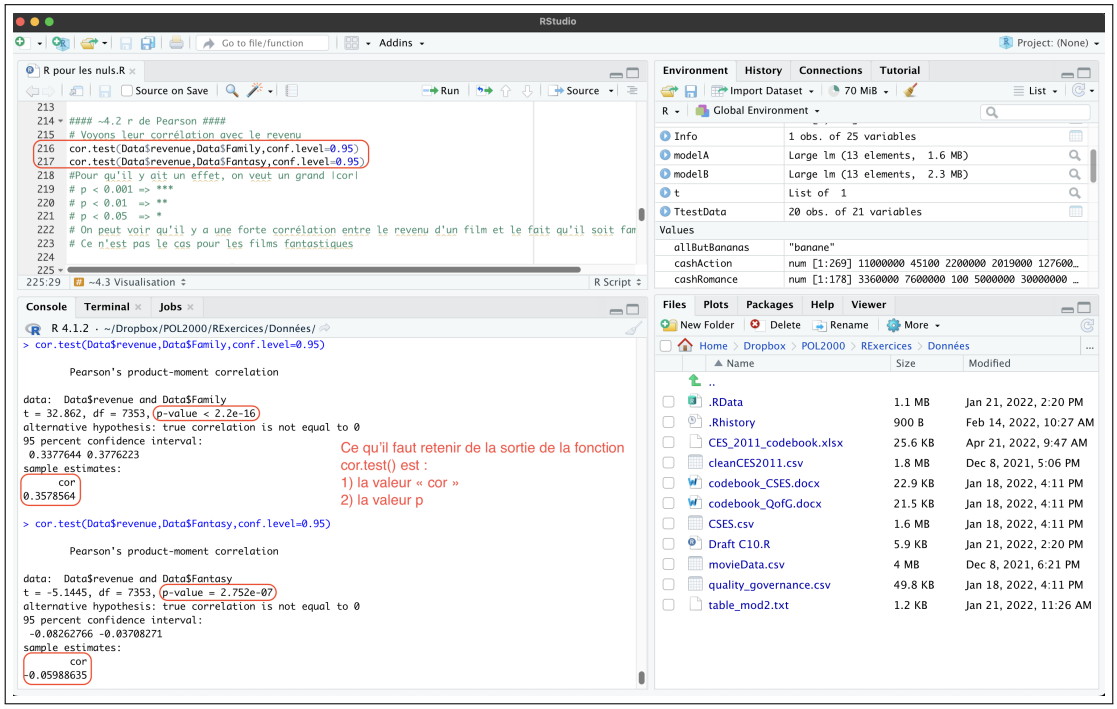


Figure 18: Test de corrélation (r de Pearson)

7 Ne vous inquiétez pas lorsque R vous retourne un résultat avec un e, comme les valeurs p dans la figure 18. Le symbole *e* est la notation standard pour les puissances de 10. Autrement dit, une valeur p de 2.2e-16 signifie 0.0000000000000022 et est très près de zéro.

8 Analyse - Régression linéaire simple et multiple

La régression est sans doute l'outil le plus utile de l'arsenal de l'analyse quantitative. Elle est à la base de plusieurs modèles statistiques plus avancés allant même jusqu'à l'apprentissage machine. Le principe de la régression devrait vous être plus familier à la suite du cours. Cette section se concentrera donc sur la façon d'en faire avec **R**.

La fonction `lm()` permet de calculer un modèle de régression comme le montre la figure 19. Le premier argument est toujours la variable dépendante (ou variable prédite) suivie d'un tilda (~) et de la ou des variables indépendantes. Celles-ci s'ajoutent à l'aide du `+`. Une fois toutes les variables du modèle spécifiées, il faut ajouter une virgule suivie de l'argument `data=` et du bon *data frame*. Attention, il est important d'utiliser le même nom de variable que les noms associés aux colonnes du *data frame* en question. À la figure 19, on assigne l'extrant de la fonction à un objet appelé `modelA`. De cette façon, il est possible d'utiliser la fonction `summary(modelA)` pour obtenir plus d'informations sur le modèle. La figure 20 présente un modèle de régression multiple. La différence? Il y a plus de variables après les `+`. Voilà! Vous savez coder un modèle de régression linéaire en **R** maintenant. C'est aussi simple que ça!

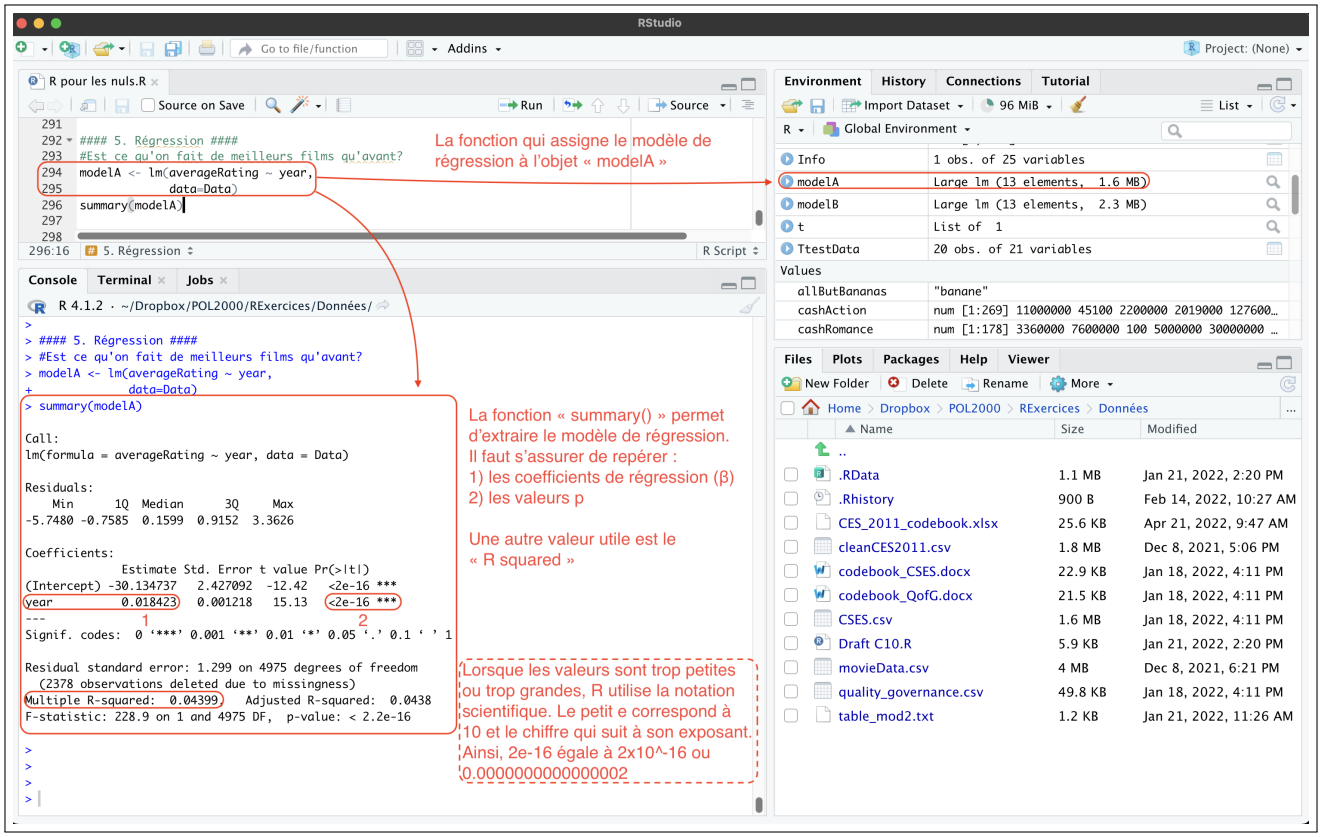


Figure 19: Régression linéaire simple

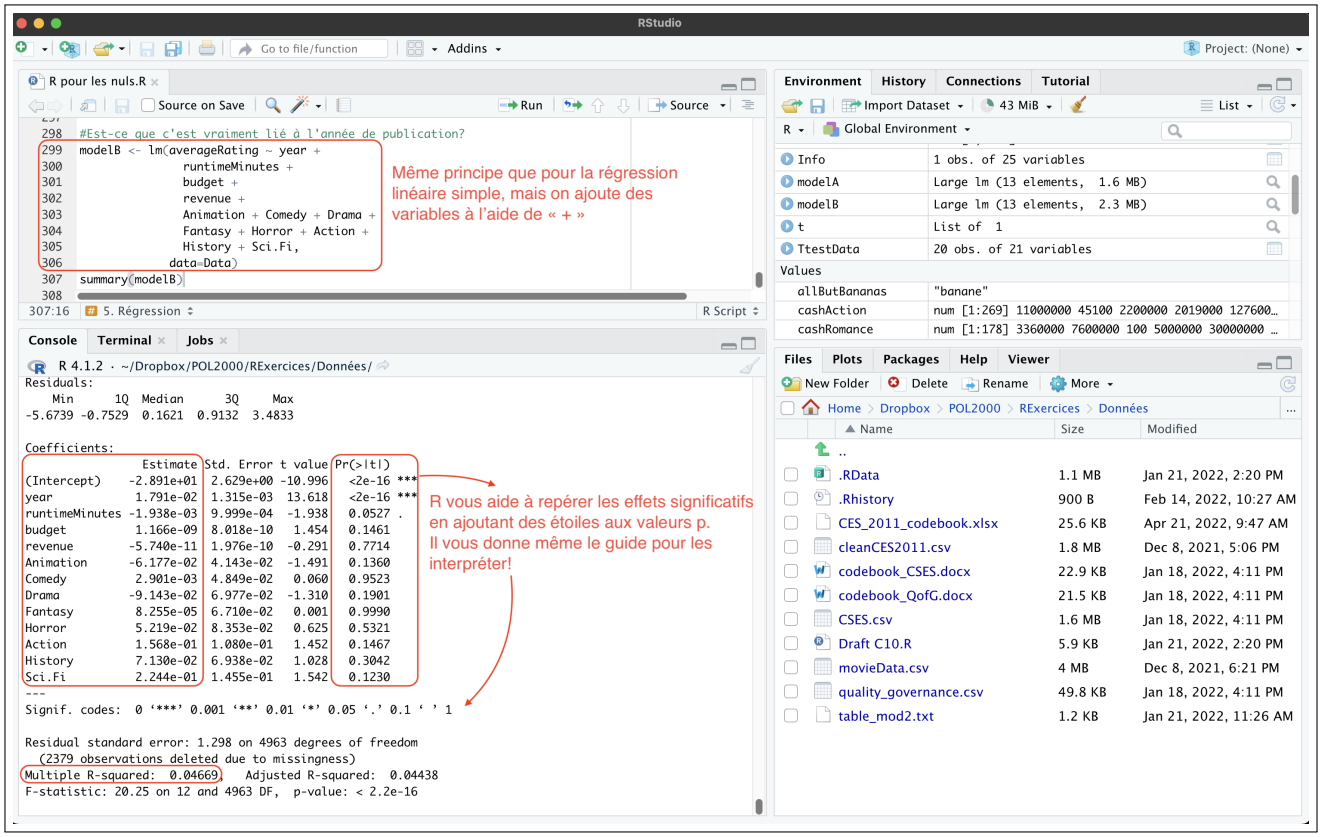


Figure 20: Régression linéaire multiple

9 Visualisation

Bien souvent, il est difficile de se souvenir de l'interprétation des différents tests et les chiffres sont rarement le meilleur moyen de transmettre l'information à un plus large public. Grâce à **R** et la fonction `ggplot()`, il est facile de présenter visuellement les résultats de nos analyses.[8] Nous ne ferons ici qu'une introduction aux principales fonctionnalités de *ggplot2*. Pour une revue exhaustive des fonctionnalités, consultez ce lien.[9]

Tout d'abord, vous devez télécharger **tidyverse**, une collection de packages R qui inclut **ggplot**, et demande à R de chercher ce package dans votre librairie.[10] Pour ce faire, regardez la figure 21 et écrivez les lignes de code suivantes :

```
# Télécharger un package
> install.packages("tidyverse")
# Demander à R d'utiliser un package disponible dans votre librairie
> library(tidyverse)
```

8 En fait, `ggplot()` est une fonction issue de la librairie *ggplot2*, elle même issue de la meta-librarie *tidyverse*.

9 Le voici si vous lisez sur papier https://ggplot2.tidyverse.org/reference/

10 Les autres packages nécessaires pour ce cours seront téléchargés en classe.

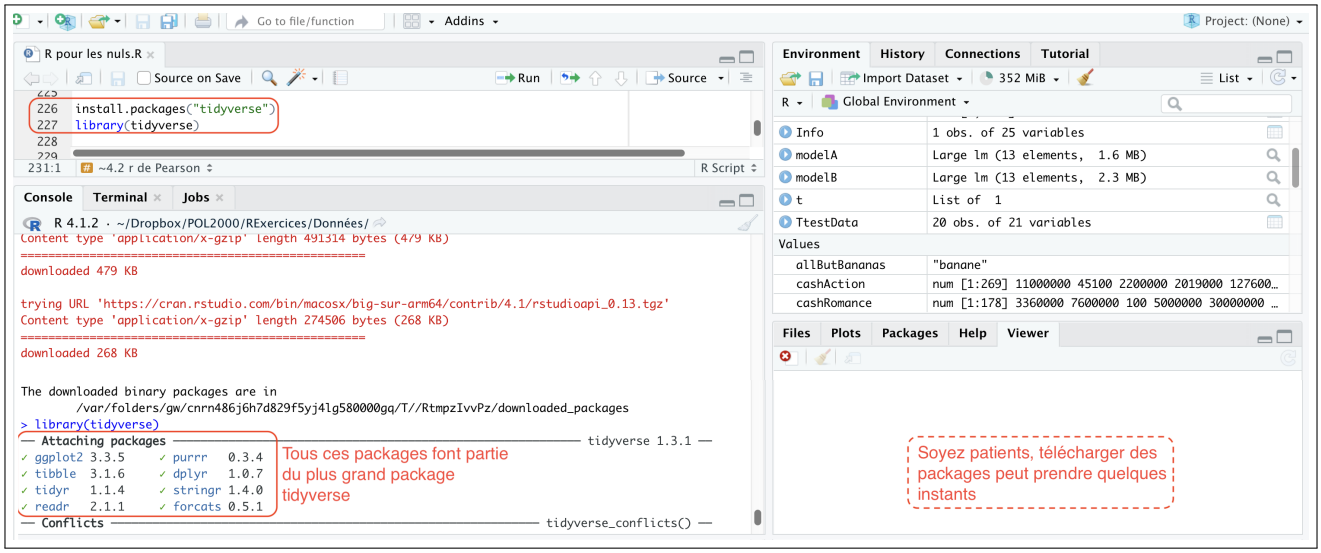


Figure 21: Tidyverse

Vous pouvez maintenant utiliser `ggplot()`. `ggplot` s'attend à recevoir au moins 3 éléments présentés aux lignes 238-39 de la figure 22 : 1) un *data frame*, 2) un *aesthetic*, et 3) un `geom`. Le *data frame* contient les informations qui composeront le graphique. Il peut s'agir de la base de données originale, d'un sous-échantillon ou même des résultats d'une analyse quelconque (une suite de moyennes par année, par exemple). L'*aesthetic* (ou `aes`) contient les arguments permettant d'identifier le rôle des variables d'intérêt du *data frame*. C'est là où sont spécifiés l'axe des X, l'axe des Y, et la façon d'associer les groupes par couleur ou par texture. Par exemple, aux lignes 238-39 de la figure 22, il est seulement spécifié que la variable `runtimeMinutes` correspond à l'axe des X. Pourquoi n'est-il pas spécifié un axe des Y ? C'est là qu'entrent en jeu les `geom`. Ce sont essentiellement les fonctions qui vous permettent de spécifier ce qui doit être dessiné. Voici une liste des principaux `geom` :

- `geom_histogram()` ↦ un histogramme
- `geom_bar()` ↦ un graphique à barres
- `geom_col()` ↦ un graphique à barres[11]
- `geom_point()` ↦ un nuage de points
- `geom_smooth()` ↦ une courbe de régression

Ce qu'il faut bien comprendre, c'est que les arguments de l'*aesthetic* doivent bien s'agencer avec le `geom` voulu. Par exemple, un histogramme n'a pas besoin d'une position en Y puisque le `geom` calcule la fréquence lui-même. Or, pour un nuage de points, il faut spécifier à quelle variable correspond l'axe des X et l'axe des Y. Ainsi, le *data frame*, l'*aesthetic* et le `geom` doivent être en harmonie pour que la fonction retourne le graphique désiré. Pour vous faciliter la vie, je vous propose de toujours répondre à ces 3 questions avant de tenter la production d'un graphique :

1. Comment vous imaginez-vous le graphique ?
2. Quelles sont les variables d'intérêt ?
3. Est-ce que mon *data frame* a le bon format ?

De cette façon, vous vous assurerez de ne pas faire d'erreurs de logique lors de l'écriture de la fonction. Bien entendu, ces trois questions et les `geom` ne vous permettent d'accéder qu'à une partie de l'ensemble des possibilités et des fonctionnalités qu'offre `ggplot`. Que ce soit les couleurs, la taille des points ou des barres, l'espacement des barres, le nom des axes, leur gradation, l'ajout d'un titre, d'un sous-titre, d'une note en bas de page, d'annoter le graphique, d'ajuster la légende, la taille et l'angle du texte, de modifier la grille, la couleur de fond, etc. Tout ceci s'apprend sur le tas à l'aide de Google. Si vous avez une question sur une idée de graphique, forte chance que quelqu'un se soit déjà posé cette question. Demandez à Google!

[11] La différence entre `geom_bar()` et `geom_col()` est que `geom_bar()` prend une fonction en Y (`..count..` par exemple) alors que `geom_col()` prend une variable.

Le dernier concept à bien comprendre pour utiliser `ggplot()` à son plein potentiel est le concept de couche. C'est ce qui rend `ggplot()` aussi versatile. Vous avez sans doute remarqué les petits `+` à la fin de chaque ligne à la figure 22. Ceux-ci se traduisent pour **R** comme «ajoute ce qui suit au-dessus de ce qui se trouve avant». Il est donc possible de superposer les `geom` afin de présenter plus d'informations aux lecteurs ou de mettre en relief certaines informations. Par exemple, à la figure 23, un `geom_smooth()` est ajouté par-dessus un `geom_point()` et un `geom_line()` afin de présenter l'augmentation du nombre de films produits au fil des années. La figure 25 quant à elle présente le code du graphique de la figure 24. Vous y trouverez un code de visualisation plus avancé qui cartographie les fonctions principales de modification de l'aspect visuel du graphique. Il est fortement recommandé d'exécuter le code ligne par ligne afin d'identifier ce que fait chaque partie de la fonction. Comme pour toute chose, c'est en pratiquant que vous apprendrez!

La fonction `ggsave()` permet d'enregistrer sous différents formats (pdf, png, jpeg, gif, mp4, etc.) les graphiques produits à l'aide de `ggplot()`. Pour ce faire, il faut spécifier l'arborescence en la terminant par le nom du fichier suivi de l'extension (.png) désirée. Le ratio largeur:hauteur (*width:height*) peut être spécifié ou non. Dans le cas de la figure 24, le ratio spécifié permet de respecter le standard des images Twitter de 16:9.

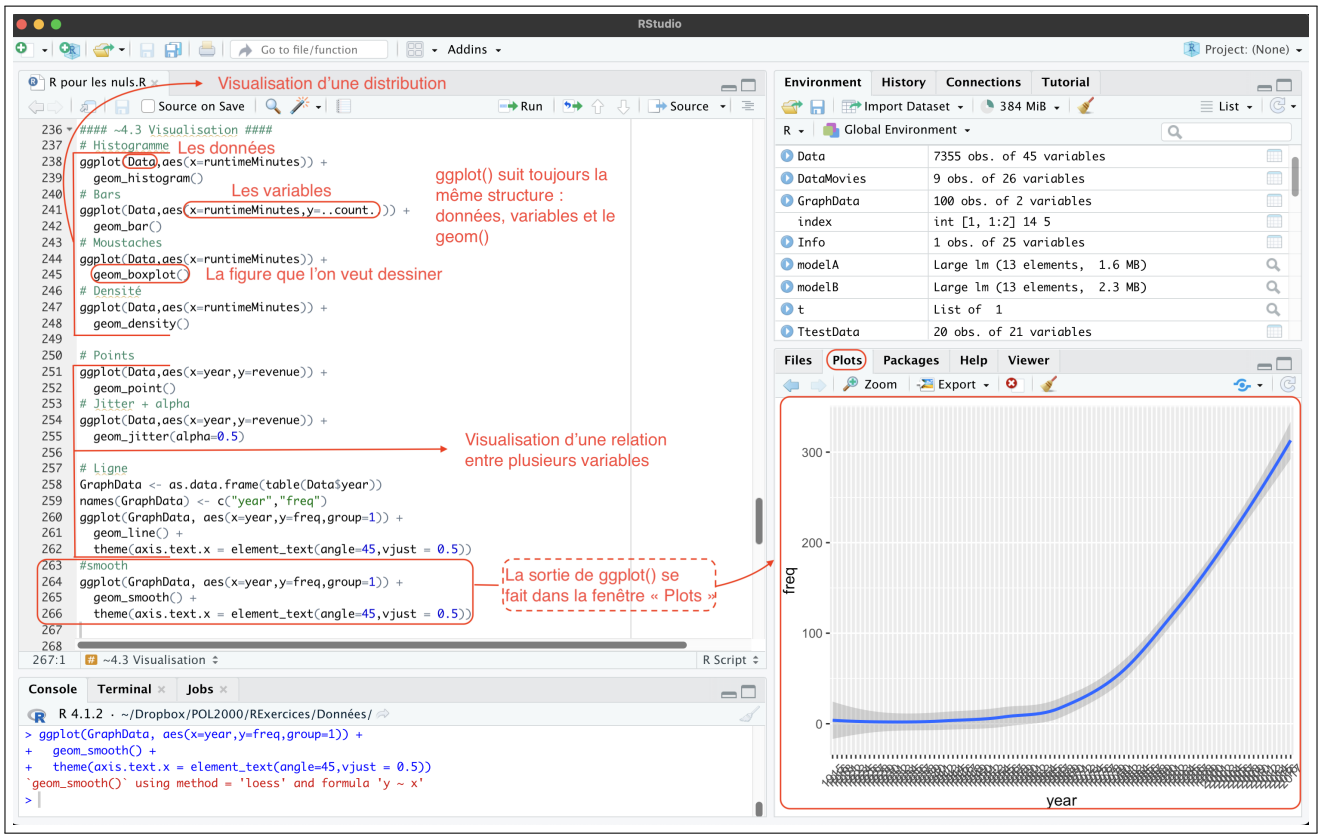


Figure 22: Quelques exemples d'utilisation de `ggplot2`

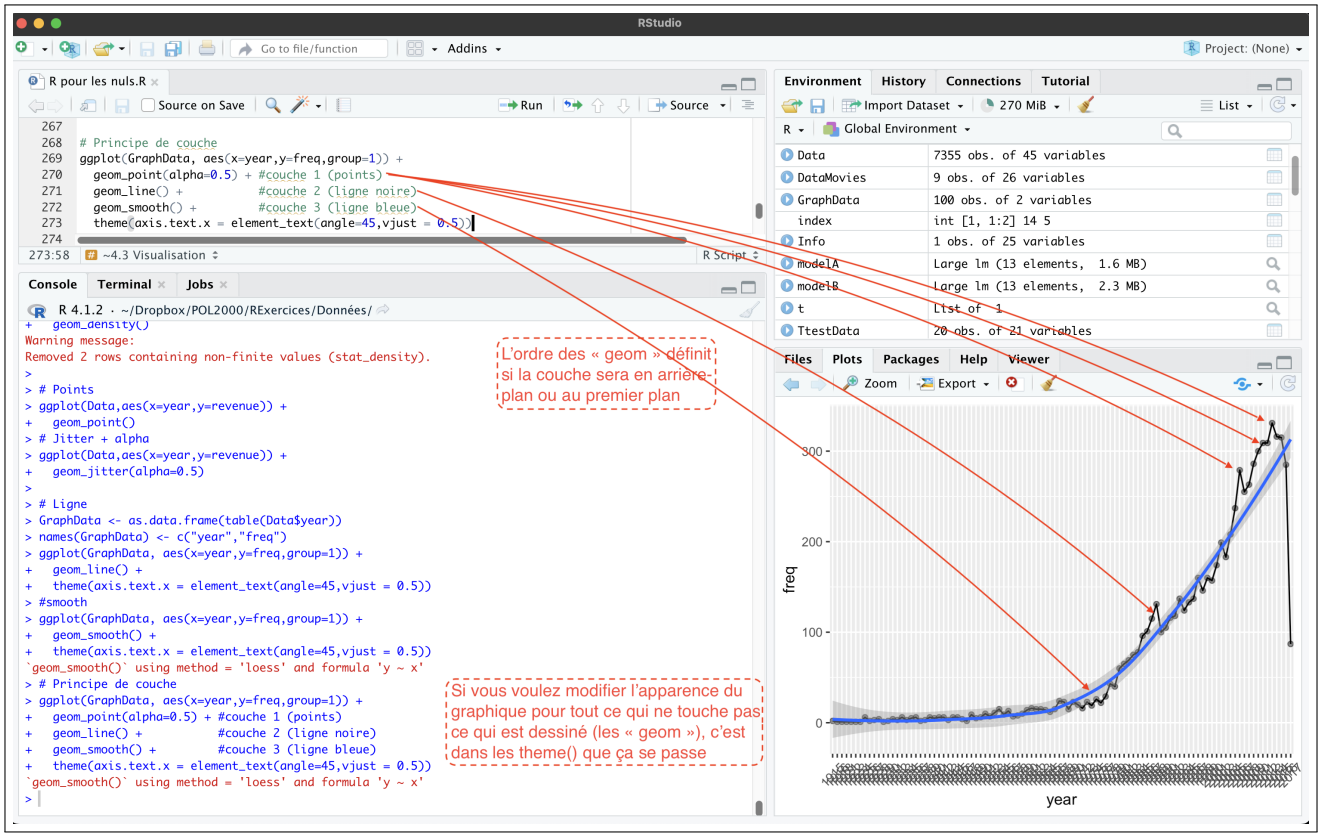


Figure 23: Illustration du principe de couche

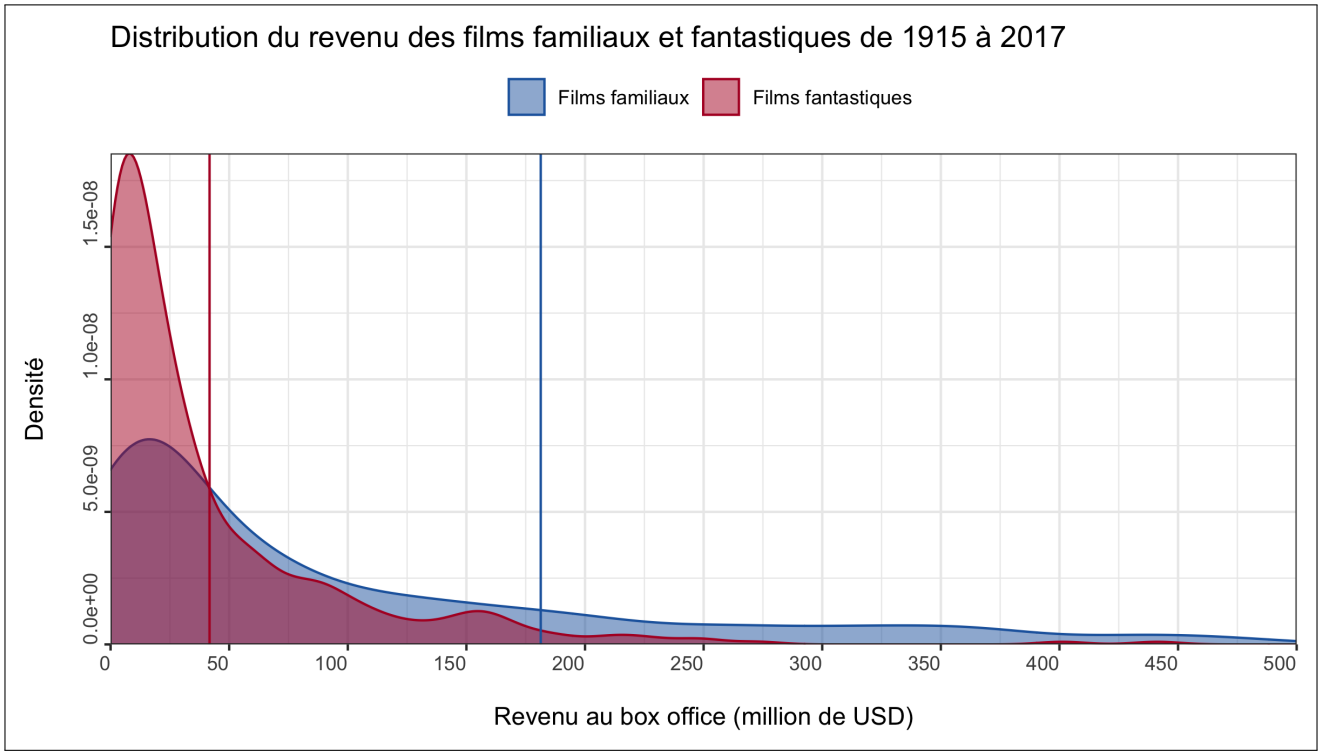


Figure 24: Exemple d'un graphique plus avancé

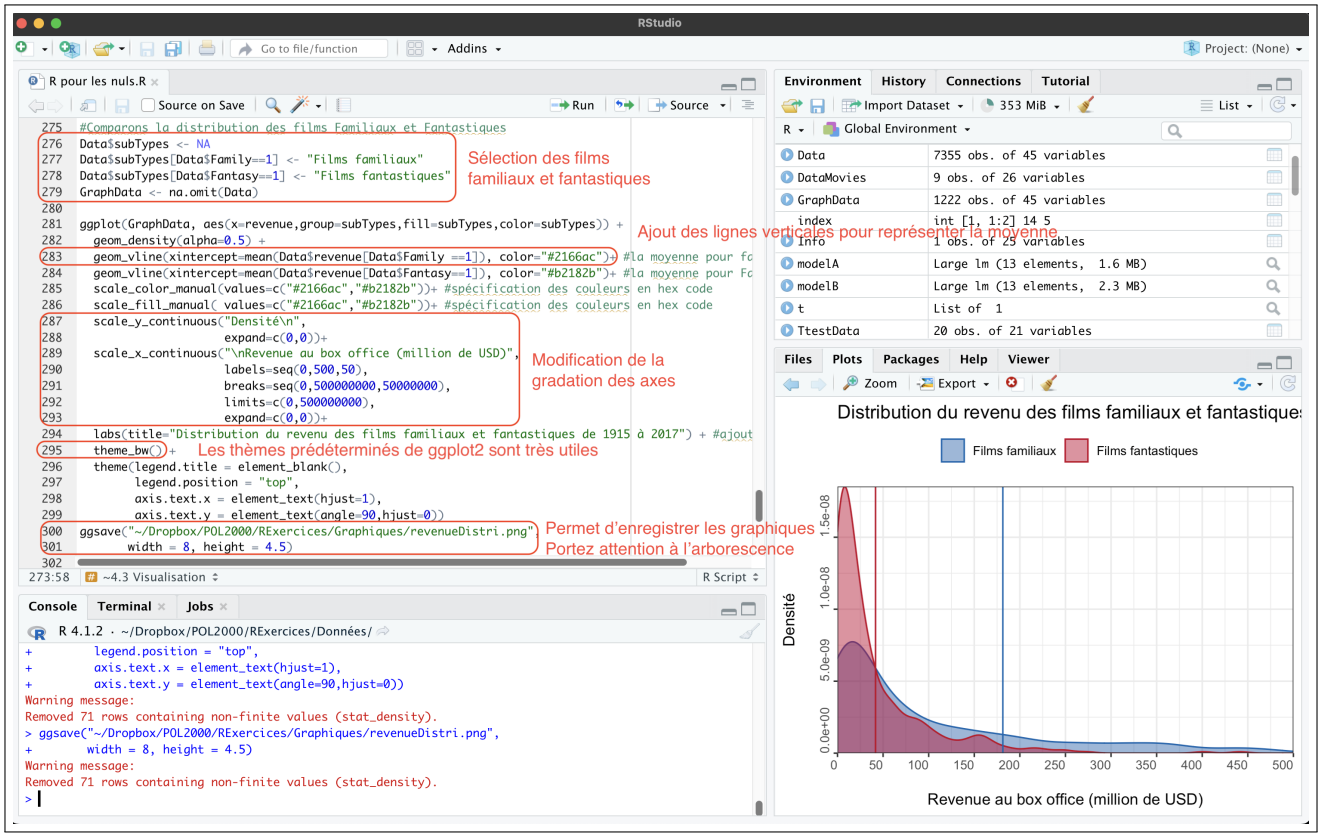


Figure 25: Code d'un graphique plus avancé