Introduction aux sciences sociales numérique

CLESSN

2023-07-15

Table of contents

Avant-propos

Ceci est un exemple de citation Adcock and Collier (2001) .

1 Trois défis pour une contribution aux sciences sociales numériques

Ce premier chapitre n'est sans doute pas le plus excitant. Il ne comprend ni graphique ni exercice. Il s'ancre dans la réflexion théorique plutôt que dans la pratique méthodologique. Habituellement, c'est la partie que l'on ignore, celle que l'on saute pour passer aux « choses sérieuses ». Amateur de « choses sérieuses »? Bonne nouvelle! Cet ouvrage en est rempli. Tout comme la carrière qui s'offre à vous si vous choisissez de poursuivre dans l'étude des sciences sociales numériques.

En 2020, le monde est numérique, et rien ne semble présager un inversement de la tendance. Au contraire, celle-ci risque plutôt de s'accélérer. La pandémie de la COVID-19 a offert quelques-uns des meilleurs exemples de cette tendance: télétravail généralisé, école numérique, livraison en ligne, mobilisation via les réseaux sociaux, intelligences artificielles pour le dépistage de fausses nouvelles et application mobile pour tracer les déplacements et freiner les pandémies. L'avenir est au numérique. Pour les jeunes chercheurs en sciences sociales, cela équivaut à une montagne de «choses sérieuses».

Dans ce contexte, il ne fait aucun doute que votre carrière sera passionnante. Si vous n'en êtes pas déjà convaincu, ce livre vous fournira une panoplie d'exemples de vos nombreuses possibilités. De l'analyse textuelle dans les médias aux sondages en ligne de milliers d'individus, en passant par l'extraction de données massives des sites web ou à l'analyse de larges réseaux de communication, vous trouverez assurément des défis à la hauteur de vos aspirations.

Devant ce déluge de données numériques, le jeune chercheur peut avoir l'impression qu'il est possible, voire permis de tout faire. Entendons-nous bien: c'est presque le cas. Tous les jours, vous aurez des idées de projets plus invraisemblables les unes que les autres. Avec vos nouveaux outils, plusieurs de ces idées n'auront aucun problème à se réaliser. Le véritable problème surviendra peut-être le jour où sera négligée la réflexion théorique. La réflexion au cœur même de ce chapitre. Rappelez-vous: c'est ce chapitre que vous avez considéré sauter, au départ!

En fait, il serait surprenant que vous ne soyez pas happés, très tôt dans vos études, à des limites fondamentales à votre travail. Dans cet ouvrage, nous les appelleront « défis ». Nous ne parlons pas ici de données manquantes ou d'accès restreints à l'information. Il s'agit de défis beaucoup plus élémentaires. Ils se comptent au nombre de trois et sont à la base de toute réflexion préalable à la recherche en science sociales numériques. Ils sont:

1. Le défi technique;
2. Le défi théorique;
3. Le défi éthique.

Sachez une chose: ces défis sont présents dans toutes les grandes branches de la science, c'est-à-dire lors de la recherche, lors de la diffusion des résultats et lors de l'enseignement. Que vous comptiez opérer dans l'une, dans l'autre ou dans toutes ces branches, une bonne compréhension des trois défis permettra de limiter les risques d'impair, mais surtout d'élargir l'univers de vos possibles.

1.1 Défi #1: l'inévitable défi technique.

Le premier défi est technique, lié à l'extraction et à l'analyse des données numériques. Il nécessite l'apprentissage et le développement des méthodologies. Avec R dans sa poche, ce défi est hautement simplifié. R permet de penser autrement les possibilités de recherche, et de travailler avec des outils tels que *Shiny*, pour la création instantanée d'applications web interactives ou *Mechanical Turk*, pour la mise en ligne de micro-tâches (*crowdsourcing*) à réaliser à faible coût par des volontaires. R facilite également la réalisation de revues de la portée de la littérature (*scoping review*), une technique permettant de cartographier la littérature scientifique dans un champ donné.

Les données massives nous entourent. Que ce soit au travers de sondages, via les médias sociaux ou à l'intérieur des archives gouvernementales en ligne, il est plus facile que jamais de rassembler de grandes quantités d'information. Le défi demeure toutefois complexe lorsque vient le temps d'extraire et d'analyser ces données afin de contribuer à la connaissance scientifique.

Déjà, cet ouvrage offre une base solide sur laquelle développer vos méthodes. Celles-ci sont de plus en plus simples à apprendre et à appliquer, notamment grâce aux réseaux de collaboration en ligne. Aujourd'hui, une question peut rapidement être répondue après une recherche sur Google. *Stack Overflow* est un site web dédié à l'entraide entre programmeurs. Vous le trouverez hautement utile.

Si les méthodologies sont plus efficaces que jamais, beaucoup restent encore à faire pour permettre la transparence et l'accessibilité des données publiques, la collaboration entre chercheurs et l'optimisation des outils d'extractions de données. Le cœur du défi technique réside dans l'amélioration des outils utiles et nécessaires aux chercheurs.

En effet, après l'apprentissage des méthodes disponibles à l'heure actuelle, vous pourrez rapidement contribuer à leur optimisation. La beauté d'un logiciel libre comme R est qu'il est possible pour tous de développer de nouvelle manière de facilité la recherche, et d'ensuite partager ces trouvailles avec le monde entier. Sur R, vous pourrez construire des fonctions qui accéléreront votre travail. Le développement de quatre ou cinq fonctions pourrait ensuite faire l'objet d'un tout nouveau «package», que vous partagerez en ligne.

Tous les jours, de nouveaux packages R sont développés et mis en ligne. Des dizaines existent simplement pour réaliser de l'analyse textuelle automatisée, par exemple, une méthodologie qui permet l'étude quantitative de large corpus de textes. Plusieurs de ces packages, comme «Quanteda», «Topicmodels», ou ceux de la «Tidyverse» sont hautement performants, et en constante amélioration.

Il est à la portée de toute chercheuse et de tout chercheur de participer à la bonification des outils et à l'avancement des méthodologies. C'est la réponse attendu au défi technique.

1.2 Défi #2: le nécessaire défi théorique.

- Nécessite une formation selon les principaux travaux scientifiques qui étudient l'impact des données numériques sur les théories en sciences sociales:
  - On ne doit pas réinventer la roue à chaque article scientifique;
  - Comment intégrer nos travaux à la littérature actuelle?;
  - Comment faire progresser cette littérature? Démontrer l'impact des données numériques sur les théories existantes;
  - Exemple: le nationalisme: peut-on mesurer le nationalisme au travers des médias sociaux? Si oui, comment cela peut-il contribuer à la littérature sur le nationalisme?

1.3 Défi #3: l'épineux défi éthique.

- Autour des questions de l'effet de l'ère numérique sur la confidentialité, la sécurité informatique, le consentement et le droit des sujets secondaires:
  - Le numérique offre beaucoup d'opportunité tout à fait légale, mais pas nécessairement éthique;
  - Nécessaire d'encourager la réflexion par rapport aux défis humains entourant l'utilisation des nouvelles données numériques;
  - Comment utiliser ces données pour améliorer les vies, sans brimer les libertés individuelles?;
  - Exemple: intelligence artificielle (machine learning): Le milieu académique est loin d'être seul à s'intéresser à la grande quantité d'information disponible. Les partis politiques, les agences de marketing et bien d'autres organisations utilisent ces informations à des fins de victoires, ou de ventes.

1.4 Conclusion du chapitre

- Au travers des nouveaux apprentissages et des exemples qui sont offerts dans ce livre, le lecteur est encouragé à se poser ces 3 questions:

1. D'abord, comment puis-je utiliser ces nouveaux outils pour faire progresser les méthodologies de recherche actuelles?
2. Ensuite, comment puis-je utiliser ces nouveaux outils pour contribuer à l'avancement des théories de mes champs de recherche?
3. Enfin, comment puis-je utiliser ces nouveaux outils pour exercer un impact positif sur mes semblables?

2

3

4 Le monde du libre

« Vous n'avez pas à suivre une recette avec précision. Vous pouvez laisser de côté certains ingrédients. Ajouter quelques champignons parce que vous en raffolez. Mettre moins de sel car votre médecin vous le conseille — peu importe. De surcroît, logiciels et recettes sont faciles à partager. En donnant une recette à un invité, un cuisinier n'y perd que du temps et le coût du papier sur lequel il l'inscrit. Partager un logiciel nécessite encore moins, habituellement quelques clics de souris et un minimum d'électricité. Dans tous les cas, la personne qui donne l'information y gagne deux choses : davantage d'amitié et la possibilité de récupérer en retour d'autres recettes intéressantes. » - Richard Stallman

Cette analogie illustre bien trois concepts au coeur de la philosophie de Richard Stallman, souvent considéré comme le père fondateur du logiciel libre: liberté, égalité, fraternité. Les utilisateurs de ces logiciels sont libres, égaux, et doivent s'encourager mutuellement à contribuer à la communauté. Ainsi, un logiciel libre est généralement le fruit d'une collaboration entre développeurs qui peuvent provenir des quatre coins du globe. Une réflexion éthique est au coeur du mouvement du logiciel libre, dont les militants font campagne pour la liberté des utilisateurs dès le début des années 1980. La Free Software Foundation (FSF), fondée par Richard Stallman en 1985, définit rapidement le logiciel «libre» [free] comme garant de quatre libertés fondamentales de l'utilisateur: la liberté d'utiliser le logiciel sans restrictions, la liberté de le copier, la liberté de l'étudier, puis la liberté de le modifier pour l'adapter à ses besoins puis le redistribuer [1] Il s'agit ainsi d'un logiciel dont le code source[2] est disponible, afin de permettre aux internautes de l'utiliser tel quel ou de le modifier à leur guise. Puisque le langage machine est difficilement lisible par l'homme et rend la compréhension du logiciel extrêmement complexe, l'accès au code source devient essentiel afin de permettre à l'utilisateur de savoir ce que le fait programme fait réellement. Seulement de cette façon, l'utilisateur peut *contrôler* le logiciel, plutôt que de se faire contrôler par ce dernier (Stallman, 1986).

[1] La redistribution doit évidemment respecter certaines conditions précises, dont l'enfreint peut mener à des condamnations [http://www.softwarefreedom.org/resources/2008/shareware.html].

[2] Pour rester dans les analogies culinaires, le code source est au logiciel est ce que la recette est à un plat: elle indique les actions à effectuer, une par une, pour arriver à un résultat précis. Encore une fois, cette dernière peut-être adaptée, modifiée, bonifiée.

4.1 Émergence et ascension

Plusieurs situent les débuts du mouvement du logiciel libre avec la création de la licence publique générale GNU[3], en 1983, à partir de laquelle va se développer une multitude de programmes libres. Depuis, la popularité des logiciels libres n'a cessé de croître, alors que des dizaines de millions d'usagers à travers le monde utilisent désormais ces logiciels. Parmi les plus populaires, on retrouve notamment le navigateur Firefox, la suite bureautique OpenOffice et l'emblématique système d'exploitation Linux, qui se développe d'ailleurs à partir de la licence GNU. Les logiciels libres ont différents usages (en passant par la conception Web, la gestion de contenu, les sytèmes d'exploitation, la bureautique…). Encore une fois, le logiciel libre est avant-tout une philosophie, voire un mouvement de société. C'est une façon de concevoir la communauté du logiciel, où le respect de la liberté de l'utilisateur est un impératif éthique central (reformuler?) (Williams et al., 2020:26). Si ce mouvement fut d'abord initié par quelques militants dans les années 1980, c'est aujourd'hui un véritable phénomène sociétal: des milliers d'entreprises, d'organisation à but non lucratif, d'institutions ou encore de particuliers adoptent tour à tour ces logiciels, dont la culture globale et les valeurs (entraide, collaboration, partage) s'arriment avec le virage technologique de plusieurs entreprises à l'ère du numérique (retravailler, mais l'idée est là). [blablabla]

Il faut garder en tête que logiciel libre ne rime pas nécessairement avec gratuité. Bien que plusieurs logiciels libres soient téléchargeables gratuitement (donner des exemples), il est aussi possible de (re)distribuer des logiciels libres payants (reformuler, pas clair). Par ailleurs, aucun logiciel libre n'est réellement «gratuit» dans la mesure où son déploiment et son utilisation nécessitent généralement différents coûts, dont les degrés sont variables en fonction des compétences et de l'infrastructure dont disposent les utilisateurs (coût d'apprentissage, coûts d'entretien, etc.). Enfin, il est important de garder en tête les logiciels libres possèdeux eux-aussi une licence - cette dernière est d'ailleurs garante des libertés que confèrent les logiciels libres aux utilisateurs.

4.1.1 Logiciel libre et *open source*

"Les deux expressions décrivent à peu près la même catégorie de logiciel, mais elles représentent des points de vue basés sur des valeurs fondamentalement différentes. L'open source est une méthodologie de développement ; le logiciel libre est un mouvement de société."

[3] expliquer ce qu'est GNU en quelques lignes/le modèle collaboratif de développement logiciel initié par le projet GNU

4.2 Principaux avantages et inconvénients

La disponibilité du code source et le mode de développement collaboratif du logiciel libre facilitent également le transfert des connaissances et ce, au-delà des frontières. Où qu'ils soient, les institutions, les entreprises et les particuliers peuvent utiliser ces logiciels et les adapter en fonction de leurs besoins respectifs. Par ailleurs, l'accès libre et égal de tous les internautes à l'ensemble de ces connaissances constitue un enjeu majeur pour la vitalité démocratique des sociétés à l'ère du numérique, caractérisées par une surabondace d'information.

Les logiciels libres, parce qu'ils sont souvent moins coûteux (voire téléchargeables gratuitement) et qu'ils démocratisent l'accès à l'information, contribuent à réduire les disparités en termes d'accessibilité aux nouvelles technologies.

Stallman - Lui-même issu du monde de la recherche scientifique. L'esprit même du logiciel libre est très proche ; contribution à la culture globale de partage, d'entraide, etc. que l'on peut retrouver dans le domaine scientifique

5 R ou ne pas R?

William Poirier

À ce point du livre, vous avez été introduit aux problèmes et aux opportunités qu'amène l'ère numérique. Vous avez d'ailleurs sans doute déjà une idée de comment ou d'à propos de quel sujet vous pourriez exploiter ce nouveau monde de possibilité. Les prochaines sections du livre auront ainsi pour but de vous introduire aux outils qui permettront la réalisation de vos projets, en commençant par l'outil d'analyse de base – le langage R.

5.1 Pourquoi R?

R est un langage de programmation *OpenSource* développé par des statisticiens pour des statisticiens dans les années 1990 (Tippmann 2015). C'est d'un élan d'amour propre et du désire d'honorer le langage de programmation S que Ross Ihaka et Robert Gentleman nommeront leur création, infirmant ainsi la légende selon laquelle les scientifiques seraient mauvais pour nommer les choses. Ces derniers feront des choix non orthodoxes lors de l'élaboration du langage, des choix qui font aujourd'hui la popularité de R auprès d'un large pan de la communauté académique. En effet, Morandat et al. (2012) rapportent que le langage a été élaboré afin qu'il soit intuitif et qu'il permette aux nouveaux utilisateurs de rapidement réaliser des analyses. Ils rapportent même que dans plusieurs départements de statistiques, R est introduit en 2 semaines – environ le temps que prend l'individu moyen pour oublier ses résolutions du Nouvel An.

Toutefois, avant de débuter l'apprentissage d'un nouvel outil, il faut être convaincu de sa pertinence, de son utilité. À quoi bon apprendre à utiliser une perceuse alors que mon tournevis fonctionne parfaitement bien ? C'est pourquoi ce chapitre a deux objectifs, d'abord, il s'agira de vous convaincre de la pertinence de R suite à quoi il vous sera introduit diverses utilisations possibles de R. Plus spécifiquement, la section de réflexion théorique exposera les avantages et les inconvénients de R et le comparera à ses principaux compétiteurs. Ensuite, la réflexion méthodologique présentera brièvement la programmation de base en R et en quoi l'OpenSource fait de R un outil si puissant. Le chapitre se conclura avec quelques trucs et astuces qui vous permettront de surmonter l'anxiété que peut causer l'apprentissage d'un outil étant, pour plusieurs, quelque chose de véritablement étranger à leur relation typique avec les ordinateurs.

5.2 Réflexion théorique

R a deux types de compétiteurs lorsqu'il est question d'analyses statistiques – les logiciels à licences comme SAS,STATA et SPSS, et les langages *OpenSource*, principalement Python et sa libraire Pandas. Le chapitre précédant ayant déjà élaborer un cas exhaustif en faveur du logiciel libre, il ne sera ici que rappelé les grandes lignes de l'argument, à savoir que : 1) l'*OpenSource* est gratuit d'utilisation; 2) l'*OpenSource* est développé de façon bottom-up, ce qui lui procure une grande flexibilité; et 3) il permet aux utilisateurs de créer leurs propres fonctions. À l'inverse, les logiciels à licences sont coûteux, rigides et l'ajout de fonctionnalités se fait par les développeurs internes à la compagnie ce qui rend le processus plus lent et réduit l'éventail des possibilités. Ceci étant dit, certains avanceront que le c'est justement ce processus interne lent qui assure la validité et la fiabilité des analyses effectuées par SAS, STATA ou SPSS. Or, dans son livre dédié aux utilisateurs de SPSS et de SAS, Muenchen (2011) soulève le point que bien souvent, ce sont des individus atomisés qui développent les nouvelles fonctionnalités de ces langages et que le processus de révisions se fait ensuite par des comités internes de testeurs. Il en va de même pour le développement des *Packages* R dans la mesure où ce dernier se vois tester et amender par plusieurs programmeurs indépendants dans un processus itératif sur GitHub ou sur d'autres plateformes similaires. De plus, bien des nouvelles techniques statistiques sont développées pour R par des professeurs qui publie d'abord leur travail dans des journaux académiques revus par des pairs. Bien entendu, rien n'empêche un étudiant gradué de publier ses propres *packages*. C'est pourquoi Muenchen (2011) recommande de visiter le site *MACHIN* afin d'avoir une idée de la validité et de la fiabilité du *package* en question. Enfin, le fait que SAS et SPSS permettent à leur utilisateur d'intégrer des routines R à leur programme est un indicateur fort ne serait-ce que de l'utilité de R (Muenchen 2011).

R n'est cependant pas qu'un outil statistique, il s'agit également d'un outil de programmation puissant. Ceci fait en sorte que le coût d'entrer de R est plus important que celui des logiciels comme SAS, STATA et SPSS puisqu'il impose l'apprentissage d'une syntaxe et d'un jargon particulier. Alors, pourquoi apprendre à programmer en plus d'apprendre à réaliser des analyses statistiques? Après tout, faire des statistique c'est déjà beaucoup! D'abord, apprendre à programmer permet de développer la résolution de problème et la logique, deux compétences aux cœur de la recherche scientifique. Programmer est au cerveau ce que courir est au coeur, il s'agit d'un exercice difficile au départ mais dont les résultats bénéfique se font sentir rapidement. Apprendre à programmer permettra également de mieux comprendre la façon dont son propre ordinateur fonctionne. Contrairement au mythe urbain qui veux que l'humain n'utilise que 10% de son cerveau, la plupart des individus ne font qu'utiliser une infime partie du potentiel de leur ordinateur. Par exemple, l'ordinateur ayant permis aux américains d'aller sur la lune lors de la mission Apollo-11, le *Apollo Guidance Computer (AGC)*, avait 4 096 octets (bytes) de mémoire RAM[1] [*SOURCE FIABLE*]. L'ordinateur sur lequel sont écrits ces lignes

[1]La RAM (*Random Access Memory*) ou "mémoire vive" est l'espace utilisée par l'ordinateur pour enregistrer l'information directement nécessaire pour les opérations en cours d'exécution. Elle s'oppose à la ROM (*Read-Only Memory*) ou "mémoire morte" qui contient toute l'information enregistré de façon permanante dans

a 8GB de RAM, soit approximativement 2 millions de fois plus que celle de l'AGC. Enfin, l'argument le plus probant sur la nécessité d'apprendre à programmer est celui du marché de l'emplois. Au Canada, il est prévu une pénurie de main d'oeuvre pour les emplois requérant de aptitude en statistique et en programmation comme celui de scientifique de données ou d'analyste (Employment and Social Development Canada 2023). Un rapport de PWC indique même que les employeurs devrons s'attendre à ce battre pour engager des individus compétent dans les deux domaines. Apprendre à programmer devient alors, non seulement, une façon d'améliorer votre résonnement scientifique, mais également une façon de vous démarquer sur le marché de l'emplois.

- Avantages :
  - Gratis!!
  - Plus facile d'accès que d'autres langages de programmation
  - Arguments technos :
    * Fait pour les stats
    * Fait tout ce que SPSS et Stata font sans le carcan du logiciel privé
  - Arguments d'autorités <- popularité du langage
  - Ouvre vers un monde de possibilités
  - Développement d'une expertise recherchée
- Inconvénients :
  - Courbe d'apprentissage rude pour certains
  - Développement anarchique
  - Risque de se perdre dans les profondeurs pleines de microbes de CRAN
- Propriété et utilisation de R
- Comparaison avec d'autres langages
  - Python, SPSS, Stata?

5.3 Réflexion méthodologique

- Base R
  - Stable mais parfois bof
  - Manipulation de données
  - Fonctions et Boucles
- La puissance de l'OpenSource
  - Peut être instable, mais souvent plus intéressant que les options en Base R

l'ordinateur.

  - TidyVerse
    * Manipulation avec Dplyr
    * All hail Hadley!
  - Analyse textuelle avec ???
  - Shiny

5.4 Trucs et astuces

- 10 choses à garder en tête lorsque l'on apprend R :

  1. Vous n'allez pas briser votre ordinateur.
  2. C'est en "gossant" que l'on apprend! Essayez des trucs, expérimentez – souvenez-vous de 1.
  3. Contrairement à la vraie vie, il y a toujours le crtl-z pour vous sauver.
  4. Ayez de l'empathie pour vos futurs lecteurs, ou du moins pour le futur vous – COMMENTEZ VOTRE CODE!
  5. Même Wozniak était mauvais au début.
  6. Pensez à votre santé – levez-vous de votre chaise aux 30 minutes.
  7. S'il y a un bogue, et il y en aura, *Google* est votre ami.
  8. Souvent c'est une question de type de variable – caractère, numérique, facteurs, etc.
  9. Si vous faites beaucoup de copier-coller de code, il y a surement une façon de l'automatiser.
  10. Sérieusement, faites le 4!

6 Les environnements de développement intégré

6.1 Où coder en R ?

Un environnement de développement intégré (IDE), permet aux programmeurs de consolider les différents aspects de l'écriture d'un programme informatique. Ils permettent de réaliser toutes les activités courantes d'un programmeur – l'édition du code, la construction des exécutables et le débogage – au même endroit. Les environnements de développement intégrés sont conçus pour maximiser la productivité du programmeur. Ils fournissent de nombreuses fonctionnalités – notamment la coloration syntaxique et le contrôle de version – pour créer, modifier et compiler du code.

Certains environnements de développement intégré sont dédiés à un langage de programmation spécifique. Par conséquent, ils contiennent des fonctionnalités qui sont plus compatibles avec les paradigmes de programmation du langage auquel is sont associés. Cependant, il existe de nombreux environnements de développement intégré multilingues.

R est un des langages de statistiques et d'exploration de données les plus populaires en sciences sociales et il est open-source. Par conséquent, il est logique de choisir un environnement de programmation open-source. R est pris en charge par de nombreux environnements de programmation. Plusieurs ont été spécialement conçus pour la programmation en R – le plus notable étant RStudio – tandis que d'autres sont des environnement de programmation universels – tel que Visual Studio – et prennent en charge R via des plugins. Il est également possible de coder en R à partir d'une interface en ligne de commande. Une telle méthode permet la communication entre l'utilisateur et son ordinateur. Cette communication s'effectue en mode texte : l'utilisateur tape une « ligne de commande » – c'est-à-dire du texte dans le *terminal* – pour demander son ordinateur d'effectuer une opération précise, par exemple rouler un fichier de code R.

Le présent chapitre présente RStudio, ses avantages et inconvénients ainsi que des exemples de ses fonctionnalités de RStudio et des conseils sur comment l'utiliser et le personnaliser.

6.2 Pourquoi RStudio ?

6.2.1 Qu'est-ce que RStudio ?

«««< HEAD ## Pourquoi RStudio ?

6.2.2 Qu'est-ce que RStudio ?

Comme plusieurs autres langages de programmation, R est développé grâce à des fonctions écrites par ses usagers. Un IDE, comme RStudio, est conçu pour faciliter ce travail (Verzani, 2011). RStudio est un projet open source destiné à combiner les différentes composantes du langage de programmation R en un seul outil (Allaire, 2011). Il est conçu pour faciliter la courbe d'apprentissage des nouveaux utilisateurs. RStudio fonctionne sur toutes les systèmes d'exploitation, y compris Windows, Mac OS et Linux. En plus de l'application de bureau, RStudio peut être déployé en tant que serveur pour permettre l'accès Web aux sessions R s'exécutant sur des systèmes distants (Allaire, 2011).

Figure of RStudio with some code, a plot in the bottom right corner and some data in the top right corner

RStudio facilite l'utilisation du langage de programmation R en offrant de nombreux outils permettant à son utlisateur d'aisément réaliser ses tâches. Parmi les plus utiles, on retrouve notamment une fenêtre d'aide, de la documentation sur les différents packages R, un navigateur d'espace de travail, une visionneuse de données et une prise en charge de la coloration syntaxique (Horton, Kleinman, 2015). De plus, RStudio permet de coder dans plusieurs langages et supportent un grande quantité de formats. Il fournit également un support pour plusieurs projets ainsi qu'une interface pour utiliser des systèmes de contrôle des versions tels que GitHub (Horton, Kleinman, 2015).

6.2.3 Avantages et inconvénients de RStudio

RStudio a plusieurs avantages. L'utilisation de l'IDE est facile à apprendre pour les débutants. Les principaux éléments d'un IDE sont intégrés dans une disposition à quatre volets (Verzani, 2011). Cette disposition comprend une console, un éditeur de code source à onglets pour organiser les fichiers d'un projet, un espace pour l'environnement de travail et un quatrième volet où il est possible d'afficher des graphiques ou de la documentation sur différents packages. De plus, on y retrouve la possibilité de créer plusieurs espaces de travail – appelés projets – qui facilitent l'organisation de différents workflows.

Un autre aspect de RStudio que de nombreux programmeurs apprécient est le fait qu'il peut être utilisé via un navigateur Web pour un accès à distance (Verzani, 2011). De plus, l'IDE offre de nombreux outils pratiques et faciles à utiliser pour gérer les packages, l'espace de travail

et les fichiers. RStudio supporte plusieurs langages de programmation ainsi que différents langages de balisage. Finalement, de nouvelles fonctionnalités sont souvent ajoutées pour satisfaire aux besoins de la communauté scientifique et le logiciel est régulièrement mis à jour.

Inconvénients : - Peu de configuration, tu peux pas changer les raccourcis, etc - Très limité dans le setup des différents panneaux, tu peux pas voir 2 fichiers en même temps Tu peux pas visualiser 2 fichiers 1 à côté de l'autre, une feature de base dans n'importe quel ide Tu peux pas configurer tes raccourcis clavier, aussi une feature de base Mettons que je veux que ctrl + D copie la ligne, comme dans visual studio, je peux pas - Plus lent que d'autres alternatives pour certaines opérations

======= Comme plusieurs autres langages de programmation, R est développé grâce à des fonctions écrites par ses usagers. Un IDE, comme RStudio, est conçu pour faciliter ce travail (Verzani, 2011). RStudio est un projet open source destiné à combiner les différentes composantes du langage de programmation R en un seul outil (Allaire, 2011). Il est conçu pour faciliter la courbe d'apprentissage des nouveaux utilisateurs. RStudio fonctionne sur toutes les systèmes d'exploitation, y compris Windows, Mac OS et Linux. En plus de l'application de bureau, RStudio peut être déployé en tant que serveur pour permettre l'accès Web aux sessions R s'exécutant sur des systèmes distants (Allaire, 2011).

Figure of RStudio with some code, a plot in the bottom right corner and some data in the top right corner

RStudio facilite l'utilisation du langage de programmation R en offrant de nombreux outils permettant à son utlisateur d'aisément réaliser ses tâches. Parmi les plus utiles, on retrouve notamment une fenêtre d'aide, de la documentation sur les différents packages R, un navigateur d'espace de travail, une visionneuse de données et une prise en charge de la coloration syntaxique (Horton, Kleinman, 2015). De plus, RStudio permet de coder dans plusieurs langages et supportent un grande quantité de formats. Il fournit également un support pour plusieurs projets ainsi qu'une interface pour utiliser des systèmes de contrôle des versions tels que GitHub (Horton, Kleinman, 2015). »»»> 0767363f696a8e4a9fac38bd0de1c9327b812608

6.2.4 Avantages et inconvénients de RStudio

«««< HEAD - Exemples de fonctionnalités + Files, Plots, Packages, Help, and Viewer Pane Layout of the Components The RStudio interface consists of several main components sitting below a top-level toolbar and menu bar. Although this placement can be customized, the default layout utilizes four main panes in the following positions:

In the upper left is a Source browser pane for editing files (see Source Code Editor) or viewing some data sets. In Figure 1-3 this is not visible, as that session had no files open.

In the lower left is a Console for interacting with an R process (see Chapter 3).

In the upper right are tabs for a Workspace browser (see the section Workspace Browser) and a History browser (see the section Command History).

In the lower right are tabbed panes for interacting with the Files (The File Browser), Plots (Graphics in RStudio), Packages (Package Maintenance), and Help system components (The Help Page Viewer). If the facilities are present, an additional tab for version control (Version Control with RStudio) is presented.

The Console pane is somewhat privileged: it is always visible, and it has a title bar. For the other components, their tab serves as a title bar. These panes have page-specific toolbars (perhaps more than one)—which in the case of the Source pane are also context-specific.

The user may change the default dimensions for each of the panes, as follows. There is an adjustable divider appearing in the middle of the interface between the left and right sides that allows the user to adjust the horizontal allocation of space. Furthermore, each side then has another divider to adjust the vertical space between its two panes. As well, the title bar of each pane has icons to shade a component, maximize a component vertically, or share the space (Verzani, 2011). **Also check Nierhoff et Hillebrand 2015**

- Comment personnaliser / se familiariser avec RStudio
  - Changement de couleurs pour le fond
  -

7 Section aide

> RStudio a plusieurs avantages. L'utilisation de l'IDE est facile à apprendre pour les débutants. Les principaux éléments d'un IDE sont intégrés dans une disposition à quatre volets (Verzani, 2011). Cette disposition comprend une console, un éditeur de code source à onglets pour organiser les fichiers d'un projet, un espace pour l'environnement de travail et un quatrième volet où il est possible d'afficher des graphiques ou de la documentation sur différents packages. De plus, on y retrouve la possibilité de créer plusieurs espaces de travail – appelés projets – qui facilitent l'organisation de différents workflows.

Un autre aspect de RStudio que de nombreux programmeurs apprécient est le fait qu'il peut être utilisé via un navigateur Web pour un accès à distance (Verzani, 2011). De plus, l'IDE offre de nombreux outils pratiques et faciles à utiliser pour gérer les packages, l'espace de travail et les fichiers. RStudio supporte plusieurs langages de programmation ainsi que différents langages de balisage[1]. Finalement, de nouvelles fonctionnalités sont souvent ajoutées pour satisfaire aux besoins de la communauté scientifique et le logiciel est régulièrement mis à jour.

L'IDE comporte toutefois certains inconvénients. En effet, il n'est pas possible de configurer des raccourcis claviers personnalisés, une possibilité que plusieurs autres IDE offrent. De plus, les possibilités de disposition des différents panneaux sont très limitées et la malléabilité de l'espace de travail est restreinte. Finalement, pour certaines opérations, RStudio peut être plus lent que d'autres alternatives.

7.1 Comment utiliser RStudio ?

La première étape pour commencer à utiliser RStudio est de l'installer[2]. Une fois que cela est fait, ouvrez RStudio. La fenêtre qui apparait devrait ressembler à l'image ci-dessus. La couleur de l'arrière-plan et celle de la police, la taille des cadrans ainsi que de nombreux autres éléments peuvent être changés. La dernière section de ce chapitre aidera le lecteur à personnaliser son IDE.

Image de RStudio sans rien, settings de base.

[1] À cet effet, voir le chapitre 6 du présent ouvrage.
[2] À cet effet, voir le chapitre 9 du présent ouvrage.

Bien que de nombreux éléments puissent être personnalisés, la disposition par défaut de RStudio est composée de quatre volets principaux (Verzani, 2011). Dans le coin supérieur gauche se trouve le quadran principal. C'est dans celui-ci que l'utilisateur passera la plus grande partie de son temps. On y modifie des fichiers de différents formats et il est possible d'y afficher des bases de données. Dans le coin inférieur gauche se trouve la console et le terminal. Dans cette première, on peut interagir avec R de la même manière que dans le cadran principal, mais le code ne sera pas enregistré. Le terminal, pour sa part, est le point d'accès de communication entre un usager et son ordinateur. Bien que les différents systèmes d'exploitation viennent avec un terminal déjà intégré, il est aussi possible d'y accéder a partir de RStudio. »»»> 0767363f696a8e4a9fac38bd0de1c9327b812608

Image de RStudio qui montre les quatre cadrans. Idéalement avec un projet en cours et les différents cadrans utilisés. Settings personalisés?.

«««< HEAD One can easily switch between components using the mouse. As well, the View menu has subitems for this task. For power users, the keyboard shortcuts listed in Table 1-2 are useful. (A full list of keyboard shortcuts is available through the Help > Keyboard Shortcuts menu item.) ======= On retrouve dans le coin supérieur droit l'espace de travail. Ce cadran contient trois éléments : l'*environnement global, l'historique et les connections*. L'*environnement global* est l'endroit où l'utilisateur peut voir les bases de données, les fonctions et les différents autres objets R qui sont actifs. Il peut cliquer sur les divers éléments actifs pour les consulter. L'onglet *historique* permet à l'utilisateur de consulter les derniers morceaux de code R qu'il a roulé ainsi que les dernières commandes écrites dans la console. L'onglet *connections*, pour sa part, permet de connecter son IDE à une variété de sources de données et d'explorer les objets et les données qui la compose. Il est conçu pour fonctionner avec une variété d'autres outils pour travailler avec des bases de données en R dans RStudio.

Le cadran dans le coin inférieur droit, pour sa part, contient plusieurs outils très utiles pour les usagers de RStudio. L'onglet *Files* permet à l'utilisateur de naviguer dans les fichiers que contient son ordinateur sans avoir à sortir de RStudio. L'onglet *Plots* permet de visualiser les graphiques générer à partir de R, que ce soit en utilisant *ggplot2, lattice ou base R*[3]. L'ongliet *Packages* permet de consulter les packages installés précédemment par l'utilisteur en plus de pouvoir en consulter la documentation. C'est aussi un des différents endroits à partir d'où il est possible d'installer des packages avec RStudio. L'onglet *Help* permet à l'utilisateur de chercher et de consulter de la documentation sur de nombreux sujets, notamment sur les différentes fonctions en R ainsi que sur les packages. Pour sa part, l'onglet *Viewer* permet la visualisation de contenu web local.

L'utilisateur peut modifier les dimensions par défaut pour chacun des quatre cadrans principaux. En cliquant sur la division des sections, il est possible d'ajuster l'allocation horizontale de l'espace. De plus, chaque côté dispose d'un autre séparateur pour ajuster l'espace vertical. Qui plus est, la barre de titre de chaque cadran comporte des icônes pour ombrer un

[3]Pour en apprendre davantage sur la visualisation graphique en R, consulter le chapitre 7 du présent ouvrage.

composant, maximiser un cadran verticalement ou modifier la taille des l'espace de travail (Verzani, 2011; Nierhoff et Hillebrand, 2015).

7.2 Personnaliser son RStudio

One can easily switch between components using the mouse. As well, the View menu has subitems for this task. For power users, the keyboard shortcuts listed in Table 1-2 are useful. (A full list of keyboard shortcuts is available through the Help > Keyboard Shortcuts menu item.)

- Comment personnaliser / se familiariser avec RStudio
  - Changement de couleurs pour le fond
  - Section aide »»»> 0767363f696a8e4a9fac38bd0de1c9327b812608

8 Baliser les sciences sociales : langages et pratiques

8.1 Question

Lorsque vous lisez une page Web, un article scientifique ou un curriculum vitæ professionnel, vous vous doutez peut-être que le texte n'est pas toujours produit à l'aide d'un simple logiciel de traitement de texte comme Microsoft Word, Apple Pages ou LibreOffice Writer. La mise en page réglée au millimètre près, la qualité des figures et graphiques, le style des références, la présence d'éléments interactifs et la cohérence hiérarchique du texte sont difficiles à reproduire à l'aide d'un logiciel de traitement de texte régulier, entre autres. L'insertion de tableaux de régression, de figures et d'extraits de code de haute qualité graphique ainsi que leur personnalisation nécessitent une interface particulière.

Pour ces raisons et plusieurs autres, les chercheurs en sciences sociales font souvent appel aux langages de balisage, ou *markup languages*. Ceux-ci permettent de produire des documents et pages Web sans les limitations des logiciels de traitement de texte. Le présent livre, par exemple, est écrit à l'aide du langage de balisage Markdown. D'entrée de jeu, vous vous demandez peut-être quelle est l'utilité d'apprendre ces langages alors que les logiciels de traitement de texte sont nombreux, simples d'approche et en amélioration constante. Ce chapitre tentera donc de répondre aux questions suivantes : « Pourquoi apprendre à utiliser des langages de balisage? Dans quels contextes sont-ils plus utiles que les logiciels de traitement de texte? Comment les utiliser? » L'accent sera mis sur R Markdown, LaTeX, BibTeX et HTML.

Le premier langage de balisage, le Generalized Markup Language (GML), a été inventé en 1969 par les chercheurs Charles F. Goldfarb, Ed Mosher et Ray Lorie pour la compagnie IBM. Goldfarb et ses collègues devaient intégrer trois applications créées avec des langages différents et avec une logique différente pour les besoins d'un bureau de droit. Même après avoir créé un programme qui permettait aux trois applications d'interagir, ces langages demeuraient différents et avaient chacun leur propre fonctionnement. Le développement de GML a permis de résoudre ce problème en standardisant et en structurant le langage : les mêmes commandes étaient utilisées pour accomplir les mêmes tâches dans chaque programme (Goldfarb 1996). GML a été amélioré durant les décennies suivantes et a été suivi par d'autres langages de balisage, dont LaTeX (1984), HTML (1993), XML (1998) et Markdown (2004).

Un langage de balisage constitue un ensemble de commandes qui peuvent être entremêlées à du texte afin de produire une action informatique. Chaque langage contient son ensemble de

commandes cohérentes et complémentaires. De manière plus formelle, ces commandes sont nommées *balises* (*tags* en anglais) et inscrites par le chercheur lui-même au travers du texte. Les balises constituent une manière de communiquer avec le logiciel que vous utilisez dans un langage qu'il peut comprendre, par exemple pour lui indiquer que vous désirez qu'une section du texte soit écrite en caractères gras, en italique, à double interligne ou encore que vous souhaitez positionner une image d'une certaine manière au travers du texte. Cette interaction est rendue possible par la standardisation des langages de balisage : chaque balise correspond à une action précise, peu importe le logiciel utilisé, la langue dans laquelle le texte est rédigé, le type d'ordinateur utilisé, etc. Dans votre document source, les balises sont entremêlées au contenu de votre document, puis au moment de compiler ce dernier, les balises disparaissent, produisent les actions informatisées qu'elles commandent et ne laissent comme document final que son contenu mis en page tel que vous l'avez défini via les balises utilisées.

Plusieurs langages de balisage existent et permettent d'effectuer différentes tâches. Le plus répandu est le langage HTML, qui permet de formater des sites web. Le langage XML, lui aussi très utilisé, permet de structurer de larges volumes de données. LaTeX permet pour sa part de formater du texte et de créer des documents en format PDF. Markdown permet également de créer des documents de format PDF, mais aussi HTML et DOCX. Depuis 2014, le *package* R Markdown permet d'ajouter des extraits de code R à un fichier en langage Markdown. LaTeX et Markdown permettent aussi d'intégrer les références bibliographiques du système de traitement de références BibTeX, créé en 1985.

Les balises constituent une manière de donner manuellement des commandes au logiciel que vous utilisez. Par exemple, si vous utilisez Microsoft Word, vous avez accès à une panoplie de boutons qui vous permettent de formater votre texte. Les balises exercent les mêmes fonctions, mais de manière manuelle. Lorsque vous appuyez sur un bouton dans Word, celui-ci ajoute des balises au travers de votre texte, mais rend celles-ci invisibles dans l'interface que vous utilisez. Cela permet d'avoir un texte élégant et facile à lire, mais comporte aussi plusieurs inconvénients. Le principal inconvénient est que vous êtes condamné à avoir un pouvoir limité sur le formatage de votre texte. En effet, si les boutons à votre disposition ne vous permettent pas de réaliser une opération, celle-ci sera éternellement impossible à réaliser pour vous. A contrario, les langages de balisage permettent un contrôle presque infini sur les opérations que vous souhaitez réaliser. Incidemment, dans la mesure où vous utilisez le langage approprié pour la tâche que vous souhaitez accomplir, vous devriez être capable de donner exactement la commande nécessaire à votre logiciel. Les langages de balisage, bien qu'ils aient un coût d'apprentissage qui peut s'avérer important et qu'ils soient moins élégants qu'un simple document Word, vous offrent une plus grande flexibilité.

Afin d'utiliser un langage de balisage, il est impératif que le logiciel que vous utilisez puisse prendre en compte ce langage. Un logiciel permet rarement d'utiliser n'importe quel langage. Il est aussi impératif de bien utiliser le langage de balisage. En effet, comme pour les langages de programmation, les langages de balisage ne peuvent pas déduire ce que vous souhaitez leur faire comprendre. Si vous souhaitez mettre du texte en gras, vous devez utiliser les bonnes balises. La moindre erreur est fatale, puisqu'une erreur dans la balise que vous utilisez produira

un message d'erreur, le logiciel ne réussissant pas à associer votre balise mal inscrite à une action informatisée. Conséquemment, il est impératif de bien vérifier les balises utilisées afin d'éviter toute erreur qui empêcherait votre document d'être compilé, c'est-à-dire d'être traduit dans son format final.[1] Chaque caractère dans une balise est important et il y a rarement plus d'une seule manière de commander une action. Le positionnement des balises est lui aussi critique : il délimite la portion de texte à laquelle doit être appliquée l'action commandée par la balise.

Il est important de distinguer les langages de balisage des langages de programmation. En effet, ceux-ci sont similaires à certains égards, mais ont des vocations différentes. Les deux s'appuient sur un langage informatisé, mais les langages et leurs objectifs diffèrent. Un langage de programmation définit des processus informatisés alors qu'un langage de balisage permet d'encoder du contenu de manière à ce que celui-ci soit lisible tant pour l'humain que pour son ordinateur.

Dans le contexte de la recherche en sciences sociales, la programmation est généralement utilisée afin de récolter, d'analyser et de présenter visuellement des données. Une fois cartes, tableaux et graphiques produits, ceux-ci peuvent être enregistrés – par exemple en format PDF ou PNG – et inclus au sein d'un document qui sera formaté en utilisant un langage de balisage. De manière simple, le langage de programmation contribue à l'analyse alors que le langage de balisage est essentiellement utile afin de présenter les travaux de recherche, que ce soit dans un document écrit ou sur un site web. C'est principalement de cette manière que sont utilisés les langages de programmation et de balisage dans le cadre de la recherche en sciences sociales.

8.2 Réflexion théorique

La plupart des langages de balisage permettent de remplir l'une des deux fonctions suivantes, qui sont particulièrement importantes dans le contexte de la recherche en sciences sociales : produire des documents écrits et gérer des pages Web. Dans les deux cas, cependant, certains sites Web et applications permettent également de remplir ces fonctions, mais avec des limites importantes.

Pour l'écriture de documents très simples comme une liste d'épicerie ou des notes rapides pendant une conférence, les logiciels de traitement de texte sont tout-à-fait convenables : ils sont simples et rapides à utiliser, un formatage professionnel du document n'est pas de mise. Utiliser un langage de balisage pour des tâches de base n'est en effet pas nécessaire. Par contre, plus la complexité d'un document augmente, plus il devient difficile d'obtenir un résultat satisfaisant en utilisant un logiciel de traitement de texte tel que Word, Pages ou Writer. A contrario, LaTeX permet de produire des documents de tous les niveaux de complexité, tel que démontré sur la Figure **??**. Plus généralement, utiliser un langage de balisage comme

[1] Les logiciels permettent plus ou moins efficacement d'identifier les balises problématiques. Certains ne produisent qu'un message d'erreur sans donner d'indication sur la source du problème, alors que d'autres ciblent très spécifiquement la ligne de syntaxe où se situe la balise problématique.