R à l'usage des sciences sociales

SÉMINAIRE RUSS

R au long cours : retours sur trois années de thèse avec R en géographie.

Présentation à l'Ined, Aubervilliers, le 2 février 2024.

Maxime Guinepain
Doctorant Géographie-cités (UMR8504) et ESO Nantes (UMR6590)
ATER Sorbonne Université (Licence Géographie/Master TLTE)
maxime.guinepain@posteo.net

ined INSTITUT NATIONAL D'ÉTUDES DÉMOGRAPHIQUES

ESO NANTES UMR 6590 Espaces et Sociétés

Géographie-cités UMR 8504

LETTRES SORBONNE UNIVERSITÉ

Présentation de mon travail

- Étude de la mobilité quotidienne des personnes en emploi en France
- Une approche pensée au départ comme relevant des méthodes mixtes

La mobilité comme domaine commun ?

Comment les mobilités quotidiennes reflètent-elles le positionnement social des travailleur·ses ?

Comment prolongent-elles les rapports sociaux (Kergoat, 2001 et 2011) à l'œuvre au travail et au sein des ménages ?

La mobilité comme sujet de conflit ?

Dans le contexte de la valorisation sociale d'un mode de vie fondé sur de longs déplacements, comment négocier la transition écologique dans les classes populaires ?

Présentation de mon travail

- Étude de la mobilité quotidienne des personnes en emploi en France
- Une approche pensée au départ comme relevant des méthodes mixtes
  - Un travail qui s'appuie sur l'exploitation d'une base de donnée « massive » (>600.000 individus), la Base Unique du Cerema, dans le cadre du projet Mobiliscope
  - Des analyses statistiques et spatiales qui sont centrales dans le processus d'administration de la preuve

R dans le cadre d'une thèse

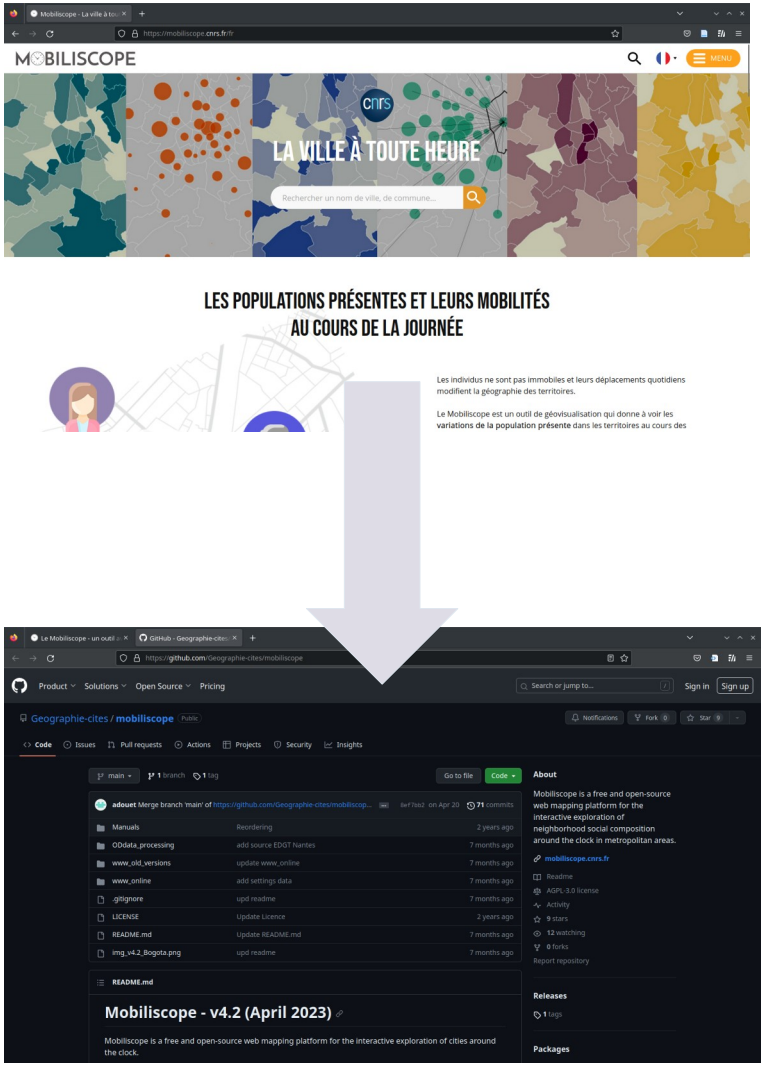
- Des données particulièrement nombreuses, avec une architecture particulière
- Des « aménagements » de ces données qui produisent de nombreux fichiers intermédiaires
- De nombreuses analyses qui partent parfois dans des directions différentes
- Un retour permanent sur des « vieux » bouts de code
- Cependant, une cohérence à maintenir

→ *À discuter :* sûrement des points communs avec d'autres types de projets…

Cependant, une autre particularité : travail individuel.

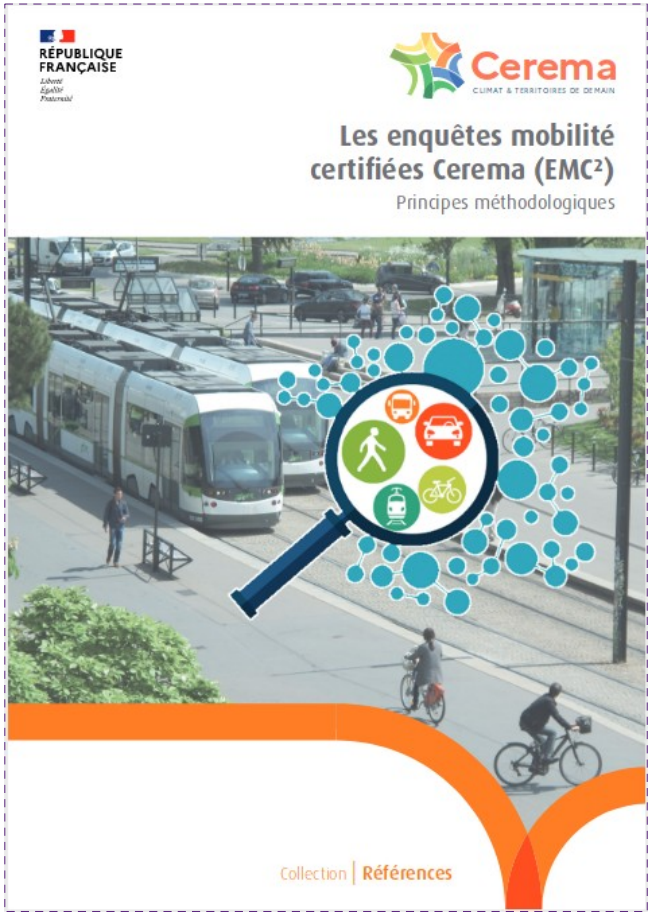
Le Mobiliscope

- La thèse s'inscrit dans le cadre du projet Mobiliscope, dont elle exploite une partie de la base de données.
- Par cet objet commun, par ma formation et par des questionnements convergents, la thèse reprend une architecture propre au projet et s'inspire de la façon dont il est programmé.



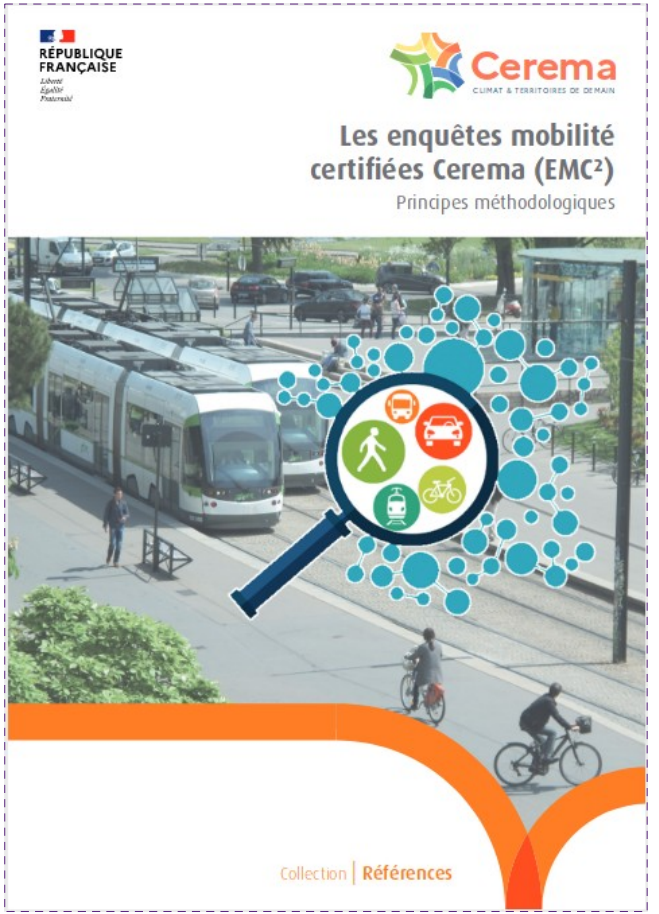
La Base unique du Cerema

- Une base issue des enquêtes de déplacement certifiées Cerema
- Des données standardisées, issues de nombreuses enquêtes (= quelques soucis méthodologiques…)
- Non pas « une » mais cinq bases encastrées



Les enquêtes origine-destination du Cerema

- Une méthodologie standardisée, héritée des enquêtes de mobilité des années 1960 (ancien « standard CERTU »)
- Questions générales sur les mobilités + relevé horodaté des déplacements de la veille (jour ouvrable) sur 24h, par ménage (2 ou tous les individus), localisé sur un maillage *ad hoc*



Un exemple de questionnaire qui montre la complexité de la donnée

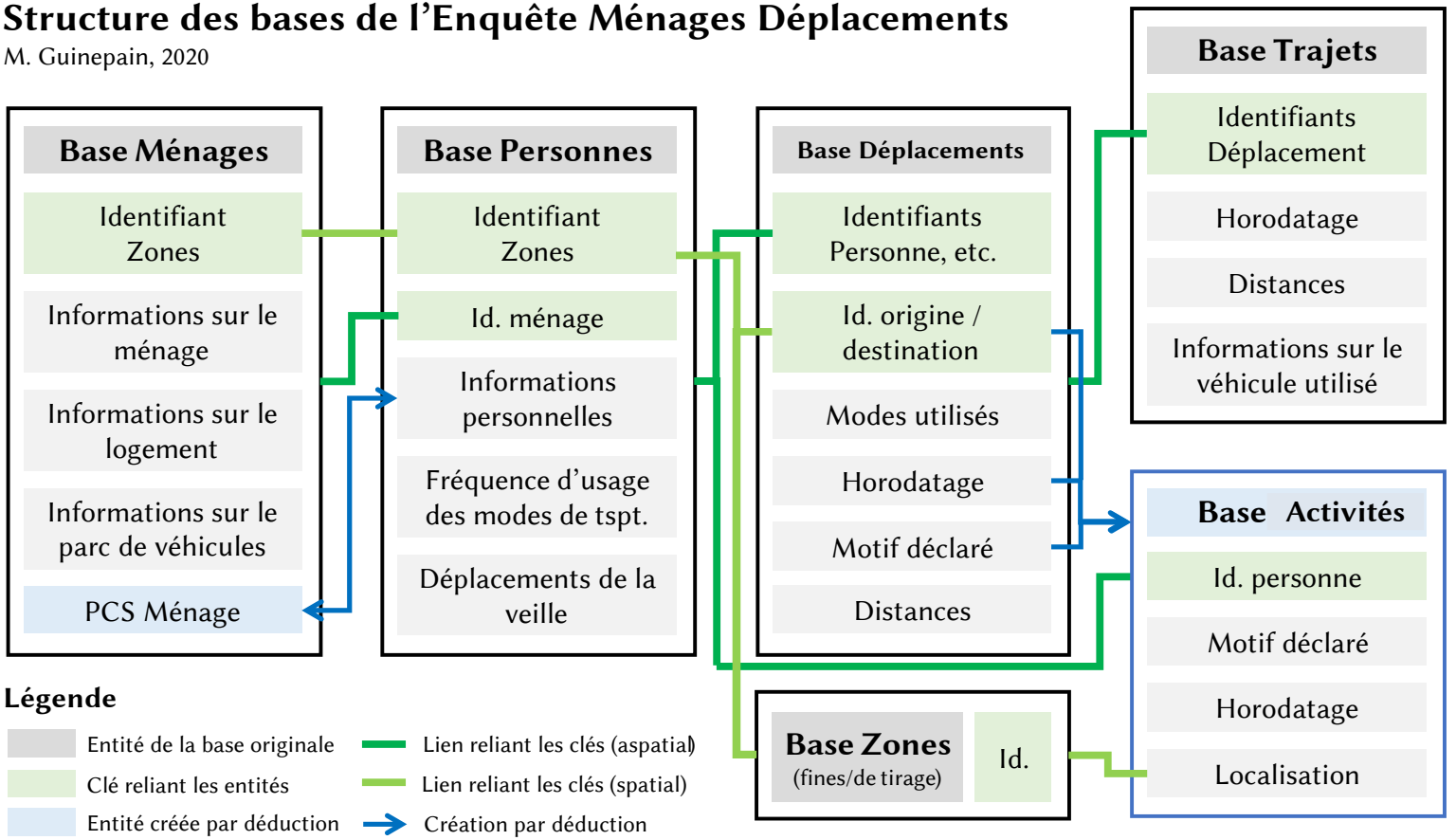
LES DÉPLACEMENTS → DESCRIPTION DES DÉPLACEMENTS									
	ORIGINE DU DÉPLACEMENT			DESTINATION DU DÉPLACEMENT					
D1 N° du déplacement	**D2** Motif de la personne *(plus éventuellement celui de la personne accompagnée)*	**D3** Zone fine origine	**D4** Heure de départ	**D5** Motif de la personne *(plus éventuellement celui de la personne accompagnée)* *Si motif tournée (**81 ou 82**), passer à **D6**. Sinon passer à **D7**.*	**D6** Si D5 **= 81 ou 82** (motif tournée) *Indiquer le nombre d'arrêts, sur le **premier déplacement** de la tournée*	**D7** Zone fine destination	**D8** Heure d'arrivée	**D9** Durée du déplacement	**D10** Nombre de modes mécanisés utilisés pour effectuer le déplacement *(coder **0** si le déplacement est fait **uniquement à pied***)
		voir cartes	heures minutes			voir cartes	heures minutes	en minutes	
PREMIER DÉPLACEMENT	**Si MAP uniquement, après D10 déplacement suivant**				**Si mode(s) mécanisé(s) → description du ou des trajets**				
⌊⌋	⌊⌋⌊⌋	⌊⌋⌊⌋⌊⌋	⌊⌋⌊⌋	⌊⌋⌊⌋	⌊⌋	⌊⌋⌊⌋⌊⌋	⌊⌋⌊⌋	⌊⌋⌊⌋	⌊⌋

Extrait du questionnaire de l'EDGT Loire-Atlantique

La Base unique du Cerema

Structure des bases de l'Enquête Ménages Déplacements

M. Guinepain, 2020



La Base unique du Cerema

- Certaines bases sont accessibles via Progedo (avec des caractéristiques spécifiques)
- Certaines ont été mises en ligne par leurs maîtres d'œuvre sur des portails d'OpenData
  - *C'est le cas de l'EDGT Loire-Atlantique 2015, exemple traité lors de cette séance.*

D'autres enquêtes en accès libre :

Royan (2014), Lille (2016), Grenoble (jusqu'à 2016), Poitiers (2018), Brest (2018)…

Mais des cas d'enquêtes qui ont « disparu »

Calvados, Nord Pas-de-Calais

Itinéraire de formation

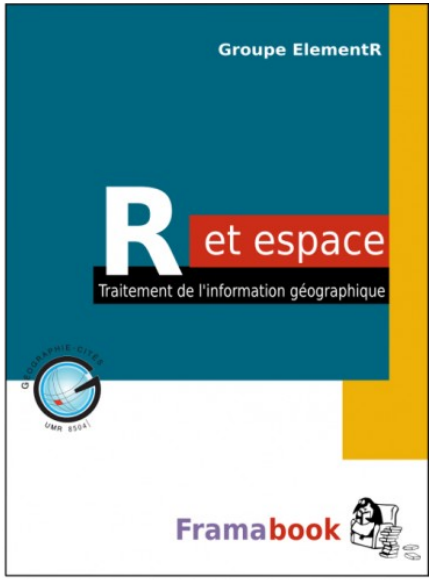
- Comment se former pour apprendre, mais aussi pour progresser en R ?
- Une formation continue nécessaire
  - On rencontre souvent de nouveaux problèmes inédits…
  - Le projet R dans son ensemble évolue, apportant de nouvelles possibilités qui facilitent la vie
  - Ces évolutions *imposent* parfois des changements dans le code si on veut travailler avec la version de R la plus à jour…

Itinéraire de formation

Exemple concernant ma trajectoire

- Des bases de programmation (comme « hobby ») en Basic et en C#
  - *Des langages qui, par certains aspects, ressemblent à R, et des « tics de langage »…*
- Formation à l'ENS de Lyon en 2017
- Formation à Paris-Cité dans le cadre du master Géoprisme en 2019
- Autoformation en mars 2020…

Le manuel *R et espace*, une bonne synthèse sur les méthodes les plus utilisées en géographie : statistiques et analyse spatiale



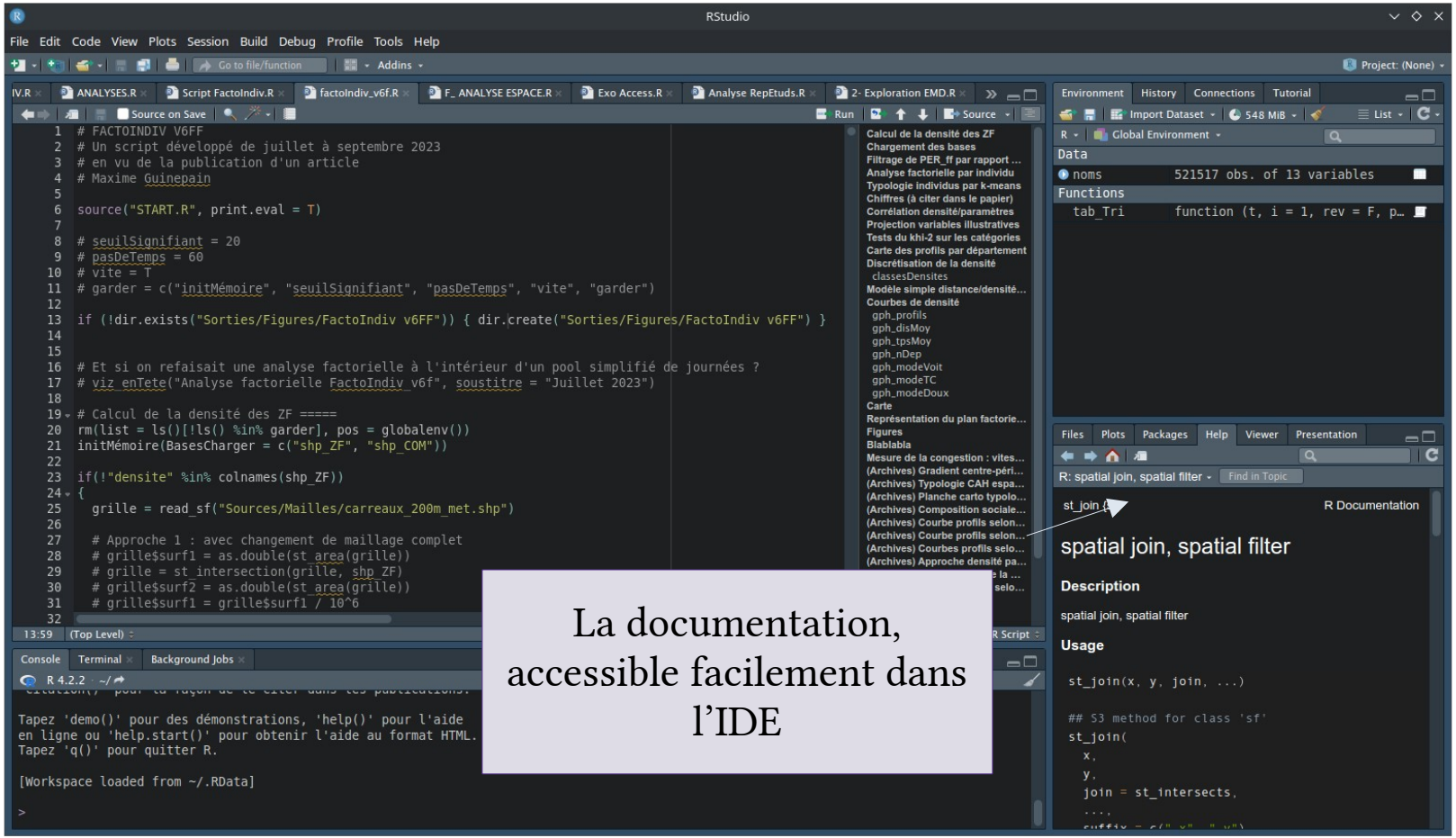
Ressources de formation

- Les ateliers pour débuter
- Des exercices sur des cas d'étude
- Un manuel générique « papier » (tel qu'*R pour les géographes*) ou de véritables manuels numériques tels que : https://juba.github.io/tidyverse/

Mais aussi…

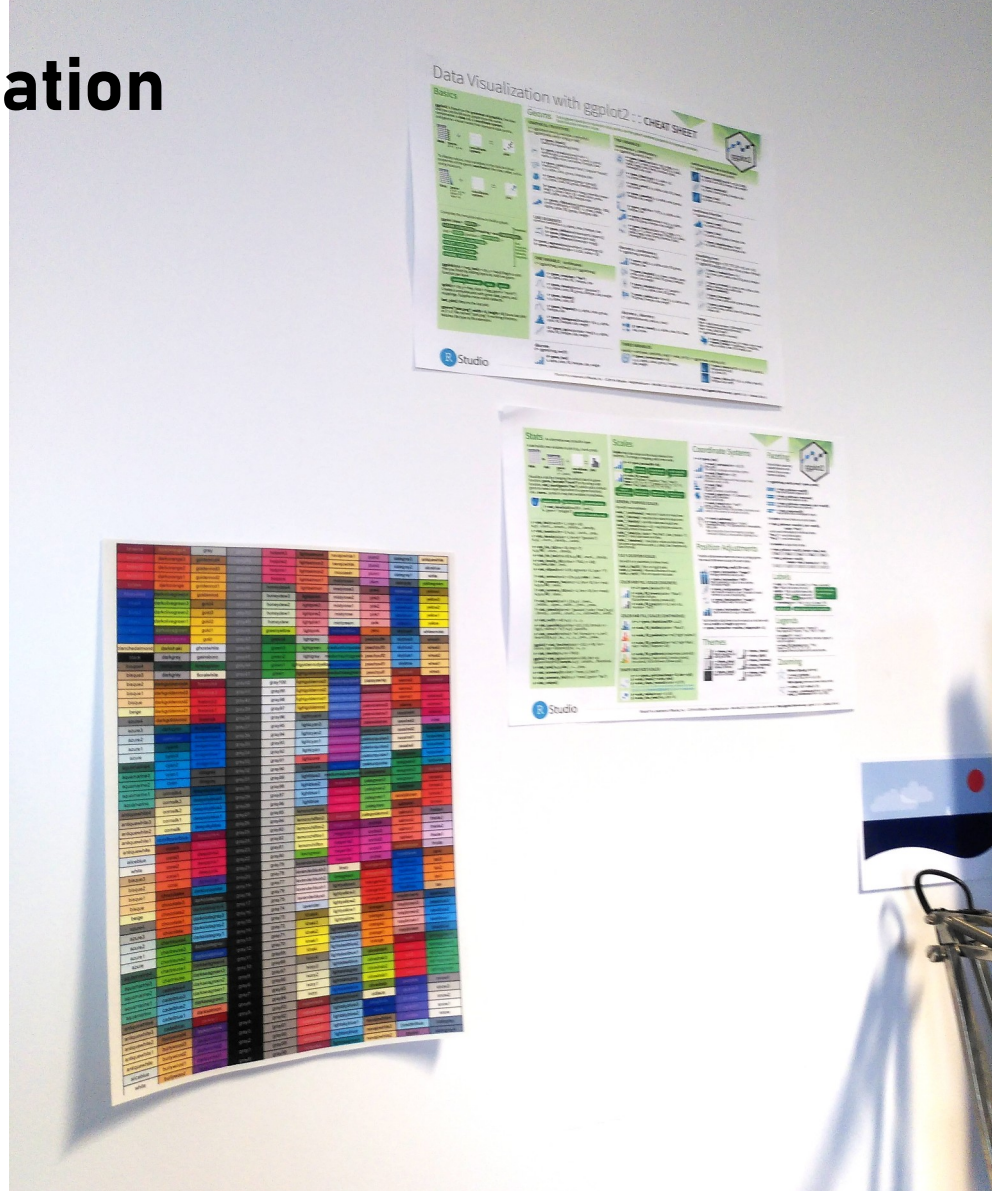
- La documentation, souvent très utile, détaillée et riche d'exemples dans R
- La « paradocumentation » = les « cheatsheet », essentielles, notamment au début, et toujours comme aide-mémoire ensuite
- StackOverflow, énormément

Ressources de formation

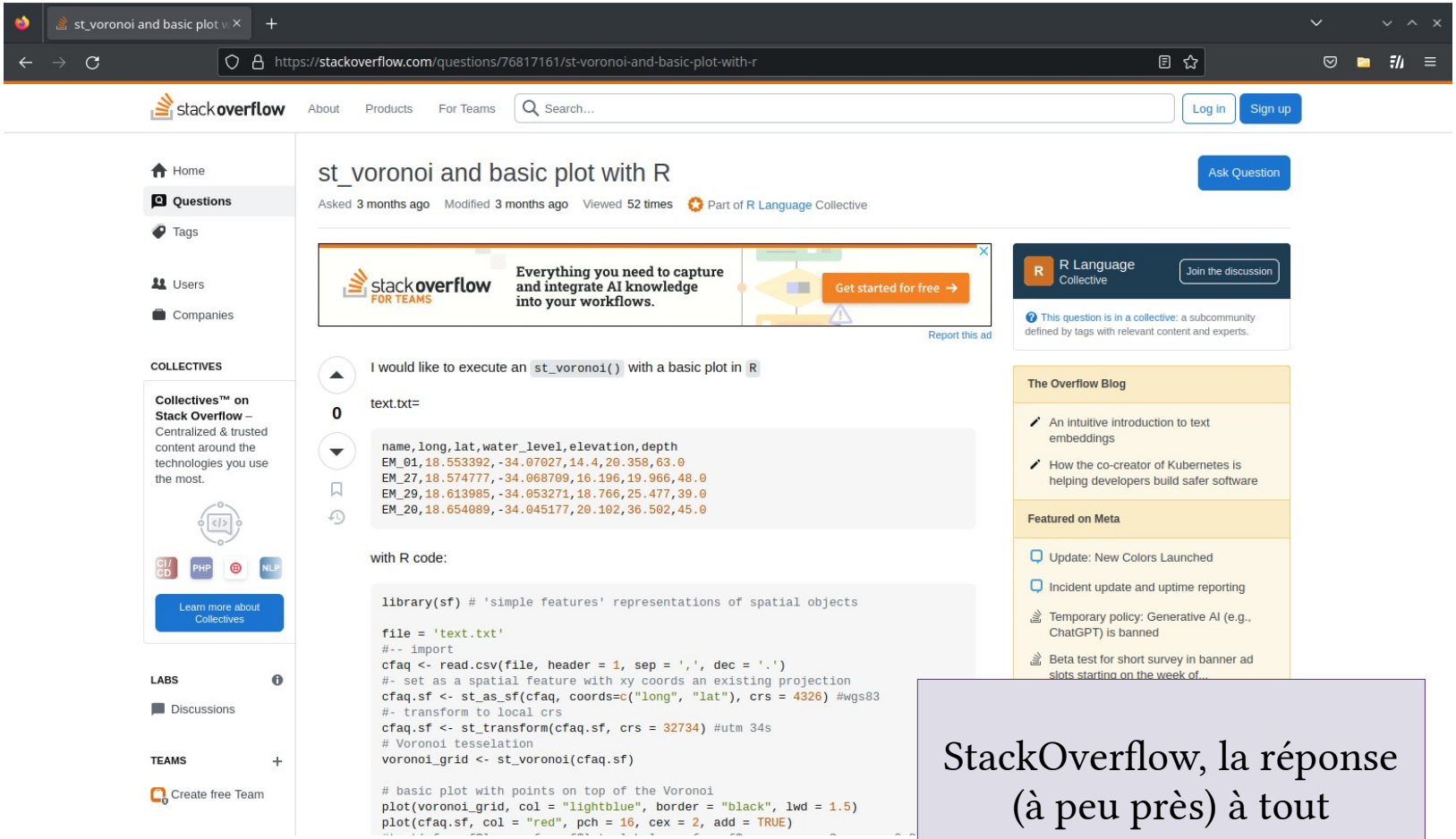


Ressources de formation

Les cheatsheets, à toujours garder sous la main



Ressources de formation



StackOverflow, la réponse (à peu près) à tout

Ressources de formation

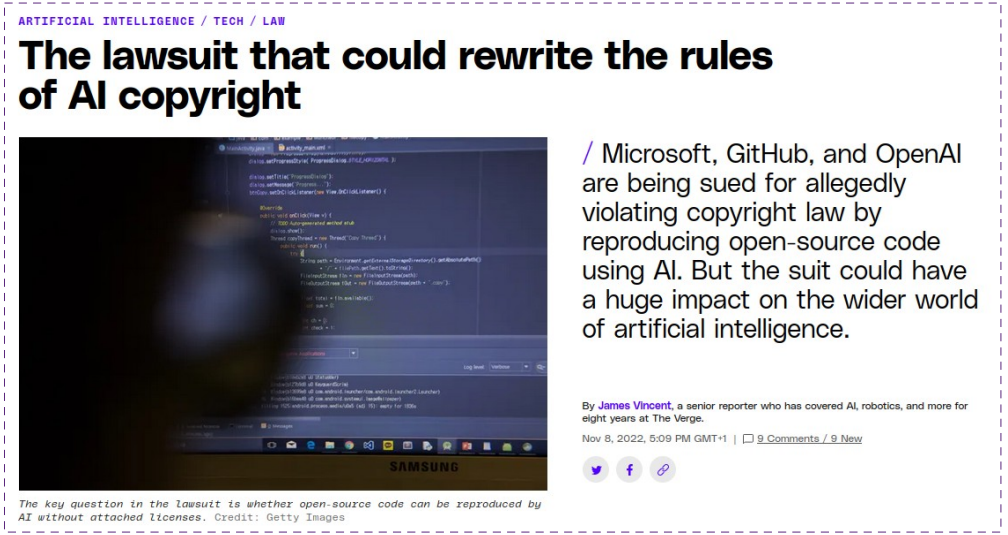
- Des questions détaillées, avec (en principe) des exemples reproductibles
- La plupart des questions trouvent une, voire plusieurs réponses, ce qui permet de comparer les solutions proposées. Elles sont parfois complémentaires et, dans l'ensemble, le ton est bienveillant
- Possibilité de poser ses propres questions, même si je ne l'ai jamais fait…
- Réinterroge parfois la pertinence de la démarche !
- **Mais** : il faut parler anglais

StackOverflow, la réponse (à peu près) à tout

Ressources de formation

Quid de ChatGPT ?

- Des réserves éthiques : les IA se nourrissent du code mis à disposition librement par les développeur·ses sans respecter les termes des licences (poursuites judiciaires en cours, scandale au départ autour de Copilot sur GitHub…)
- Cependant, une aide précieuse, en particulier pour les développeur·ses non anglophones



ARTIFICIAL INTELLIGENCE / TECH / LAW

The lawsuit that could rewrite the rules of AI copyright

/ Microsoft, GitHub, and OpenAI are being sued for allegedly violating copyright law by reproducing open-source code using AI. But the suit could have a huge impact on the wider world of artificial intelligence.

By James Vincent, a senior reporter who has covered AI, robotics, and more for eight years at The Verge.

Nov 8, 2022, 5:09 PM GMT+1 | 9 Comments / 9 New

The key question in the lawsuit is whether open-source code can be reproduced by AI without attached licenses. Credit: Getty Images

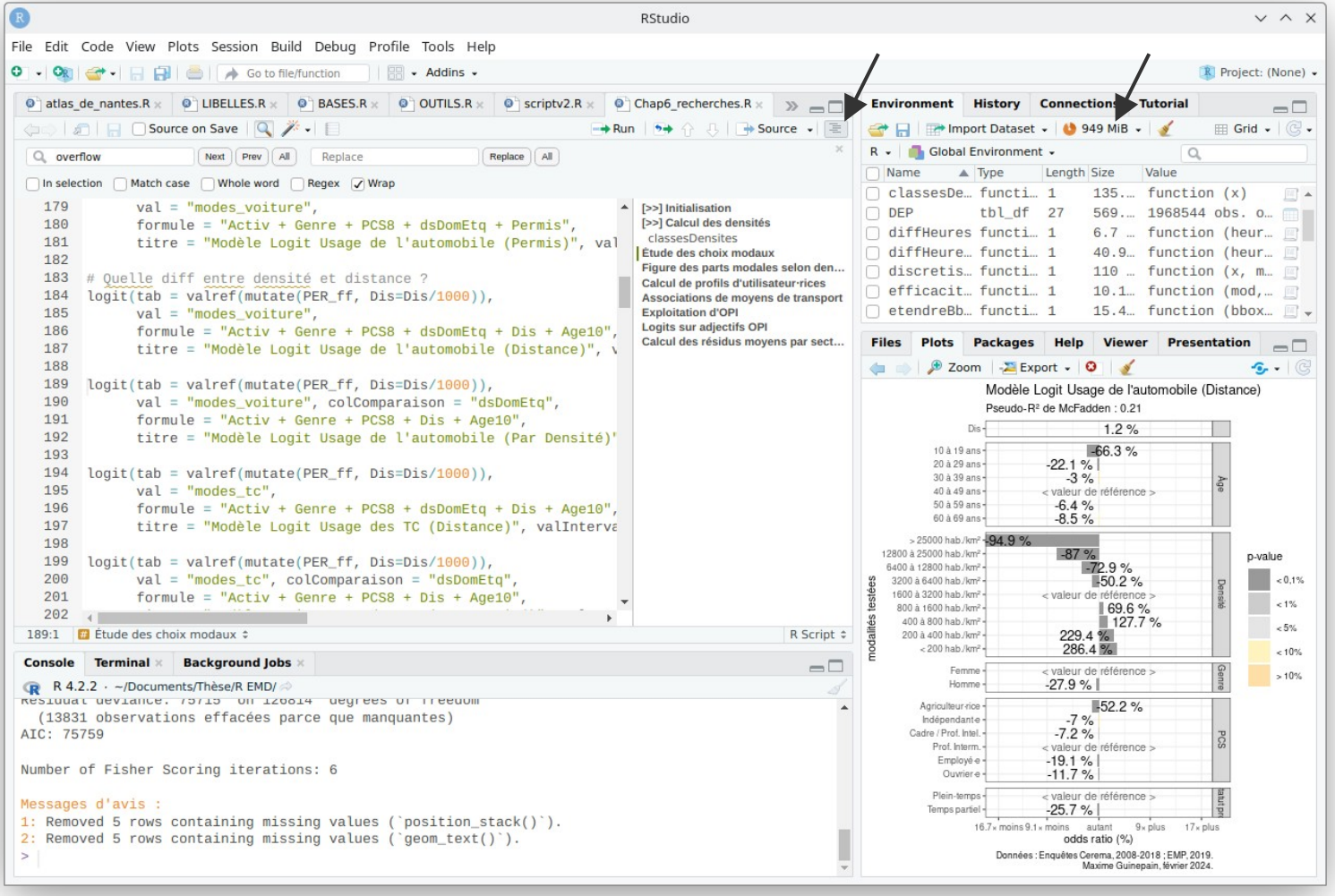
→ un accompagnement utile pour apprendre à utiliser certains outils, moins utile pour « faire le travail »… pour l'instant
→ pas toujours les réponses les plus pertinentes

L'IDE : Rstudio

- Le système des onglets permet de gérer plusieurs scripts à la fois, même si ce n'est pas toujours si facile.
- Le mode projet est pratique aussi (mais j'ai des soucis avec la sauvegarde des .Rdata).
- Accès facilité aux sorties et aux manuels.

- J'en fais une utilisation assez minimale…

L'IDE : Rstudio

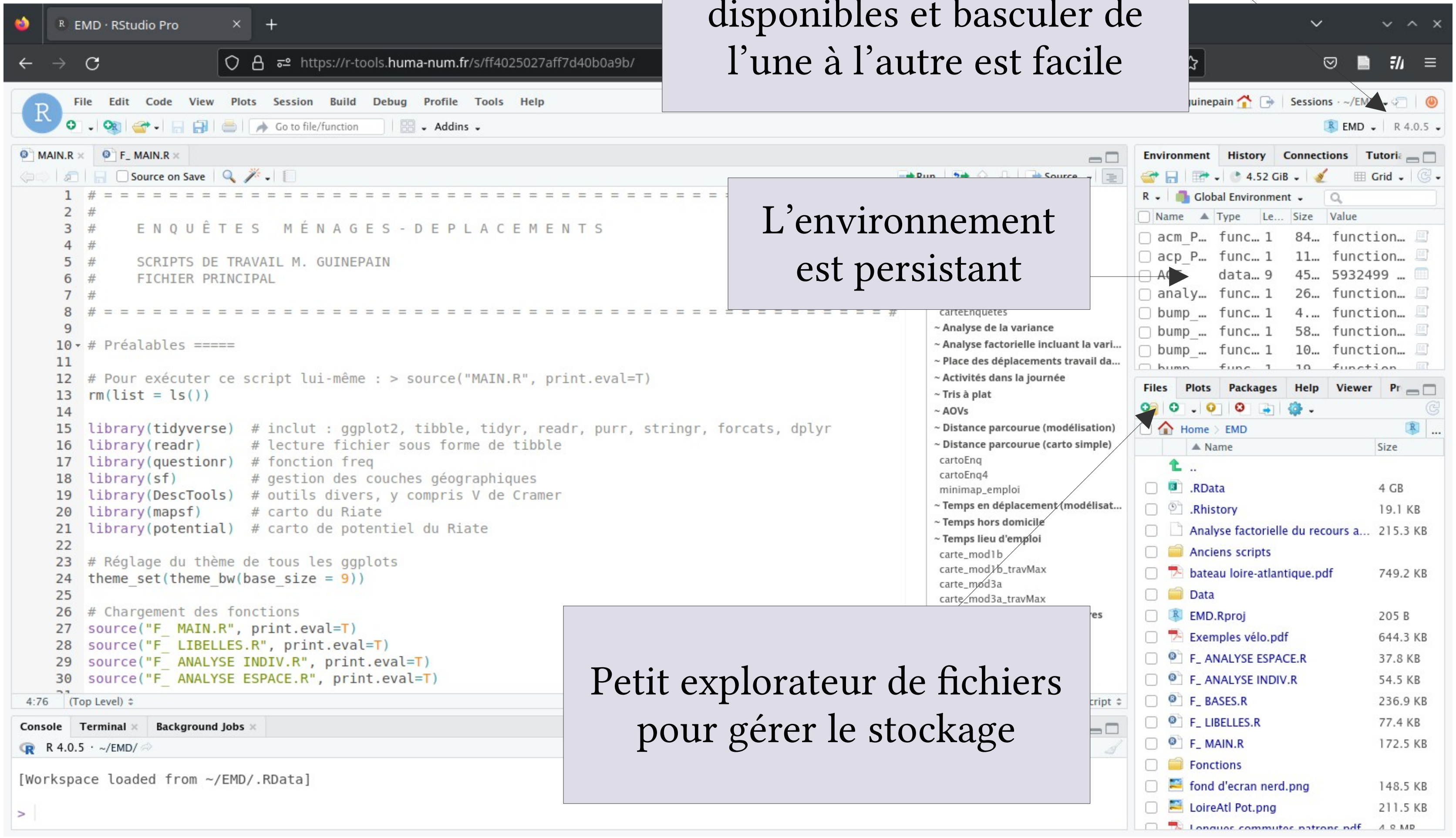


R dans HumaNum

- Très pratique : reprend fidèlement l'interface de Rstudio, si ce n'est qu'elle est dans le navigateur et en ligne.
- Très efficace : possibilité d'effectuer des calculs sur un ordinateur léger en performance (ils sont faits à distance sur le serveur), voire en se déconnectant.
- Une grande capacité de mémoire vive qui permet de manipuler sans efforts de très lourdes bases de données.
- Possibilité de travailler sur plusieurs ordinateurs, sauvegarde des scripts, travail à plusieurs…

- Présentation
- Ressources & formation
- Environnement logiciel
- Mode d'organisation
- Discussions

R dans HumaNum



Plusieurs versions de R sont disponibles et basculer de l'une à l'autre est facile

L'environnement est persistant

Petit explorateur de fichiers pour gérer le stockage

R dans HumaNum

- Cependant, l'installation de paquets par soi-même est compliquée (même si elle semble possible, dans le répertoire créé pour l'utilisateur·rice)
- Quelques soucis de mise à jour parfois, mais il y a une équipe réactive à contacter !

Le tidyverse

- Réunit un grand nombre de paquets utiles
- Notamment dplyr, tibble, readr, ggplot2…
- Un chargement facile avec un seul appel de `library`
- Attention aux conflits avec d'autres paquets, souvent plus anciens

Tidyverse et ggplot, un langage dans le langage…

- Le paquet dplyr : un travail systématisé sur les bases
- Possibilité de faire des opérations complexes avec des commandes simples, et de systématiser des opérations sur des colonnes (en les appelant toutes, ou selon la nature de la variable, ou selon une partie de leur nom…)
- Pionnier de l'utilisation du « pipe » %>%

  (intégré en RBase avec |>)

Tidyverse et ggplot, un langage dans le langage...

- Ggplot : une approche complètement différente par rapport à plot()
  - Configuration d'un objet graphique qui n'est tracé qu'à la toute fin
  - Ces objets graphiques peuvent être mis dans l'environnement et on peut agir sur eux à volonté
  - Ils peuvent être imbriqués et combinés à l'aide du paquet cowplot
  - Nécessite de disposer la base en format « long » (facile avec dplyr)
  - Utilise l'opérateur + pour combiner les instructions, ce qui est très bizarre !

Tidyverse et ggplot, un langage dans le langage…

- Ggplot : une approche complètement différente par rapport à plot()
  - Configuration d'un objet graphique qui n'est tracé qu'à la toute fin
  - Ces objets graphiques peuvent être mis dans l'environnement et on peut agir sur eux à volonté
  - Ils peuvent être imbriqués et combinés à l'aide du paquet cowplot
  - Nécessite de disposer la base en format « long » (facile avec dplyr)
  - Utilise l'opérateur + pour combiner les instructions, ce qui est très bizarre !
- Traçage automatisé de la légende qui force à employer de bonnes pratiques, mais limite les possibilités. Ceci étant, tout est paramétrable au millimètre près…

Les tibble et readr

- Les tibble : des bases de données, en mieux (qu'on peut construire depuis un fichier source avec readr).
- L'amélioration est surtout cosmétique (sortie dans la console avec plus d'indications, des couleurs, etc.)
- Le gros plus pour moi : la gestion de la nature des variables, qui est un peu moins souple mais évite bien des problèmes.
  - Avec readr, on peut forcer la reconnaissance des colonnes d'un fichier CSV dans les types que l'on souhaite ⟶ moins de soucis de chaînes qu'il faut transformer en facteurs, d'identifiants interprétés comme des chiffres…
  - Readr permet aussi d'utiliser un « dictionnaire » pour importer les bases. Très pratique !

Les tibble et readr

Exemple : extrait du dictionnaire pour lire la base Ménage de l'EDGT Loire-Atlantique.

L'utilisation des dictionnaires facilite énormément la lecture des fichiers FWF (sans séparateurs !) et le passage d'une base à l'autre.

Variable	Type	Position	Taille	Libelle
MP1	f	1	1	CF
ANNE	c	2	6	EngDate
MTIR	c	8	3	ZT
MP2	c	11	3	ZF
ECH	c	14	4	Ech
M1	f	18	1	LogType
M2	f	19	1	LogOcc
M3D	f	20	1	LogInternet
M5	i	21	1	VehN
M7A1	f	22	1	Veh1_Eng
M13A	i	23	4	Veh1_Ann
M14A	i	27	2	Veh1_Psc
M7B1	f	29	1	Veh2_Eng
M13B	i	30	4	Veh2_Ann
M14B	i	34	2	Veh2_Psc

Autres paquets utilisés

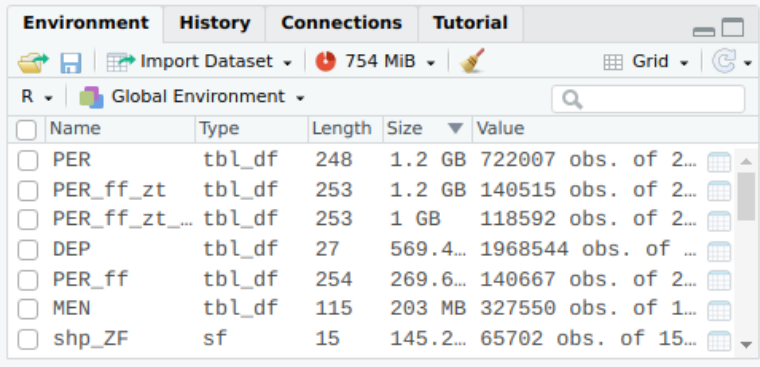
- Pour gérer les données spatiales, le paquet sf
- Pour la cartographie, mapsf
- Quelques paquets supplémentaires selon les besoins (plyr pour le recodage des variables, spatstat… sans doute pas les plus efficaces)
  - Certains paquets génèrent des conflits. C'est le cas de plyr avec le tidyverse.
  - On peut les appeler uniquement quand on a besoin avec l'opérateur :: → `plyr::revalue()`.

Limites matérielles sous R

- Difficile de travailler sur de grandes bases de données avec R sur un ordinateur ancien
- La principale limite est moins la puissance de calcul que la **mémoire vive**, qui semble mal gérée sous R : il faut que l'ensemble des bases puisse y être stockée en permanence, et si la capacité est dépassée, R plante
  - Cela peut advenir au cours de l'exécution du projet, mais aussi lors de son chargement (si .RData excède la capacité de la mémoire à lui tout seul)
- La gestion de la mémoire est un peu difficile à comprendre…
  - Supprimer les objets de l'environnement ne semble pas toujours libérer de la mémoire

Limites matérielles de R

- Bien surveiller l'occupation de la mémoire par les objets de l'environnement, dont la taille est affichée dans l'onglet « Environnement » en mode tableau
- Garder un œil, si possible, sur la mémoire vive restante de l'ordinateur
- Éviter de travailler avec moins de 8 Go de mémoire…
- Effacer les bases au fur et à mesure de leur utilisation et éviter les duplicats, créer des versions avec les colonnes nécessaires uniquement…
- Générer de la mémoire virtuelle



Le poids brut des bases apparaît dans la colonne « Taille »

Les boucles

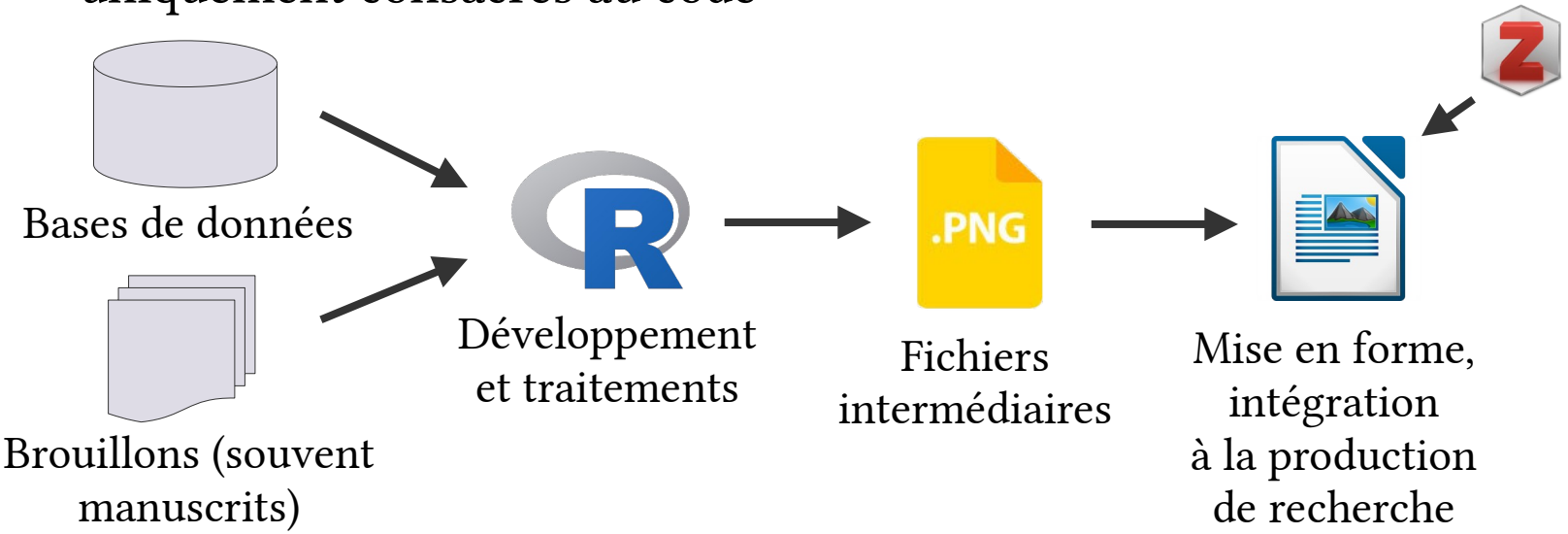
- Outre les contraintes de mémoire vive, R ne semble pas bien gérer les boucles (`while{}`, `for{}`), qu'on utilise pourtant très souvent dans d'autres langages
- Il faut s'appuyer sur des fonctions telles que `apply()` ou `lapply()` dont la syntaxe et l'usage n'est pas évident
  - Prennent une fonction comme argument
  - Sortie sous forme de tableau ou d'objet liste (qui n'est pas si facile à manipuler)
  - Toujours regarder des exemples…

Le problème des versions…

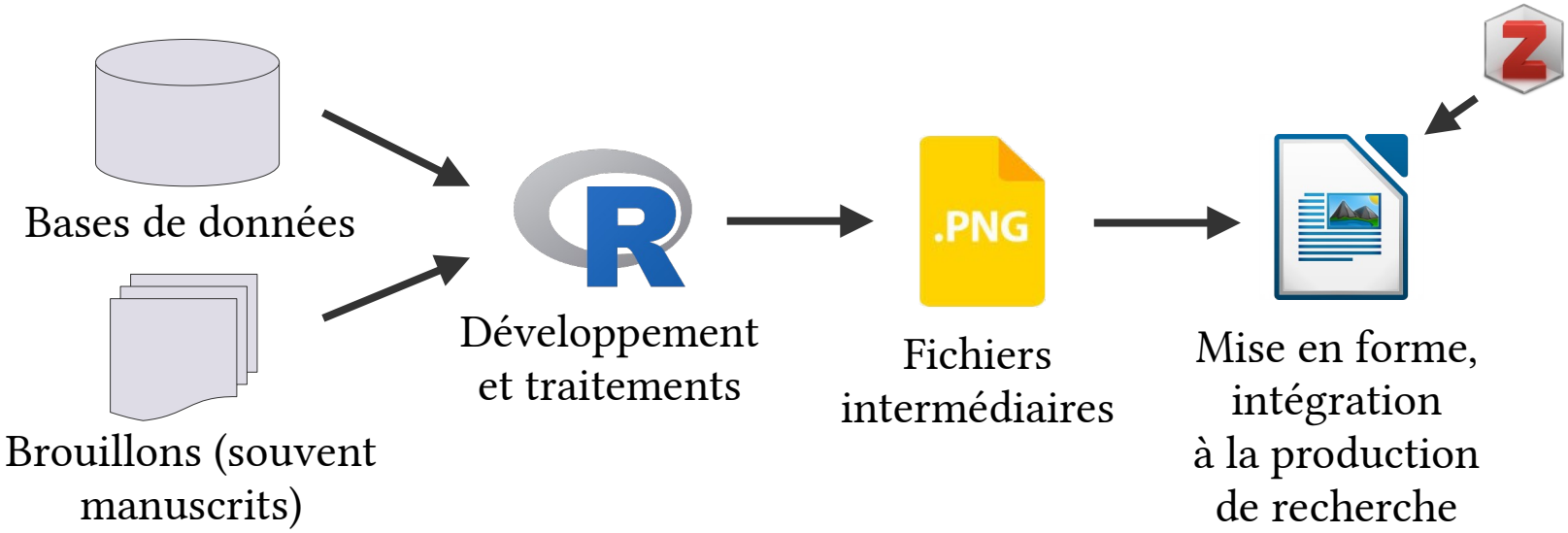
- R est un projet vivant, le logiciel et les paquets sont régulièrement mis à niveau… et cela peut « casser » le code
- Situations rencontrées, auxquelles être vigilant·e
  - Une fonction mise à jour qui n'accepte plus les mêmes arguments : m'est arrivé avec dplyr récemment. Parfois, est précédé d'un avertissement avant la mise à jour (« attention, va prochainement expirer »)…
  - Un paquet qui n'est plus mis à jour par ses créateur·rices et qui cesse de fonctionner sur une version de R mise à jour.

Un « workflow » sans automates

- Le « workflow » = l'organisation du travail dans ses différentes étapes, dont R n'est qu'un élément
- Des possibilités d'intégration, voire d'automatisation, offertes par les outils RMarkDown
- Cependant, je préfère travailler « à l'ancienne », dans des scripts uniquement consacrés au code



Un « workflow » sans automates



Nécessite pas mal de travail d'organisation des répertoires, de la gestion de versions, etc.

Envisager de travailler avec **Git**, système de versionnage et de synchronisation pensé pour le code
→ Permet de suivre l'évolution du code, de revenir en arrière… de travailler ensemble aussi !

Éviter les redites

- But : ne pas réécrire de nombreuses fois les mêmes bouts de code, et pouvoir les « mettre à jour » (pour faire face à des mises à jour, pour améliorer certains rendus ou à mesure que mon « niveau » augmente…)
- Une « architecture » de scripts qui font référence les uns aux autres, inspirée de comment fonctionnent les projets de programmation logiciels
- Tout est rassemblé dans un même répertoire qui représente l'environnement du projet

```
source("START.R", print.eval = T)
```

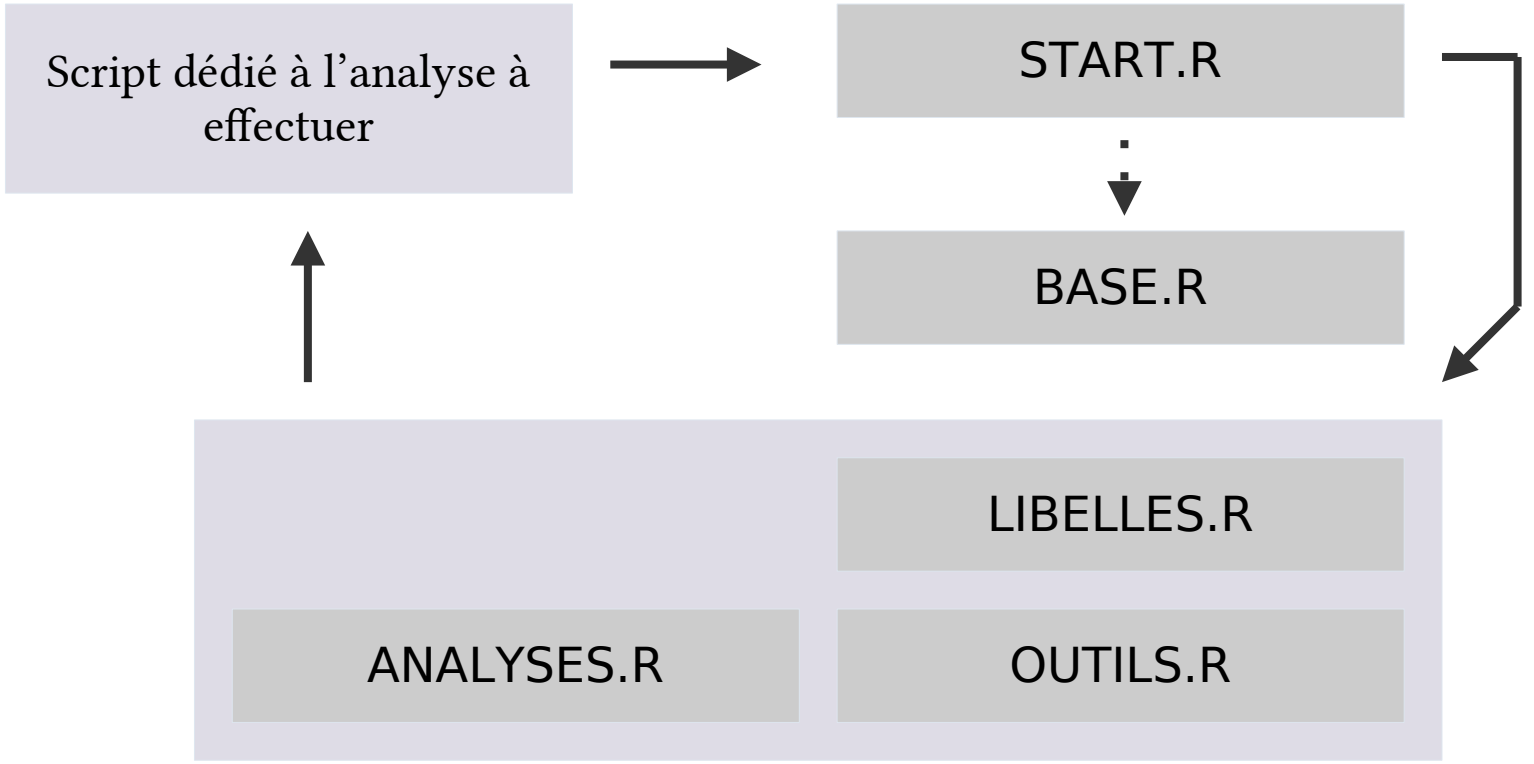
Des fonctions partout

- Les fonctions comportent une série d'instructions à appliquer (en général à leurs arguments)
- Elles peuvent effectuer des actions « en silence » ou rendre un résultat (sous la forme d'un objet)
- Elles constituent dans R un objet de l'environnement, comme les variables, les bases…
- Elles permettent d'exécuter de nombreuses fois une même séquence de codes (avec des arguments qui peuvent changer)
  - Dans certains cas, il est nécessaire d'écrire des fonctions pour pouvoir paralléliser ou appliquer à de grands ensembles de données une même séquence de code (type `lapply()`).

Des fonctions partout

- Les fonctions de l'environnement local sont à utiliser comme n'importe quel fonction d'un paquet : on les appelle par leur nom, on leur donne des arguments (nommés si on veut) entre parenthèses (qui sont obligatoires pour les appeler), on peut attribuer leur sortie avec l'opérateur `<-` ou `=`.
- Cependant, elles n'exécutent pas le code de la même manière qu'une ligne de code hors fonction :
  - Les sorties consoles doivent être explicitement formulées à l'aide de `cat()` ou `print()`, tout comme les sorties graphiques ;
  - Les fonctions opèrent dans leur propre environnement et laissent l'environnement général inchangé.

Un ensemble de scripts interdépendants




```
source("START.R", print.eval = T)
```

(utiliser des chemins relatifs)

Un script « Start »

- Permet d'aller plus vite en mettant dans un seul fichier ce que je devais faire à chaque fois
  - Appel des paquets que j'utilise
  - Appel des scripts secondaires qui contiennent toutes mes fonctions (futur paquet?)
  - Règle les paramètres de ggplot, …
  - Charge les bases dont j'ai besoin à la suite
  - J'en profite pour y introduire des variables que j'utilise dans tous mes scripts et que je peux vouloir changer partout en même temps (exemple : seuil de significativité, pas de temps de calcul pour les analyses temporelles…)
- Démarrer rapidement en initialisant l'environnement de travail de zéro : on charge START.R (après avoir choisi le bon répertoire), on lance sa fonction principale et on est « prêt·e à travailler » sur n'importe quel ordinateur

Des fonctions partout

- Un exemple de fonction très bête :

```
# Equivalent de dev.off(),
# sans sortie intempestive sur le terminal
off = function()
{
  o = dev.off()
}
```

- Autre exemple simple :

```
mode <- function(x) {
    ux <- unique(x)
    u_mode = ux[which.max(tabulate(match(x, ux)))]

    return(u_mode)

    # https://stackoverflow.com/questions/64678599/most-frequent-value-in-a-given-column
    # https://stackoverflow.com/questions/2547402/how-to-find-the-statistical-mode/8189441#8189441
}
```

Des fonctions partout

- Quatre types de fonction :
  - Pour **standardiser** (typiquement, les sorties graphiques)
  - Pour **envelopper** (= se substitue dans mon code à des fonctions natives ou issues de paquets, dont j'adapte le fonctionnement à mes besoins)
  - Pour **des opérations pratiques que je fais souvent** (= aller plus vite sur certaines opérations : conversions, tris…)
  - Pour exécuter des analyses complexes en parallèle ou plusieurs fois ou pour clarifier le cheminement du code, et éviter d'y répéter de gros blocs indigestes avec quelques caractères de différence

Des fonctions partout → pour standardiser

- Fonction `sortie(nom = "sortie", format = "png", taille = "def", portrait = F, h = 11, l = 17, chemin = NULL, pointsize=8)`
- Son but :
  - Gagner du temps sur les appels des fonctions graphiques dans R (type `pdf()`, `png()`...)
  - Utiliser toujours les mêmes paramètres (formats, définition...), y compris pour l'enregistrement des fichiers
  - Gagner en flexibilité (possibilité de changer le format en changeant simplement les arguments)

Des fonctions partout → pour standardiser

- Fonction `src_fig()`
- Son but :
  - Ne pas répéter dix mille fois les mentions des sources dans le code !
  - Pouvoir mettre à jour d'un seul coup toutes les mentions s'il y a un changement dans les données
  - Adapter le texte en fonction des données utilisées (par exemple, elle accepte comme argument la base à sourcer et détecte alors ce qu'il faut indiquer)
  - Gadget : générer automatiquement la date actuelle…

Des fonctions partout → pour « envelopper »

- Exemple : analyseFacto() et categ_kMeans() ou categ_cah().
- But = « envelopper » des fonctions qui existent déjà (les fonctions PCA() et MCA() ou les fonctions kmeans() et cluster()) en mettant les données en forme, en générant des graphiques ou des informations systématiquement utiles, en ajoutant des étapes de traitement.
  - Exemple : tri des données et réattribution des n° des clusters de la fonction kmeans() pour en stabiliser les résultats, car cette fonction de catégorisation repose sur un processus aléatoire.
- Avantage : s'il faut changer de fonction d'analyse (notamment pour des problèmes de mise à jour…), c'est facile !
  - Passage facile du paquet FactoMineR à ade4.

Des fonctions partout → pour des trucs pratiques

- Exemple typique : fonctions de conversion horaires fournies dans le fichier exemple (HHMM vers un nombre à décimale de minutes…)
- D'autres exemples dans le fichier : une échelle symétrique pour voir des données de sur/sous-représentation /100, une fonction pour centrer-réduire avec des pondérations, une fonction de tri de tableau que je préfère aux fonctions natives…
  - Des procédures partagées entre doctorant·es : les fonctions facilitent aussi ça :)
- Parfois, c'est plus pratique d'écrire sa propre fonction pour faire ces petites choses que d'en chercher une dans les paquets disponibles !

Des fonctions partout → pour clarifier

- Choix fait dans le code BASE.R qui contient de nombreuses étapes : utiliser des fonctions pour clarifier le cheminement (et appliquer les mêmes opérations aux différentes bases qui composent la base de l'enquête quand les opérations à effectuer sont similaires, par exemple pour générer des identifiants uniques).
- Inconvénient : peut conduire à perdre de vue les opérations qui sont vraiment effectuées.

Des fonctions partout → pour clarifier

- Un cas où je pense qu'il n'est pas bon d'utiliser des fonctions : pour se dispenser d'apprendre la syntaxe…
  - Au début de ma thèse, j'utilisais une fonction pour tracer les graphiques en barre à l'aide de ggplot pour ne pas avoir à manipuler la syntaxe très compliquée propre à ce paquet.
  - Plutôt une mauvaise idée : vite limitant, vite très encombrée car je devais la complexifier pour répondre à des besoins parfois spécifiques, et a ralenti mon apprentissage de ggplot, qui a fini par s'avérer incontournable.

Le fichier « LIBELLES »

- Au départ, conçu pour stocker les vecteurs utilisés pour recoder les niveaux des variables et les palettes pour les graphiques !
- À présent, j'y mets des fonctions qui recodent les variables au vol et selon mes besoins

```
niv_PCS8  = c("Agriculteur·rice",
              "Indépendant·e",
              "Cadre / Prof. Intel.",
              "Prof. Interm.",
              "Employé·e",
              "Ouvrier·e",
              "En études",
              "Inactif·ve",
              "PCS inconnue", "PCS inconnue")

pal_PCS8  = c("darkolivegreen3",
              "goldenrod",
              "steelblue",
              "darkorchid",
              "rosybrown",
              "firebrick3",
              "grey60", "grey", "grey40")
```

```
etqPCS8 = function(champ, num=F, low=F, prefixe = F, rev=F, genre="both")
{
    numeros = c("01", "02", "03", "04", "05", "06", "07", "08", "09", "00")

    if (genre[1] == "both") { etiqs = niv_PCS8 }
    if (genre[1] == "H") { etiqs = niv_PCS8_H }
    if (genre[1] == "F") { etiqs = niv_PCS8_F }

    if(prefixe){ numeros = paste0("PCS8", numeros) }

    if (num) { etiqs[1:7] = paste(numeros[1:7], etiqs[1:7]) }
    if (low) { etiqs = tolower(etiqs) }

    liste = set_names(nm = numeros, x = etiqs)
    champ = plyr::revalue(champ, liste, warn_missing = F)

    if(!prefixe){
        if (!rev) { champ = factor(x = champ, levels = unique(etiqs)) }
        if (rev)  { champ = factor(x = champ, levels = rev(unique(etiqs))) }
    }

    return(champ)
}
```

Le cheminement des données

- Les données sources restent stockées dans un répertoire. Aucune modification ne leur est apportée. Il est toujours possible de tout régénérer à partir de ce répertoire.
- Les données transformées sont stockées dans le format .Rds (commande `save()`) dans un autre répertoire.
- Contrôle de l'état des données dans le script (est-il nécessaire de générer une colonne ou peut-on se reposer sur la base sauvegardée ?)
- Fonction pour vérifier régulièrement qu'il n'y a pas de doublons (liés à des jointures mal faites…) !

La journalisation

- Utilisation d'une petite fonction `rapport()` qui permet d'écrire du texte à la fois sur la console et dans un petit fichier texte, où il est horodaté.
- La base est aussi versionnée (numéro incrémenté à chaque génération complète).
- Idée =
  - pouvoir comparer le résultat d'une nouvelle exécution du code avec l'ancienne,
  - contrôler le déroulement de l'exécution du code (notamment quand il est fait hors connexion sur HumaNum ou quand je laisse l'ordi tourner tout seul)
  - garder une trace des manipulations successives de la base.

La journalisation

de_nantes.R | LIBELLES.R | BASES.R | OUTILS.R | scriptv2.R | Chap6_recherches.R | START.R | Journal n°88.txt

```
1
2
3  ========== 2023-09-28 ================================================================================
4
5
6
7  [16:14:47] Initialisation complète des bases depuis les sources.
8
9
10 [16:16:23] Initialisation complète des bases depuis les sources.
11           Pas de calcul des indicateurs séquentiels fixé à 60
12 [16:16:23] Chargement base des Pôles Générateurs de Trafic
13 [16:16:23] Chargement de la base des PGT...
14 [16:16:24] Adaptation des CRS des fichiers sources et fusion...
15 [16:16:25] Association aux données communales...
16 [16:16:39] Association aux ZF...
17           Impossible d'associer certains PGT à des codes de Zone Fine : 421 erreur(s)
18 [16:16:57] Chargement base Ménage
19 [16:17:02] Chargement base Personnes
20 [16:17:03] Application du patch manuel de correction des erreurs de composition de ménage
21 [16:17:33] Vérification de la composition des ménages (depuis la base Personnes).
22           0 doublons d'UID trouvés parmi les Ménages.
23           0 doublons d'UID trouvés parmi les Personnes.
24 [16:18:00] Calcul de la variable 'Bourdieu'
25 [16:18:06] Typologies de ménages...
26 [16:18:06] Préparation des PCS Ménage
27           Échec d'attribution des PCSM : 54.658256138444 %
28           Part d'erreurs constatées dans les PCSM simples, niveau 1 : 1.32 %
29           Part d'erreurs constatées dans les PCSM simples, niveau 2 : 1.32 %
```

1:1 Text file

Questions ?

:)

La portée pédagogique de R

- Un questionnement sur la substitution de R en cours aux tableurs et logiciels de SIG.
- Est-ce plus pratique pour les étudiant·es ?
  - Rare encore sont celles et ceux qui ont vraiment appris à coder dans les filières de géo. Coût d'entrée qui peut sembler abrupt par rapport à l'utilisation de tableurs et de logiciels de SIG, surtout quand ce sont des étudiant·es avancé·es qui en ont fait avant.
  - Toutefois, on s'arrache tellement les cheveux avec ces logiciels que la question se pose : serait-il vraiment pire d'apprendre quelques bases de code ?
  - La question se pose d'autant plus alors que de nombreux·ses étudiant·es, utilisant des tablettes et smartphones, ne sont plus très familier·es avec les interfaces « clique-bouton »

La portée pédagogique de R

- Des avantages sûrs
  - Possibilité de dérouler des scripts de façon logique, de corriger des erreurs a posteriori, d'annoter les scripts pour mémoriser les étapes de traitements
  - Pas besoin de se préoccuper des réglages des logiciels et de contourner l'étrange logique qui préside à la conception de leurs interfaces…
  - Une compétence valorisable par la suite
- Mais…
  - Quelle pertinence au regard des projets professionnels ?
  - Quelles injustices causées par le recours au code, qui peut impressionner ?
  - Quels problèmes d'accessibilité pose le recours à des scripts ?

Le code ouvert et l'informatique libre

- R (et Rstudio, bien que ce produit soit aussi vendu par une compagnie) s'inscrivent dans le logiciel libre
  - Bonne intégration avec Linux, passés certains obstacles :)
- Philosophie du libre dans la mise à disposition des paquets, encourage à la mise à disposition du code
- Intégration à réaliser avec les dépôts git, mise à disposition de ressources…
  - En projet : la création d'un paquet à partager !
  - Mais possibilité aussi de copier et de partager de simples fonctions, de consulter le code pour s'inspirer de la conception de quelqu'un d'autre…
  - Nécessité de bien documenter, commenter toujours !

Merci de votre attention

Pour me contacter :
maxime.guinepain@posteo.net

Remerciements spéciaux :
- Timothée, Bénédicte et l'équipe RUSS
- Hugues, Robin, Léa, Marion, et l'équipe ElementR
- Julie et Aurélie du projet Mobiliscope,
- Julien, Luc, Pierre, Ronan, Romain…
pour nos discussions « code »