TP Bien démarrer son projet

Ludovic Deneuville

Introduction

Contexte

Vous allez démarrer votre projet statistique de première année.

Vous avez déjà été sensibilisés à la gestion de projet et à l'utilisation de l'intelligence artificielle. Aujourd'hui, nous allons voir comment bien démarrer un projet en nous appuyant sur des outils et des bonnes pratiques que vous pourrez réutiliser tout au long de votre formation et lors de vos stages.

> **i** Note
>
> Même si le but est de travailler en équipe, ce TP est à réaliser de préférence individuellement. L'objectif est que chacun puisse comprendre les principes, utiliser les outils et manipuler les données.

Objectifs

Les objectifs 🎯 de cette séance sont :

- S'organiser en équipe
- Initialiser votre dépôt de code
- Importer et nettoyer vos données
- Mener les premières analyses
- Initier votre rapport de projet

Vous aurez besoin de vos comptes SSP Cloud et GitHub.

> **i** Fil rouge
>
> Vous allez utiliser les données du Loto pour essayer de devenir millionnaire. 💰 🐓 💵

1 Démarrer un projet

Avant de commencer, prenez le temps de faire connaissance avec les membres de votre groupe et de définir votre organisation.

Discutez notamment :

- des forces et compétences de chacun : programmation, statistiques, rédaction, organisation
- des appétences de chacun : quelles tâches aimez-vous particulièrement ?
- de vos éventuelles contraintes d'organisation
- de vos modes de communication : outils utilisés, fréquence des échanges, réunions
- de la répartition des rôles et des premières tâches
- d'une première version macro de votre calendrier

Les rôles ne sont pas nécessairement fixes : ils peuvent évoluer au cours du projet. L'objectif est que chacun participe aux différentes dimensions du projet et que le travail ne repose pas sur une seule personne.

Osez prendre des initiatives !

Un projet est aussi l'occasion de faire preuve de curiosité et d'autonomie. N'hésitez pas à proposer des idées, à explorer des pistes qui n'ont pas été évoquées en cours, à tester de nouveaux outils ou à chercher des solutions par vous-même.

Toutes les initiatives ne seront pas nécessairement concluantes, et ce n'est pas un problème : expérimenter, se tromper et ajuster sa démarche font pleinement partie du travail de projet. L'essentiel est de rester **force de proposition** et de **partager vos découvertes** avec le reste de l'équipe.

2 Données

Commençons par une partie facile, déposons nos données dans un endroit sûr.

- ☐ Récupérez sur internet l'historique des tirages du Loto depuis novembre 2019

Vous allez ensuite vous connecter au Datalab pour le déposer sur votre stockage S3.

- ☐ Connectez-vous sur https://datalab.sspcloud.fr
- ☐ Allez dans stockage de données
- ☐ Créez un dossier nommé **diffusion**

🚧 Lien vers partage de données :

- diffusion : lecture seule
- partage de fichier : 1 semaine max
- rendre public un dossier : écriture ?

- ☐ Dans ce dossier, créez un dossier **projet**, puis un dossier **input**
- ☐ Chargez le fichier du Loto

Vous avez maintenant vos données prêtes à être utilisées.

3 Créer et organiser son dépôt

Passons maintenant à la gestion de votre code avec Git.

3.1 Dépôt distant

Le dépôt distant (*remote*) est le dépôt commun à toute l'équipe. Il permet de vous synchroniser.

Votre code sera stocké dans un dépôt GitHub.

- ☐ Connectez-vous à GitHub
- ☐ Créez un nouveau Repository : GitHub ⏩ Repositories ⏩ New
  - Repository name : `projet-stat-1a-groupe<N>`
  - Description: *Projet statistique de première année*
  - Visibility Level : *Private*
  - ☑️ Cochez *Add a README file*
  - Add .gitignore : *R*
  - Choose a license : *MIT Licence* (ou une autre)
  - Cliquez sur Create Repository

Ceci n'est à faire qu'une seule fois.

> ! Important
>
> Pour le véritable projet, vous choisirez un seul des trois dépôts GitHub créés individuellement et l'utiliser comme dépôt commun à toute l'équipe.
>
> Son propriétaire devra alors inviter les deux autres membres comme collaborateurs (*Settings* ⏩ *Collaborators*). Vous pourrez ainsi tous travailler sur le même dépôt.

3.2 Dépôt local

Avant de créer votre dépôt local (code sur lequel vous allez travailler), il faut créer votre environnement de travail.

Vous allez ci-dessous utiliser les paramètres de lancement pour créer votre clone :

- ☐ Allez dans le catalogue des services
- ☐ Lancez un service Rstudio
  - Cliquez sur Lancer **une seule fois**
- ☐ Allez dans l'onglet Git
  - Vérifiez que votre Token est présent
  - Renseignez l'url de votre dépôt GitHub
- ☐ Cliquez sur Lancer
- ☐ Ouvrez le service

Si tout s'est bien passé, un clone de votre dépôt a été créé dans votre service. Vous voyez apparaitre votre dépôt et ses fichiers dans l'explorateur en bas à droite.

> i Si votre dépôt n'a pas été créé
>
> Dans le terminal de votre service, clonez manuellement en utilisant le token :
>
> - exemple : `git clone https://$GIT_PERSONAL_ACCESS_TOKEN@github.com/<username>/<repo_name>.git`

3.3 Rappels Git

Vous allez commencer par vérifier que vous arrivez bien à envoyer du code vers le remote.

Toutes les actions ci-dessous sont à faire dans Rstudio.

- ☐ Ouvrez le fichier *README.md*

! README, What do you think that means?

Le fichier README.md est le mode d'emploi de votre projet.

Il est essentiel : c'est la première porte d'entrée de votre dépôt. Il doit permettre à quelqu'un qui découvre votre projet de comprendre rapidement son objectif et de savoir comment l'utiliser ou y contribuer.

Il doit comporter au minimum :

- Présentation : objectif du projet et description de son contenu
- Fonctionnalités : principales possibilités offertes
- Environnement : outils, logiciels et packages nécessaires
- Mode utilisateur : instructions pour utiliser facilement et rapidement
- Mode développeur : instructions pour comprendre l'organisation du code et le compléter

Pensez à rédiger votre README comme si vous présentiez votre travail à une personne qui ne connaît pas votre projet !

- ☐ Collez le contenu ci-dessous 👇 et enregistrez

```
# Devenir millionnaire grâce au Loto ! 🍀

## Présentation
Analyser les résultats du Loto pour tenter de découvrir les combinaisons gagnantes !

## Fonctionnalités
- **Charger** : récupérer les données depuis S3.
- **Nettoyer** : apurer et préparer les données.
- **Analyser** : produire des statistiques et visualisations.

## Utilisation
1. Cloner le dépôt Git.
2. Ouvrir le projet dans RStudio.
3. Exécuter `renv::restore()` pour installer les dépendances.
4. Lancer les scripts d'analyse.
```

- ☐ Avec Git, envoyez ces modifications vers le remote
  - ▸ Faites un petit détour par la section Git si besoin
- ☐ Sur GitHub, vérifiez que le README est à jour

🚧 Exo pratique de gestion de conflit

3.4 Fichier .Rproj

.Rproj est le fichier de configuration de votre projet RStudio.

Il permet à RStudio de reconnaître votre dossier comme un projet et de conserver certains paramètres de travail.

i Si le fichier n'a pas été créé

- ☐ File ⏩ New Project ⏩ Existing Directory
- ☐ Sélectionnez `projet-stat-1a-groupe<N>` ⏩ Create project

Il permet notamment de :

- Définir le répertoire de travail : à l'ouverture du projet, RStudio se place automatiquement à sa racine
- Centraliser les paramètres : encodage des fichiers, indentation du code, etc.
- Faciliter la navigation : RStudio reconnaît les fichiers et les dossiers appartenant au projet.

Ce fichier est à versionner avec Git.

3.5 renv

Avant de commencer à coder, intéressons-nous à la gestion des packages.

Lorsque plusieurs personnes travaillent sur un même projet, il est important que tout le monde utilise les mêmes packages avec les mêmes versions.

renv permet de créer un environnement R isolé et propre au projet et d'enregistrer les packages utilisés ainsi que leurs versions.

- ☐ Exécutez dans la console la commande `renv::init()`
  - Do you want to proceed? ➡️ `y`

Cette commande crée un dossier *renv/* et un fichier *renv.lock* (qui stocke la version de R utilisée, les packages nécessaires et leurs versions).
Ne les supprimez pas ! Ils doivent être versionnés dans votre dépôt.

> **Commandes utiles**
>
> - `renv::install("<package>")` pour installer un package
> - `renv::dependencies()` : quels sont les packages utilisés dans le code
> - `renv::status()` : pour voir les différences entre les packages installés et *renv.lock*
> - `renv::snapshot()` : enregistre l'état des packages dans *renv.lock*
> - `renv::restore()` : applique *renv.lock* i.e. désinstalle les packages en trop, installe les packages manquants

Vous allez avoir besoin des packages suivants :

- **dplyr** : manipulation de données (pipes, mutate, select…)
- **stringr** : manipulation de chaînes de caractères

- ☐ installez-les avec renv
- ☐ Vérifiez que les packages installés sont dans le renv.lock
- ☐ Créez un commit et poussez vos modifications

3.6 Charger les librairies

Installer un package ne suffit pas pour pouvoir l'utiliser dans un script R.

Il faut également le charger avec `library()`. Pour éviter de répéter ces appels dans tous les scripts, nous les regroupons dans un fichier `01_initialisation.R`.

> **Note**
>
> Vous l'avez sans doute remarqué, ça commence à devenir le bazar à la racine du dépôt. Pour bien ranger les fichiers contenant du code, vous allez créer un dossier *src* (sources).

- ☐ Créez un dossier *src*
- ☐ Dans ce dossier, créez un fichier nommé *01_initialisation.R*
- ☐ Collez-y quelques librairies dont vous aurez besoin
  - À compléter par la suite si nécesssaire

```
# renv::install("aws.s3")
library(aws.s3)

library(dplyr)
library(stringr)
library(lubridate)
library(tidyr)
```

4 Charger et apurer

Il est maintenant temps de coder !

4.1 Charger les données brutes

Commençons par charger le fichier.

- ☐ Créez à l'intérieur un script `02_import.R`
- ☐ Récupérez vos données depuis S3 et chargez-les dans un dataframe que vous nommerez `loto_raw`
  - Vous trouverez comment faire sur 👉 cette page 👈
  - Il manque des packages ? Installez-les !

```
bucket_name <- "ludo2ne"
file_path <- "diffusion/projet/input/loto_201911.csv"

communes <- aws.s3::s3read_using(
  FUN = readr::read_delim,
  delim = ";",
  object = file_path,
  bucket = bucket_name,
  locale = readr::locale(decimal_mark = ","),
  opts = list("region" = "")
)
```

En projet stat, pour charger un fichier `.rds`, utilisez la fonction `readRDS`.

> ! Important
>
> Si lors du chargement, vous rencontrez une erreur 403 (Forbidden), il est probable que vos credentials S3 soient expirés.
>
> La solution est expliquée sur la même page que ci-dessus.

4.2 Mettre les données au propre

Certaines colonnes ne nous intéressent pas, certaines données sont mal formatées. Il faut mettre tout cela au propre avant de commencer les analyses.

- ☐ Créez un nouveau script que vous nommerez `03_nettoyage.R`

> 💡 Pourquoi créer un nouveau fichier ?
>
> Imaginez un unique script de 2000 lignes qui charge les données, les nettoie, réalise toutes les analyses et produit le rapport. Pour retrouver une instruction, bon courage ! Et si vous souhaitez modifier le nettoyage, gare aux effets de bord sur le reste du projet !
>
> En séparant les différentes étapes, vous rendez votre code plus lisible, plus facile à maintenir et à faire évoluer. Votre futur vous (et vos coéquipiers) vous remercieront !

À partir du dataframe *loto_raw*, vous allez créer un nouveau dataframe propre que vous nommerez *loto* :

summary(loto_raw$annee_numero_de_tirage)

- ☐ Observez la colonne *annee_numero_de_tirage*, vous ne remarquez pas des problèmes ?
- ☐ Utilisez le code ci-dessous 👇 pour harmoniser cet identifiant

```
loto <- loto_raw |>
  mutate(
    date_de_tirage = dmy(date_de_tirage), # conversion en format date
    annee_numero_de_tirage = paste0(
      year(date_de_tirage),
      str_sub(as.character(annee_numero_de_tirage), -3)
    )
  ) |>
  rename(id = annee_numero_de_tirage)
```

- ☐ Examinez les différentes valeurs présentes dans la colonne *jour_de_tirage*
  - Que constatez-vous ?
- ☐ Utilisez la fonction `str_trim()` pour corriger
- ☐ Vérifiez le résultat obtenu

```
loto <- loto_raw |>
  mutate(
    jour_de_tirage = str_trim(jour_de_tirage)
  )

table(loto$jour_de_tirage)

loto <- loto |>
  mutate(jour_de_tirage = str_trim(jour_de_tirage))
```

Certaines colonnes du jeu de données ne seront pas utiles pour notre analyse.

- ☐ Supprimez les colonnes concernant le second tirage, celles sur les *codes gagnants*, ainsi que celles nommées : *date_de_forclusion*, *numero_7*, *devise* et *...50*

Tip

Pour retirer des colonnes :

```
loto |>
  select(
    -contains("picsou"),
    -boule2
  )
```

```
loto <- loto |>
  select(
    -contains("second_tirage"),
    -contains("code"),
    -date_de_forclusion,
    -numero_7,
    -devise,
    -`...50`
  )
```

Vous avez maintenant un jeu de données propre. 🧼 🧹

- ☐ Exportez-le vers votre bucket S3 en créant un fichier *loto.csv*

```
aws.s3::s3write_using(
  loto,
  FUN = readr::write_delim,
  object = "diffusion/projet/input/loto.csv",
  bucket = bucket_name,
  opts = list("region" = ""),
  delim = ";"
)
```

Nous allons également nous interesser aux numéros.

- ☐ Créez ce dataframe

```
numeros <- loto |>
  tidyr::pivot_longer(
    cols = starts_with("boule_"),
    values_to = "numero"
  ) |>
  group_by(numero) |>
  summarise(
    nombre_sorties = n(),
    derniere_sortie = max(date_de_tirage)
  )
```

5 Une première analyse

Commençons par quelques statistiques descriptives.

Vous pouvez continuez à coder dans le fichier `03_nettoyage.R`

> ⚠ Warning
>
> Bien sûr que NON, c'est juste pour voir si vous suivez !
>
> - ☐ Créez un script nommé `04_stats_desc.R`

- ☐ Quels sont la moyenne, le min, le max de gains au rang 1 ?
  - i.e. si l'on trouve les 5 numéros, ainsi que le numéro chance
- ☐ Quelle proportion des tirages a vu le jackpot remporté ?

Il n'est pas possible de connaître directement le nombre de grilles jouées. Nous allons en proposer une estimation à partir des résultats du rang 9. La probabilité de gagner au rang 9 est d'environ 5,7 %.

- ☐ Estimez le nombre de grilles jouées pour chaque tirage
  - vous pouvez ajouter une colonne *nb_grilles_estime* au dataframe
- ☐ Quel jour de la semaine semble attirer le plus de joueurs ?
  - calculez le nombre moyen de grilles estimé pour chaque jour de la semaine
  - comme en SQL, regroupez puis calculez la moyenne

```
loto |>
  filter(nombre_de_gagnant_au_rang1 > 0) |>
  summarise(
    moyenne = mean(rapport_du_rang1, na.rm = TRUE),
    minimum = min(rapport_du_rang1, na.rm = TRUE),
    maximum = max(rapport_du_rang1, na.rm = TRUE)
  )

# Proportion de jackpots remportés
loto |>
  summarise(
    proportion = mean(nombre_de_gagnant_au_rang1 > 0) * 100
  )

# Estimez le nombre de grilles jouées pour chaque tirage
loto <- loto |>
  mutate(
    nb_grilles_estime =
      nombre_de_gagnant_au_rang9 / p_rang9
  )

# nombre de joueurs moyens par jour de la semaine
loto |>
  group_by(jour_de_tirage) |>
  summarise(
    moyenne_grilles = mean(nb_grilles_estime)
  )
```

- ☐ Affichez la distribution de la moyenne des cinq numéros

```
loto |>
  mutate(
    moyenne_numeros = rowMeans(across(starts_with("boule_")))
  ) |>
  ggplot(aes(x = moyenne_numeros)) +
  geom_histogram(
    bins = 50,
    fill = "steelblue",
    color = "white"
  ) +
  labs(
    title = "Distribution de la moyenne des cinq numéros tirés",
    x = "Moyenne des cinq numéros",
    y = "Nombre de tirages"
  ) +
  theme_minimal()
```

- ☐ Créez d'autres scripts pour vos futures analyses
  - exemples : `05_analyse_multivariee.R`, `06_acp.R`, `99_sandbox.R`

6 Préparer les livrables

Vous allez utiliser l'écosystème Quarto pour préparer vos livrables.

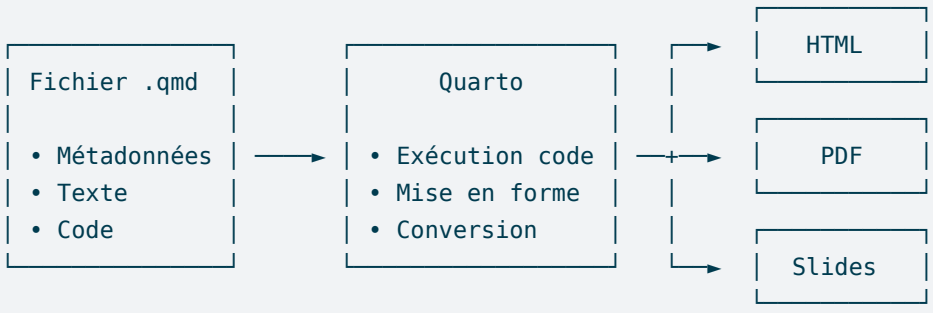
> **Tip**
>
> Vous allez explorer Quarto prochainement dans le cadre d'un TP de Stats avec R.

6.1 Quarto ?

Vous allez créer des fichiers avec l'extension `qmd` qui contiennent :

- Des métadonnées pour expliquer le format de sortie souhaité
- Votre texte
- Votre code à exécuter

6.1.a Fonctionnement



Il est possible de générer plusieurs types de rendu avec Quarto. Ici vous générerez uniquement un fichier pdf.

6.1.b Exemple de qmd

- Vous commencez en entêtes par décrire le fichier que vous souhaitez générer (entre les `---`)
- Puis vous écrivez votre contenu et votre code

````
---
title: Comment faire fructifier son patrimoine
author: Picsou
format:
  html: default
---

## Ne rien dépenser

La première règle de Picsou est simple : **dépenser le moins possible**.

```{r}
depense <- 0
revenu <- 1000
epargne <- revenu - depense
```

Avec cette méthode, Picsou économise `r epargne` pièces d'or !

## Conclusion {.unnumbered}

> Chaque pièce compte !
````

6.1.c Exemple de rendu

6.2 Initialiser son rapport

Vous allez utiliser Typst pour rédiger votre rapport.

Typst ?? Mais c'est écrit Quarto juste au dessus !? 😵

En fait Quarto utilise Typst comme moteur de mise en page, qui transforme le contenu en un document PDF. 😅

Un modèle vous est fourni. Vous trouverez sur ce site un exemple de rapport.

- ☐ Cliquez sur le lien ci-dessus
- ☐ Cliquez sur le bouton `</> Code` pour voir le contenu du rapport
- ☐ Cliquez sur le bouton `Typst` pour voir le rendu pdf

> **ℹ Remarque**
>
> Vous remarquerez dans le code du rapport qu'il y a une partie du code rédigée directement en Typst.
>
> C'est nécessaire pour avoir une page de garde et une table des matières propre. Ce code utilise un fichier model.typ qui vous est fourni. Vous avez simplement à renseigner quelques informations : titre, membres de l'équipe…

Initialisons maintenant votre rapport :

- ☐ Créez un dossier *doc*
- ☐ Dans ce dossier, créez un dossier *img*
- ☐ Ajoutez-y ce logo de l'ENSAI
  - Upload ⏩ Choisir un fichier ⏩ Collez l'url de l'image ⏩ Ouvrir

  - ⚠️ Important : renommez ce fichier `ensai_logo.png`
- ☐ Revenez dans le dossier *doc* et créez le fichier *model.typ* et collez-y son contenu
- ☐ Créez un fichier *rapport.qmd*`
- ☐ Pour initier votre rapport, collez le code ci-dessous 👇 qui contient :
  - métadonnées : décrit le fichier pdf à générer
  - chargement des données en R
  - utilisation du modèle typst pour générer la page de garde
  - corps du rapport en markdown

rapport.qmd

````
---
format:
  typst:
    papersize: a4
    margin:
      x: 1.5cm
      y: 2.0cm
    section-numbering: 1.1.a
    mainfont: "Libertinus Serif"
    fontsize: 11pt
execute:
  message: false
  echo: false
  warning: false
---

```{r}
#| label: librairies-and-load

source("../src/01_initialisation.R")
source("../src/02_import.R")
source("../src/03_nettoyage.R")
```

```{=typst}
#import "model.typ": project_template

#show: body => project_template(
  title: [Créer un rapport de projet\ avec Quarto et Typst],
  team: [Équipe 8],
  authors: ([Jean-Guy Badiane], [Serena Williams], [François Pignon], [Anne Hathaway]),
  tutor: [Philippe Lucas],
  project: [Projet Statistique de 1ère année - Année 2025-2026],
  body
)
```

## Introduction

```{=typst}
#lorem(100)
```

## Partie 1
````

Ajoutons une petite figure, dans la partie 1 :

- ☐ Affichez la distribution de la moyenne des gains au rang 1

```r
loto |>
  ggplot(aes(x = rapport_du_rang1)) +
  geom_histogram(
    bins = 20,
    fill = "steelblue",
    color = "white"
  ) +
  scale_x_continuous(
    labels = scales::label_number(big.mark = " ", decimal.mark = ",")
  ) +
  labs(
    title = "Distribution des gains au rang 1",
    x = "Gain au rang 1 (€)",
    y = "Nombre de tirages"
  ) +
  theme_minimal()
```

🔥 Qui veut gagner de l'argent en masse

Vous voyez des gains en milliard ?

Vous n'auriez pas oublié un petit truc, par exemple de préciser le séparateur de décimale au chargement du fichier ?

Dans le **terminal** de RStudio :

- ☐ Positionnez-vous dans le dossier *doc*
- ☐ `quarto render doc/rapport.qmd`
- ☐ Ouvrez le pdf généré ✨ 🤩 🍾

Vous avez maintenant toutes les cartes en main pour mener à bien votre projet : à vous de jouer !

Quelques conseils pour le rapport

Comment bien bâcler son rapport ? 💀 💣
(pas besoin de nous remercier, c'est gratuit)

1. Je m'y mets au dernier moment. Avec un peu de chance, les autres auront déjà fait une partie de mon boulot.
2. Je demande à une IA de me l'écrire : je ne veux pas rater mon cours d'aqua-poney.
3. **Introduction :** je n'utilise pas d'accroche, je ne présente ni problématique ni plan.
4. Je ne me préoccupe pas de la cohérence du rapport, ni sur le fond ni sur la forme.
5. Je ne fais pas de phrases de transition entre mes parties et sous-parties. De toute façon, mes parties ne suivent aucun ordre logique.
6. Je n'utilise pas de titres informatifs. « 1. Contexte, 2. Analyse, 3. Résultats », ça ira très bien.
7. J'utilise « on » et la forme passive : « on a noté », « on a vu », « on a remarqué que », etc. C'est ce qu'on m'a dit en techniques rédactionnelles. Ou c'était l'inverse !? Bref, je me souviens plus y'avait Chartres la veille.
8. J'essaie de battre le record de la phrase la plus longue. Ça permet au lecteur de s'entraîner pour l'apnée.
9. Je ne soigne pas mes figures (pas de titre, ni de source, ni de note de lecture). Les lecteurs n'ont qu'à deviner.
10. Je ne commente pas mes figures. Je les dépose simplement au milieu de nulle part : ça fait une demi-page de moins à rédiger.
11. J'oublie les statistiques descriptives. De toute manière, une moyenne ça ne veut rien dire.
12. Je fais une ACP sans préciser pourquoi j'ai choisi ces variables et ces individus. De toute façon, le logiciel sait ce qu'il fait.
13. Je lance un clustering sans trop réfléchir. Il paraît qu'on a eu un cours là-dessus, alors autant s'en servir !
14. Je ne me soucie pas de la mise en page. Les titres seuls en bas de page, les figures coupées et les pages presque vides donnent du caractère au document.
15. Je ne relis pas ce qui a été rédigé par les autres. D'ailleurs, je ne relis pas non plus ce que j'ai rédigé.
16. **Conclusion :** je demande à l'IA de reformuler mon introduction. Facile ! Je ne parle ni des limites ni des pistes d'amélioration.
17. Je ne cite aucun article de la bibliographie dans le rapport. D'ailleurs, je n'ai pas lu ce qui a déjà été fait sur le sujet.

6.3 Présentation

Optionnel, à faire plus tard en autonomie

- Avec RevealJS
- Répéter avant
- Backup
- Numéroter ses diapo
- Avoir son rapport ouvert s'il y a des questions

7 Liens utiles

- https://mtes-mct.github.io/parcours_r_socle_introduction/
- https://mtes-mct.github.io/parcours_r_socle_preparation_des_donnees
- https://book.utilitr.org/