

Introduction à



*Formation interne,
2024*

PROGRAMME

1. **Introduction : la programmation, R et RStudio**, *comprendre les principes de la programmation, et les spécialités du langage R et de l'interface RStudio ;*
2. **Importer des données ;**
3. **Nettoyer et transformer des données : le tidyverse**, *connaître les principales fonctions pour nettoyer et manipuler les données, avec des exercices de mise en pratique ;*
4. **Visualiser des données : ggplot2**

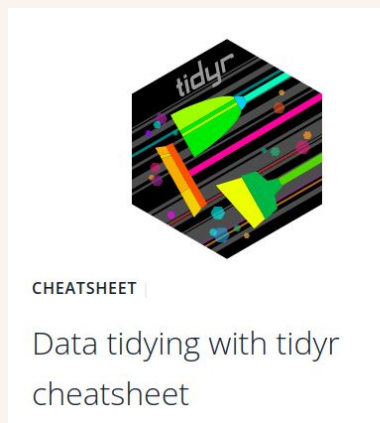
Séance n°1

La programmation, R et RStudio

```
    }).done(function(response) {  
      for (var i = 0; i < response.length; i++) {  
        var layer = L.marker(  
          [response[i].latitude, response[i].longitude]  
          // , {icon: myIcon}  
        );  
        layer.addTo(group);  
  
        layer.bindPopup(  
          "<p>" + "Species: " + response[i].species + "<br>" +  
          "<p>" + "Description: " + response[i].description + "<br>" +  
          "<p>" + "Seen at: " + response[i].latitude + " " + response[i].longitude + "<br>" +  
          "<p>" + "On: " + response[i].sighted_at + "</p>"  
        );  
      }  
  
      $('select').change(function() {  
        species = this.value;  
      });  
    });  
  }  
  $.ajax({  
    url: queryURL,  
    method: "GET"  
  }).done(function(response) {  
    for (var i = 0; i < response.length; i++) {  
      var layer = L.marker(  
        [response[i].latitude, response[i].longitude]  
        // , {icon: myIcon}  
      );  
      layer.addTo(group);  
    }  
  });  
}
```

Avant de commencer

: ❖ les fiches tuto de POSIT



<https://posit.co/resources/cheatsheets/>

❖ un support complet pour approfondir les notions



[https://dianethy.github.io/cours_R/Introduction_R.h
tml](https://dianethy.github.io/cours_R/Introduction_R.html)

La programmation, tout un concept :

- **Définition** : ensemble des activités qui permettent l'écriture de programmes informatiques.
- **Synonymes** : codage, développement.
- **Contexte** : révolution informatique, 2.5 trillions d'octets de données créés chaque jour.
- **Utilité** : obtenir des informations et connaissances à partir de données.
- **Usage** : de nombreux langages existent (HTML, Javascript, Java, C, Python, R...)

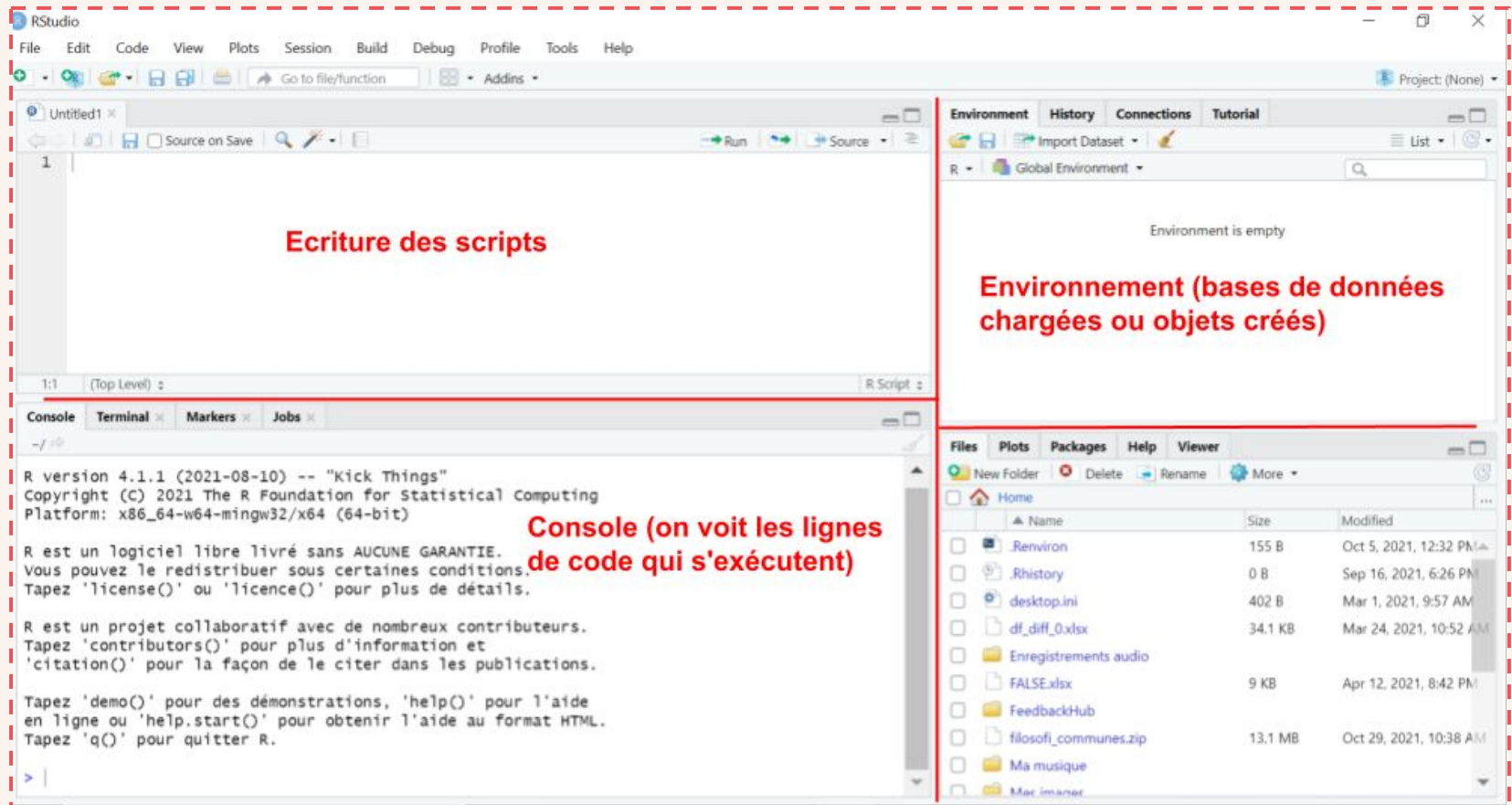
R, un langage de programmation **destiné aux statistiques**

If statistics programs/languages were cars...



Gratuit, puissant, design, avec une communauté importante ([Stack Overflow](#), [R-bloggers](#) etc.)

RStudio, une interface permettant d'interagir avec R



Les concepts clés :


漢

- run
- package
- library
- script

Abc

- **exécuter** (une ligne de code)
- **ensemble de fonctions**
- **endroit où sont gardés les packages**
- **document où l'on code** (extension *.R*)

Assez parlé, **ouvrons RStudio !**

- **Créer un nouveau script** via *File > New File > R script*
- **Commencer à interagir par de simples opérations arithmétiques:** $2+3$
- **Assigner des valeurs à des objets:**
 - $x \leftarrow 2$ (ou $x = 2$ mais la bonne pratique est d'utiliser ' \leftarrow ')
○ $y \leftarrow 7$
○ $resultat \leftarrow x * y$
○ $resultat$
- **Exécuter les lignes de code:** *Ctrl/Enter* ou bouton 
- **Commenter le code** via le `#`

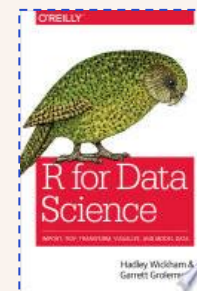
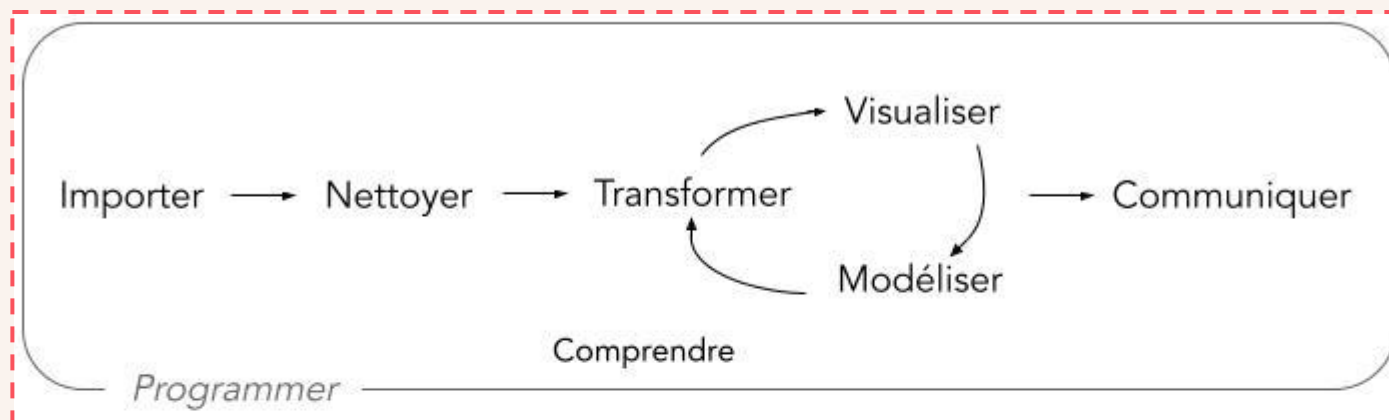
Le tidyverse



Installation des packages

- **Stockage des packages R sur le CRAN**
 - `install.packages("readxl")`
 - `install.packages("tidyverse")`

Schéma d'exploitation des données



[source](#)

Deux manières de coder en R

- **R-base**

- exemple 1 : `exemple[,2]`
- exemple 2 : `f(g(h(exemple)))`

- **Tidyverse**

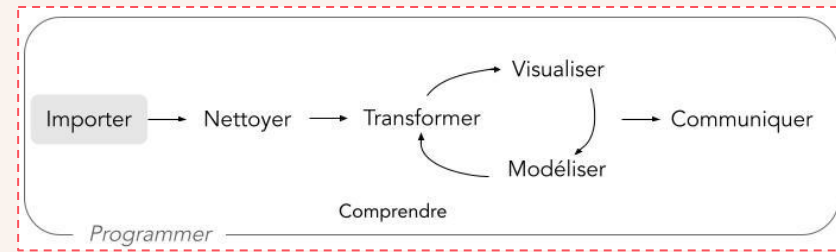
- exemple 1 : `exemple |> select(2)`
- exemple 2 : `exemple |> h() |> g() |> f()`



Le tidyverse fonctionne grâce au pipe (`|>`), qui permet d'appliquer des fonctions à une base de données. L'opération à droite du pipe est appliquée à la valeur située à gauche du pipe, il devient alors possible d'enchaîner les traitements.

Codons ensemble avec les principales fonctions du tidyverse

Importer des données



- **Excel**
 - `library(readxl)`
 - `exemple <- read_excel(path = "fichier.xlsx", sheet = "nom_feuille")`
- **CSV : valeurs séparées par des virgules**
 - `library(readr)`
 - `exemple <- read_csv("fichier.csv")`
- **SCSV : valeurs séparées par des points-virgules (“faux” CSV)**
 - `library(readr)`
 - `exemple <- read_delim("fichier.csv", delim = ";")`
- **À vous !** Importer le jeu de données ‘*Prénoms donnés par département - Millésimé - France*’ disponible sur OpenDataSoft :
 - au format CSV en utilisant le lien [**seulement en 1900 !!**]
 - si vous avez fini, au format tableur Excel en exportant le fichier [**seulement en 1900 !!**]

Importer des données

Solution

- Importer le jeu de données '**Prénoms donnés par département - Millésimé - France**' disponible sur OpenDataSoft :
 - en CSV en utilisant le lien

```
library(tidyverse)
data <-
read_delim("https://public.opendatasoft.com/api/explore/v2.1/catalog/datasets/demographyref-france-prenoms-departement-millesime/exports/csv?lang=fr&refine=birth_year%3A%221900%22&facet=facet(name%3D%22birth_year%22%2C%20disjunctive%3Dtrue)&timezone=Europe%2FBerlin&use_labels=true&delimiter=%3B",
, delim = ";")
```

- en excel en exportant le fichier

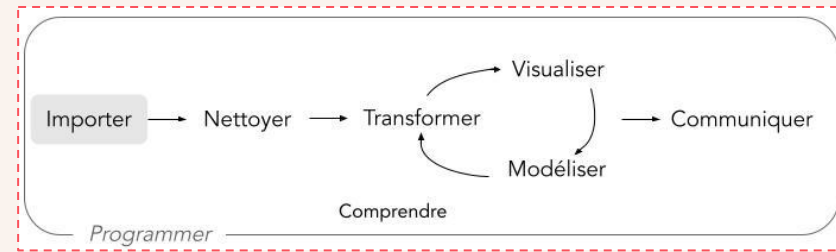
```
library(readxl)
data <-
read_excel("~/Downloads/demographyref-france-prenoms-departement-millesime.xlsx")
```


Séance n°2

Déjà vu ensemble :

1. **La programmation, R et RStudio**, *comprendre les principes de la programmation, et les spécialités du langage R et de l'interface RStudio ;*
2. **Importer des données : `read_csv()`, `read_delim()`, `read_excel()` ;**

Importer des données



- **Excel**
 - `library(readxl)`
 - `exemple <- read_excel(path = "fichier.xlsx", sheet = "nom_feuille")`
- **CSV : valeurs séparées par des virgules**
 - `library(readr)`
 - `exemple <- read_csv("fichier.csv")`
- **SCSV : valeurs séparées par des points-virgules (“faux” CSV)**
 - `library(readr)`
 - `exemple <- read_delim("fichier.csv", delim = ";")`
- **À vous !** Importer le jeu de données ‘*Prénoms donnés par département - Millésimé - France*’ disponible sur OpenDataSoft :
 - au format CSV en utilisant le lien [**seulement en 1900 !!**]
 - si vous avez fini, au format tableur Excel en exportant le fichier [**seulement en 1900 !!**]

Importer des données

Solution

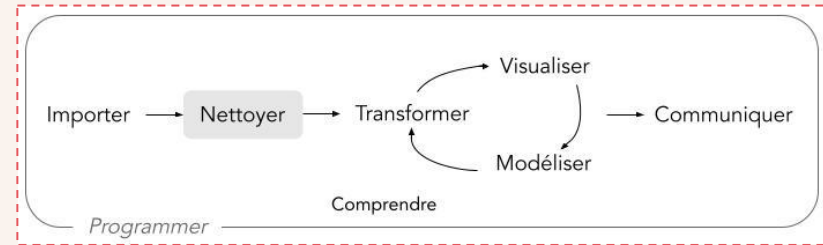
- Importer le jeu de données '**Prénoms donnés par département - Millésimé - France**' disponible sur OpenDataSoft :
 - en CSV en utilisant le lien

```
library(tidyverse)
data <-
read_delim("https://public.opendatasoft.com/api/explore/v2.1/catalog/datasets/demographyref-france-prenoms-departement-millesime/exports/csv?lang=fr&refine=birth_year%3A%221900%22&facet=facet(name%3D%22birth_year%22%2C%20disjunctive%3Dtrue)&timezone=Europe%2FBerlin&use_labels=true&delimiter=%3B",
, delim = ";")
```

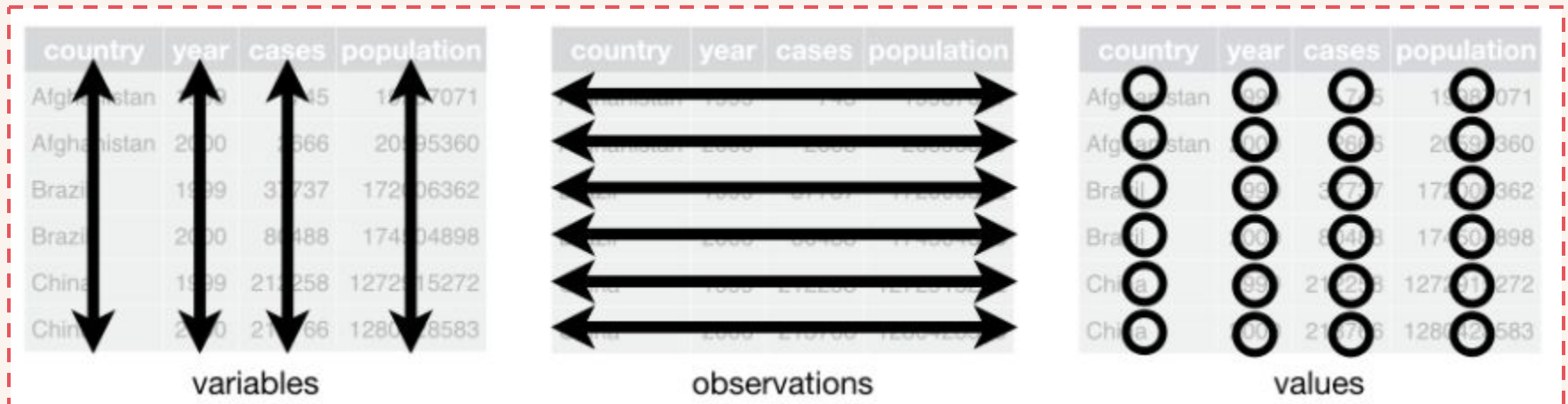
- en excel en exportant le fichier

```
library(readxl)
data <-
read_excel("~/Downloads/demographyref-france-prenoms-departement-millesime.xlsx")
```

Nettoyer des données



- **Structure de données propres**



- chaque ligne est une observation (entrée, enregistrement)
- chaque colonne est une variable
- une seule valeur dans chaque cellule

Nettoyer des données

- À vous !

- Quel jeu de données ci-dessous est proprement structuré ?
- Qu'est-ce qui ne va pas dans les 2 autres jeux ?

country	year	rate
Afghanistan	1999	745/19987071
Afghanistan	2000	2666/20595360
Brazil	1999	37737/172006362
Brazil	2000	80488/174504898
China	1999	212258/1272915272
China	2000	213766/1280428583

A.

country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

B.

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

C.

Nettoyer des données

Solution

- Quel jeu de données ci-dessous est proprement structuré ?
 - Le C.
- Qu'est-ce qui ne va pas dans les 2 autres jeux ?
 - Dans le A. on trouve 2 valeurs dans 1 case
 - Dans le B. les valeurs '*cases*' et '*population*' ne sont pas dans 2 colonnes séparées

Nettoyer des données

Avant de commencer toute analyse, il est nécessaire d'observer les données pour détecter les anomalies

- **Types de données**

Famille de données	Structure	Nom retourné sous R	Exemple
quantitatif	entier	int = integer	3
	décimal inexact	dbl = double	3.4
	décimal exact	num = numeric	3.400001
qualitatif	chaîne de caractères	chr = character	Ville de Paris
	facteur à n niveaux	Factor	petit / moyen / grand
	facteur à 2 niveaux dit "booléen"	bool = boolean	0 / 1, True / False
autre	date	POSIX	13-12-1998

- **Les fonctions pour observer les données**

- ***glimpse(data)*** : structure des données
- ***names(data)*** : nom des colonnes
- ***summary(data\$column)*** : résumé des valeurs
- ***table(data\$column)*** : occurrences des valeurs

- **À vous !** Répondre aux questions suivantes :

- le typage de la variable 'Code Officiel Région' est-il correct ?
- quel est le nombre de naissances maximal pour un prénom ?
- combien de naissances ont été enregistrées en Bretagne ?

Nettoyer des données

Solution

- le typage de la variable '*Code Officiel Région*' est-il correct ?

Le code officiel de la région correspond à un identifiant et non à un nombre à proprement parler (somme, taux, note etc.), donc doit être considéré comme une chaîne de caractères, qui correspond au format '*character*' sous R. Avec la fonction `glimpse(data)` on voit qu'il est considéré comme une chaîne de caractères ('*chr*'), donc son typage est **correct**.

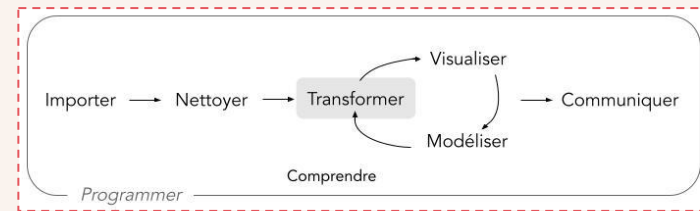
- quel est le nombre de naissances maximal pour un prénom ?

La fonction `summary(data$`Nombre de naissances`)` nous informe que le maximum est de **2519**.

- combien de naissances ont été enregistrées en Bretagne ?

La fonction `table(data$`Nom Officiel Région`)` nous informe que **757** naissances ont été enregistrées en Bretagne.

Transformer des données



Les 7 principales fonctions du **dplyr**, package de transformation de données contenu dans le **tidyverse**, sont les suivantes :

- **rename()** : pour renommer des colonnes
- **select()** et **relocate()** : pour sélectionner ou supprimer certaines colonnes, et/ou changer leur ordre
- **filter()** : pour sélectionner certaines lignes selon une condition
- **mutate()** : pour créer de nouvelles variables
- **summarise()** : pour résumer plusieurs valeurs en 1 valeur
- **group_by()** : pour grouper les observations avant d'appliquer une fonction

Transformer des données : `rename()`

Cette fonction permet de renommer une ou plusieurs colonnes d'une base de données

- **Paramètres :**

- `names(data)` pour connaître le nom des colonnes des données
- `data <- data |> rename(nouveau_nom = `ancien nom`,
nv_nom = ancien_nom)`

- **Exemple :**

- `exemple <- exemple |> rename(tx_chomage = `Taux de chômage`,
regions = régions)`

- **À vous !**

- renommer la colonne 'Année de naissance' par 'annee_naissance'
- renommer la colonne 'Code Officiel Département' par 'code_departement'
- renommer la colonne 'Nom Officiel Département' par 'nom_departement'
- renommer la colonne 'Nombre de naissances' par 'nb_naissances'

Transformer des données : `rename()`

Solution

- renommer la colonne 'Année de naissance' par 'annee_naissance'
- renommer la colonne 'Code Officiel Département' par 'code_departement'
- renommer la colonne 'Nom Officiel Département' par 'nom_departement'
- renommer la colonne 'Nombre de naissances' par 'nb_naissances'

```
data <- data |>
  rename(annee_naissance = `Année de naissance`,
         code_departement = `Code Officiel Département`,
         nom_departement = `Nom Officiel Département`,
         nb_naissances = `Nombre de naissances`)
```

Séance n°4

Déjà vu ensemble :

1. **La programmation, R et RStudio**, *comprendre les principes de la programmation, et les spécialités du langage R et de l'interface RStudio ;*
2. **Importer des données** : `read_csv()`, `read_delim()`, `read_excel()` ;
3. **Explorer des données** : `glimpse()`, `names()`, `summary()`, `table()` ;
4. **Transformer des données avec le tidyverse** :
 - `rename()` pour renommer des colonnes

Transformer des données : `select()` et `relocate()`

La fonction `select()` permet de **sélectionner** certaines colonnes, par leur nom ou leur position dans la base de données.

La fonction `relocate()` permet de **changer l'ordre** des colonnes, par leur nom ou leur position dans la base de données.

- **Paramètres :**

- `data <- data |> select(col1, col3, col2, col4)`
- `data <- data |> relocate(col4, .after = col1)`

- **Exemples :**

- `exemple <- exemple |> select(-c(column2, column4))`
- `exemple <- exemple |> select(1:3, 7, 4:6)`
- `exemple <- exemple |> relocate(col)`

- **À vous !**

- **créer un nouvel objet** sans les variables *'annee_naissance'* et *'Année triable'*
- **créer un nouvel objet** avec toutes les variables de *'Sexe'* à *'nb_naissances'*
- **créer un nouvel objet** où *'code_departement'* est la première variable, en gardant les autres colonnes telles quelles

Transformer des données : `select()` et `relocate()`

Solution

- **créer un nouvel objet** sans les variables *'annee_naissance'* et *'Année triable'*

```
dat_select1 <- data |> select(-annee_naissance, -`Année triable`)  
dat_select1 <- data |> select(-c(annee_naissance, `Année triable`))
```

- **créer un nouvel objet** avec toutes les variables de *'Sexe'* à *'nb_naissances'*

```
dat_select2 <- data |> select(Sexe:nb_naissances)
```

- **créer un nouvel objet** où *'code_departement'* est la première variable, en gardant les autres colonnes telles quelles

```
dat_relocate1 <- data |> relocate(code_departement)
```

ou

```
dat_select3 <- data |> select(code_departement, 1:3, 5:9)
```


Séance n°5

Déjà vu ensemble :

1. **La programmation, R et RStudio**, *comprendre les principes de la programmation, et les spécialités du langage R et de l'interface RStudio ;*
2. **Importer des données** : `read_csv()`, `read_delim()`, `read_excel()` ;
3. **Explorer des données** : `glimpse()`, `names()`, `summary()`, `table()` ;
4. **Transformer des données avec le tidyverse** :
 - `rename()` pour renommer des colonnes ;
 - `select()` et `relocate()` pour sélectionner ou supprimer certaines colonnes, et/ou changer leur ordre ;

Transformer des données : `filter()`

Cette fonction sert à sélectionner des lignes dans une base de données, répondant à certains critères ou conditions logiques

- **Paramètres :**

- `data <- data |> filter(col1 == "valeur")`
- opérations possibles : `==`, `!=`, `>`, `<`, `>=`, `<=`
- combinaisons d'opérations : `&` pour "et", `|` pour "ou"
- utiliser `nrow()` pour compter le nombre de ligne filtrées

- **Exemples :**

- `exemple <- exemple |> filter(col1 != 10 & col3 == "red")`
- `exemple |> filter(col > 0 | col1 <= 100) |> nrow()`

- **À vous !**

- **dans un nouvel objet**, filtrer les prénoms féminins
- **dans un nouvel objet**, filtrer les départements Aisne et Gironde
- filtrer les prénoms ayant été donnés au moins 100 fois en Occitanie et compter le nombre de lignes

Transformer des données : `filter()`

Solution

- dans un nouvel objet, filtrer les prénoms féminins

```
dat_filter1 <- data |> filter(Sexe == "Féminin")
```

- dans un nouvel objet, filtrer les départements Aisne et Gironde

```
dat_filter2 <- data |> filter(nom_departement == "Aisne" |  
                             nom_departement == "Gironde")
```

- filtrer les prénoms ayant été donnés au moins 100 fois en Occitanie et compter le nombre de lignes

```
data |> filter(nb_naissances >= 100 & `Nom Officiel Région` ==  
"Occitanie") |> nrow()
```

Séance n°6

Transformer des données : `mutate()`

Cette fonction permet de créer de nouvelles variables à partir des variables existantes, elle est très utile pour l'analyse de données

- **Paramètres :**

- `data <- data |> mutate(new_col = col1 * col2)`
- statistiques calculables : `mean`, `median`, `min`, `max`, `range`, `sum`, `n`

- **Exemples :**

- `exemple <- exemple |> mutate(new_col = col1 / col2 * 100)`
- `exemple <- exemple |> mutate(new_col2 = col3 - 52)`

- **À vous !**

- créer une **nouvelle variable** du nombre de naissances * 10
- créer une **nouvelle variable** du siècle, extrait de l'année triable
- créer une **nouvelle variable** de la part de chaque naissance (chaque ligne) sur toutes les naissances de 1900
- en + : extraire le premier mot du nom de région
- en + : créer une **nouvelle variable** du sexe formaté différemment : 2 catégories ; Femme, Homme [`str_replace_all()`, `case_when()`]

Transformer des données : `mutate()`

Solution

- créer une **nouvelle variable** du nombre de naissances * 10

```
data <- data |> mutate(nb_naissances10 = nb_naissances * 10)
```

- créer une **nouvelle variable** du siècle, extrait de l'année triable

```
data <- data |>  
  mutate(siecle = substr(`Année triable`, 1, 2),  
         siecle = as.numeric(siecle) + 1)
```

- créer une **nouvelle variable** de la part de chaque naissance (chaque ligne) sur toutes les naissances de 1900

```
data <- data |> mutate(taux = nb_naissances /  
  sum(nb_naissances) * 100)
```

- en + : extraire le premier mot du nom de région

```
data |> mutate(region_1 = word(`Nom Officiel Région`, 1))  
ou data |> mutate(region_1 = str_extract(`Nom Officiel  
Région`, "[a-zA-Z0-9_]*"))
```

Transformer des données : `mutate()`

Solution

- en + : créer une **nouvelle variable** du sexe formaté différemment : 2 catégories ; Femme, Homme [*str_replace_all()*, *case_when()*]

```
data <- data |>
  mutate(new_col = str_replace_all(Sexe,
                                    c("Masculin" = "Homme",
                                      "Féminin" ~ "Femme")))
```

ou

```
data <- data |>
  mutate(new_col = case_when(Sexe == "Masculin" ~ "Homme",
                             Sexe == "Féminin" ~ "Femme",
                             .default = NA_character_))
```


Séance n°7

Transformer des données : `summarise()`

Cette fonction permet de résumer plusieurs valeurs en une seule, ce qui est très utile pour avoir quelques statistiques sur les bases de données

- **Paramètres :**

- `data |> summarise(moyenne = mean(col1))`
- statistiques possibles : `mean`, `median`, `min`, `max`, `range`, `sum`, `n`
- si au moins une valeur manque (`NA`), ajouter l'argument `na.rm = TRUE`

- **Exemple :**

- `exemple |> summarise(min_1 = min(col1, na.rm = TRUE),
max_1 = max(col1, na.rm = TRUE))`

- **À vous !**

- calculer le nombre de naissances moyen et médian
- calculer l'étendue du nombre de naissances

Transformer des données : `summarise()`

Solution

- calculer le nombre de naissances moyen et médian

```
data |> summarise(moy = mean(nb_naissances),  
                  med = median(nb_naissances))
```

- calculer l'étendue du nombre de naissances

```
data |> summarise(etendue = range(nb_naissances))
```

Séance n°8

Transformer des données : `group_by()`

Cette fonction permet de grouper les variables, elle est spécialement utile en analyse de données pour calculer des statistiques par groupe. Depuis janvier 2023, l'argument `.by` peut-être ajouté directement **dans** les fonctions du `dyplr`.

- **Paramètres :**

- ancienne manière de coder : `data |> group_by(group) |> summarise(min = min(col)) |> ungroup()`
- nouvelle manière de coder :
`data |> summarise(min = min(col), .by = group)`
- combiner avec les opérations déjà présentées : `mean`, `median`, `min`, etc

- **Exemple :**

- `exemple |> mutate(nb_lignes = n(), .by = group_col)`

- **À vous !**

- calculer le nombre de naissances par sexe
- calculer **dans une nouvelle variable de la base**, le nombre de prénoms (lignes) par région
- garder uniquement la première ligne de chaque région

Transformer des données : `group_by()`

Solution

- calculer le nombre de naissances par sexe

```
data |> summarise(naissances_sexe = sum(nb_naissances), .by = Sexe)
```

- calculer **dans une nouvelle variable de la base**, le nombre de prénoms (lignes) par région

```
data |> mutate(n_reg = n(), .by = `Nom Officiel Région`)
```

- garder uniquement la première ligne de chaque région

```
data |> filter(row_number() == 1, .by = `Nom Officiel Région`)
```

ou

```
data |> slice(1, .by = `Nom Officiel Région`)
```

Séance n°9

Joindre des données :

a		b	
x1	x2	x1	x3
A	1	A	T
B	2	B	F
C	3	D	T

→ Jointures

Il y a 4 types de jointures applicables dès lors qu'une variable (colonne) est commune aux 2 df : cela peut être utile par exemple pour ajouter des informations d'une base extérieure (*bases nationales telles que Sirene, Prénoms, DECP...*) à une base existante.

Il est nécessaire d'avoir une variable de jointure, dite "**clé**" (*identifiants, numéros de SIRET, noms de pays...*) qui permettra d'unir les 2 bases, elle se spécifie dans l'argument 'by='. Ici, la clé est `x1` qui est présente dans **a** comme dans **b**.

x1	x2	x3
A	1	T
B	2	F
C	3	NA

`left_join(a, b, by = "x1")` : ajoute les lignes communes de **b** à **a**. Cette fonction est la **plus utilisée** pour ajouter des informations extérieures à une base de données existante.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

`right_join(a, b, by = "x1")` : ajoute les lignes communes de **a** à **b**

x1	x2	x3
A	1	T
B	2	F

`inner_join(a, b, by = "x1")` : joint en ne gardant que les lignes communes aux 2 df

x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

`full_join(a, b, by = "x1")` : joint les données en gardant toutes les valeurs

Joindre des données : `left_join()`

Cette fonction permet de joindre une base de données avec une autre, elle est particulièrement utile pour récupérer des variables de bases extérieures pour enrichir les données actuelles.

- **Paramètres :**
 - `left_join(data1, data2, by = "colonne_jointure")`
- **Exemple :**
 - `exemple |> left_join(referentiel, by = c("id" = "ID"))`
- **À vous !**
 - joindre les données de naissances avec les données de structure de la population active par département
 - ne garder que l'année 2019
 - ne garder que la valeur pour les "Femmes Actifs ayant un emploi"
 - importer les données sur R, les mettre en forme
 - enrichir les données de naissance par le nombre de femmes actives par département

Joindre des données : `left_join()`

Solution

- À vous !
 - joindre les données de naissances avec les données de structure de la population active par département
 - ne garder que l'année 2019
 - ne garder que la valeur pour les "Femmes Actifs ayant un emploi"
 - importer les données sur R, les mettre en forme

```
pop_departements <-  
read_delim("https://public.opendatasoft.com/api/explore/v2.1/catalog/datasets/demographyref-france-pop-active-sexe-activite-departement-millesime/exports/csv?lang=fr&refine=year%3A%222019%22&facet=facet(name%3D%22year%22%2C%20disjunctive%3Dtrue)&timezone=Europe%2FBerlin&use_labels=true&delimiter=%3B", ";") |>  
  filter(`Nom de la variable` == "Femmes Actifs ayant un  
emploi") |>  
  select(`Code Officiel Département`, `Valeur`) |>  
  rename(pop_femmes_actives = Valeur)
```

Joindre des données : `left_join()`

Solution

- À vous !
 - joindre les données de naissances avec les données de structure de la population active par département
 - enrichir les données de naissance par le nombre de femmes actives par département

```
jointure <- data |>  
  left_join(pop_departements, by = c("code_departement" =  
  "Code Officiel Département"))
```

