# Traitement de données avec le package Tidyverse

**Malick SENE** **Kerencia Dyvana PAHANE SEUNKAM**

**ENSAE**

**19 avril 2025**

# Qu'est ce que le tidyverse ?

- Écosystème de packages R pour la science des données moderne
- Porté par Hadley Wickham et l'équipe Posit Software.
- Offre une grammaire unifiée pour manipulation, analyse et visualisation

# Les packages phares

| Package | Usage principal |
|---|---|
| `ggplot2` | **Graphiques déclaratifs** |
| `dplyr` | **Transformation de tableaux** |
| `tidyr` | **Restructuration (spread/gather)** |
| `readr` | **Import rapide de fichiers plats** |
| `tibble` | **Data.frames modernisés** |
| `stringr` | **Manipulation de chaînes** |
| `purrr` | **Programmation fonctionnelle sur listes** |

**(Extensions :** *lubridate*, *forcats*, *readxl*, *haven*, **...)**

# Plan de la présentation

1. **Import/export de données**
2. **Manipulation avec** `dplyr`
3. **Visualisation avec** `ggplot2`
4. **Recodage de variable (**`tidyr`, `forcats`**)**
5. **Texte & chaînes (**`stringr`**)**
6. **Automatisation (**`purrr`**)**

# Importation et exportation des données

- **Sources : fichiers plats (CSV, TSV), Excel, SAS/SPSS/Stata, dBase, bases SQL**
- **Packages :** `readr`, `readxl`, `haven`, `foreign`, `DBI/RSQLite`, `feather`

## Import de fichiers texte

```
# CSV / TSV
df_csv <- readr::read_csv("fichier.csv")
df_tsv <- readr::read_tsv("fichier.tsv")
df_csv2 <- readr::read_csv2("fichier_semicolon.csv")

# Séparateurs personnalisés
df_delim <- readr::read_delim("fichier.txt", delim = ";")

# Encodage
df_enc1 <- readr::read_csv("fichier.csv", locale = locale(enc
df_enc2 <- readr::read_csv("fichier.csv", locale = locale(enc
```

# Importation et exportation des données

## Import de fichiers Excel

```
df_xl <- readxl::read_excel("fichier.xlsx")
df_xl2 <- readxl::read_excel("fichier.xlsx", sheet = "Feuille2
    range = "C1:F124")
readxl::excel_sheets("fichier.xlsx")
```

## Import de formats statistiques

```
# SPSS
df_sav <- haven::read_sav("fichier.sav")
df_por <- haven::read_por("fichier.por")
# Stata
df_dta <- haven::read_dta("fichier.dta")
# SAS
df_sas <- haven::read_sas("fichier.sas7bdat")
df_xpt <- haven::read_xpt("fichier.xpt")
```

# Importation et exportation des données

## Import dBase & bases SQL

```r
# dBase
df_dbf <- foreign::read.dbf("fichier.dbf", as.is = TRUE)

# SQLite (DBI + RSQLite)
con <- DBI::dbConnect(RSQLite::SQLite(), "ma_base.sqlite")
DBI::dbListTables(con)
df_sql <- DBI::dbReadTable(con, "ma_table")
DBI::dbDisconnect(con)
```

# Importation et exportation des données

## Export de données

```
# Fichiers plats
readr::write_csv(df, "data.csv")
readr::write_tsv(df, "data.tsv")
readr::write_csv2(df, "data_semicolon.csv")
# Formats statistiques
haven::write_sav(df, "data.sav")
haven::write_dta(df, "data.dta")
haven::write_sas(df, "data.sas7bdat")
```

# Manipulation de données avec dplyr

- **fonctions clés :** `filter()`, `select()`, `mutate()`, `arrange()`, `slice()`, `group_by()`, `summarise()`, `join()`
- **Syntaxe chaînée via %>% (ou |>), claire et performante**
- **Optimisé pour tibbles et gros volumes**

## Sélection et filtrage

- **slice() : lignes par position**
- **filter() : lignes selon condition logique**

```
wf %>%
    slice(1:5)
wf %>%
    filter(hhsize > 5)
```

# Manipulation de données avec dplyr

## Colonnes : select, rename, arrange

- **select() : choisir/éliminer/réorganiser des colonnes**
- **rename() : renommer**
- **arrange() : trier**

```
wf %>%
    select(country, hactiv12m)
wf %>%
    rename(country_21 = country)
wf %>%
    arrange(desc(hage), hhsize)
```

# Manipulation de données avec dplyr

## Création et calcul : mutate & pipes

- **mutate() : nouvelles colonnes**
- **enchaînement des fonctions**

```
wf %>%
    mutate(depense_par_eq = dtot/eqadu1) %>%
    filter(hage > 60) %>%
    select(hhid, hage, depense_par_eq)
```

# Manipulation de données avec dplyr

## Regroupements et agrégation

- group_by() + summarise() / count()
- ungroup() pour revenir à l'état "plat"

```
wf %>%
    group_by(region) %>%
    summarise(age_moyen = mean(hage, na.rm = TRUE), nb = n())
    ungroup()

wf %>%
    count(hgender, sort = TRUE)
```

# Manipulation de données avec dplyr

## Joins et fusion de tables

- **inner_join(), left_join(), right_join(), full_join()**
- **Préparer clés, types et doublons avant de joindre**

```
inner_join(wf_18, wf_21, by = "hhid")
left_join(individu, menage, by = "hhid")
```

# Visualisation avec ggplot2

## Principes de base

- **Grammaire des graphiques : construction par couches**
- `ggplot(data)` **: initialisation du graphique avec la source de données**
- `+ geom_*()` **: ajout d'éléments graphiques (points, lignes, barres...)**

## Mappage esthétique (`aes()`)

- **Définit le lien variable → attribut graphique**
  - `aes(x, y, color, size, shape, alpha, fill…)`
- **Dans** `ggplot()` **: mappage global**
- **Dans** `geom_*()` **: mappage spécifique à une couche**

# Visualisation avec ggplot2

## Géométries courantes

- `geom_histogram(aes(x = var))` : **distribution en barres**
- `geom_point(aes(x, y))` : **nuage de points**
- `geom_boxplot(aes(x, y))` : **boîtes à moustaches**
- `geom_violin(aes(x, y))` : **densité + boxplot**
- `geom_bar(aes(x = cat), position = ...)` : **barplot simple, empilé, côte-à-côte, proportions**
- `geom_text()` / `geom_label()` : **annotations textuelles**

## Faceting

- `facet_wrap(~ var)` : **mini graphes pour chaque modalité**
- `facet_grid(row ~ col)` : **grille croisant deux variables**

# Visualisation avec ggplot2

## Scales et axes

- `scale_x_continuous()` / `scale_y_continuous()`
  - `name`, `breaks`, `limits`
- `scale_x_discrete()` / `scale_y_discrete()`
  - `labels`, `limits` **pour réordonner ou renommer**
- `scale_size()` : **contrôles de la taille des points**
- `scale_color_*()` / `scale_fill_*()`
  - `gradient`, `viridis`, `manual`, `brewer`

## Thèmes

- `theme_minimal()`, `theme_classic()`, `theme_light()`, `theme_void()`
- `theme()` **pour personnaliser :**
  - `plot.title`, `axis.text`, `legend.position`, **marges, fond, etc.**

# Un peu de cartographie

```
# 1. Importer les données spatiales
donnees_vecteur <- st_read("Limite_Région.shp")
# 2. Créer la carte de base
carte <- ggplot(donnees_vecteur) + geom_sf(fill = "lightgrey",
    color = "white") + theme_minimal()
# 3. Ajouter les annotations spatiales
carte <- carte + annotation_north_arrow(location = "tl", which
    style = north_arrow_fancy_orienteering()) + annotation_sca
    width_hint = 0.25)
# 4. Ajouter des étiquettes pour chaque région
carte <- carte + geom_sf_label(aes(label = NOMREG), size = 3,
    color = "black")
# 5. Appliquer une palette de couleurs continue
carte <- carte + scale_fill_viridis_c("Indice thématique", opt
# 6. Afficher la carte
print(carte)
```

# Recodage de variables

## Variables qualitatives et facteurs

**Les variables qualitatives (ou catégorielles) peuvent être de deux types dans R : - des chaînes de caractères (`character`) ; - ou des facteurs (`factor`), qui associent chaque valeur à une modalité (niveau).**

**Les facteurs sont utiles pour : - représenter des catégories limitées ; - éviter les incohérences ou fautes de frappe ; - faciliter certaines analyses statistiques.**

# Recodage de variables

## Conversion en facteur

**Lorsqu'on importe des données avec le tidyverse (`readr`, `haven`, etc.), les variables sont souvent de type `character`.**
**On peut les convertir en facteur avec la fonction `as_factor()` du package `haven`.**

## Recoder les modalités d'une variable qualitative

**Utiliser la fonction `fct_recode()` du package forcats pour : - Renommer les modalités avec la syntaxe `"nouveau nom" = "ancien nom"`. - Attention : les correspondances doivent être exactes (accents, espaces...).**
**Utiliser `fct_explicit_na()` pour transformer les `NA` d'un facteur en une modalité visible (ex. `"valeurs manquantes"`).**

# Recodage de variables

## Regrouper des modalités

**La fonction `fct_collapse()` permet de regrouper plusieurs modalités sous un même nom (ex. regrouper types d'entreprises en `"Entreprises"`).**

## Interface graphique de recodage (`questionr`)

**Le package `questionr` propose une interface graphique via le menu Addins de RStudio : - `Levels recoding` ou la fonction `irec()` pour recoder les modalités ; - L'interface génère automatiquement le code R correspondant.**

# Recodage de variables

**Utiliser la fonction `fct_relevel()` pour réordonner manuellement les modalités dans un ordre logique (ex. du diplôme le plus bas au plus élevé).**

**Utiliser `fct_reorder()` pour classer les modalités d'un facteur en fonction des valeurs d'une autre variable (ex. revenu moyen, bien-être).**

## Interface graphique pour ordonner (`questionr`)

**L'Addin Levels ordering ou la fonction `iorder()` permet de modifier l'ordre des modalités via glisser-déposer, et génère le code automatiquement.**

# Recodage de variables

## Créer une nouvelle variable conditionnelle

- `if_else()`

**Permet de créer une variable à deux modalités selon une condition logique simple (ex. âge < 40 → “Jeune”, sinon → “Âgé”).**

- `case_when()`

**Permet d’assigner des valeurs en fonction de plusieurs conditions logiques : - Chaque condition est testée dans l’ordre ; - Il est important d’aller du plus spécifique au plus général pour éviter les erreurs.**

# Recodage de variables

## Découpage automatique

**Avec la fonction `cut()`, on peut découper une variable en un nombre donné de classes égales (`breaks = 5`).**

## Découpage personnalisé

**On peut aussi spécifier manuellement les bornes des classes (ex. `breaks = c(15.9, 25, 35, 45, 55, 65, 97)`) avec l'option `include.lowest = TRUE` pour inclure l'extrémité basse.**

## Interface graphique pour découpage (`questionr`)

**Utiliser l'Addin Numeric range dividing ou la fonction `icut()` pour créer des classes à partir d'une variable numérique.**
**L'interface permet de fixer les bornes, voir le résultat, et obtenir le code R généré.**

# Manipuler du texte avec `stringr`

**L'extension `stringr`, partie intégrante du tidyverse, permet de manipuler des chaînes de caractères de manière simple et cohérente.**

## Objectifs courants :

- **Combiner, rechercher, extraire, remplacer du texte**
- **Utiliser des expressions régulières pour détecter des motifs complexes**

# Manipuler du texte avec `stringr`

## Expressions régulières

**Les expressions régulières permettent de : - Trouver des mots ou structures spécifiques (fin de phrase, majuscule, nombre, email) - Décrire précisément un motif textuel**

**Exemples : -** `\\w+$` **: dernier mot d'une chaîne -** `\\b[A-Z]\\w*` **: mot débutant par une majuscule -** `^\\d{3,4}` **: nombre de 3 ou 4 chiffres au début -** `[\\w.+-]+@[\\w.-]+\\.[a-zA-Z]{2,}` **: adresse email**

# Manipuler du texte avec `stringr`

## Concaténer des chaînes

- `paste()` : **concatène avec un espace par défaut**
- `paste0()` : **concatène sans séparateur**
- `str_c()` : **équivalent tidyverse, plus cohérent avec** `dplyr`
- `collapse = ","` : **permet de coller tous les éléments d'un vecteur entre eux**

# Manipuler du texte avec `stringr`

## Convertir en majuscules / minuscules

- `str_to_lower()` : **tout en minuscules**
- `str_to_upper()` : **tout en majuscules**
- `str_to_title()` : **majuscule au début de chaque mot**

## Découper des chaînes

- `str_split()` : **coupe une chaîne selon un séparateur**
  - **Résultat : une liste ou une matrice (avec** `simplify = TRUE`**)**
- `separate()` **(**`tidyr`**) : crée plusieurs colonnes à partir d'une seule colonne texte**

# Manipuler du texte avec `stringr`

## Extraire des sous-chaînes

- `str_sub()` : **extrait une portion d'une chaîne selon sa position (ex. du 1er au 10e caractère)**

## Détecter un motif

- `str_detect()` : **renvoie** `TRUE/FALSE` **si un motif est présent**
  - **Utile avec** `filter()` **pour extraire des lignes ciblées**
- `str_count()` : **compte le nombre d'occurrences du motif**
- `str_subset()` : **extrait les éléments d'un vecteur qui contiennent un motif**

# Manipuler du texte avec `stringr`

## Extraire un motif

- `str_extract()` : **extrait la première occurrence du motif**
- `str_extract_all()` : **extrait toutes les occurrences**

## Remplacer des motifs

- `str_replace()` : **remplace une seule occurrence**
- `str_replace_all()` : **remplace toutes les occurrences ou plusieurs motifs à la fois (avec une liste nommée)**

# Programmation fonctionnelle avec `purrr`

**La programmation fonctionnelle permet d'appliquer des fonctions à des structures de données (vecteurs, listes, data frames) sans boucle explicite. Le package `purrr`, qui fait partie du `tidyverse`, facilite cette approche.**

# Programmation fonctionnelle avec `purrr`

## `map()`

- **Applique une fonction à chaque élément d'une liste ou d'un vecteur.**
- **Retourne une liste.**
- **Exemple : appliquer `class()` à chaque colonne d'un data frame.**

## `map_dbl()`

- **Variante de `map()` qui retourne un vecteur numérique (`double`).**
- **Idéal pour appliquer des fonctions comme `mean()`, `sum()`, etc., sur des colonnes numériques.**

# Programmation fonctionnelle avec `purrr`

## `map_dfr()`

- **Applique une fonction à chaque élément d'une liste et combine les résultats par ligne dans un data frame.**
- **Utile pour produire des résumés par colonne (moyenne, min, max, etc.).**
- **Le paramètre `.id` permet de conserver le nom ou l'indice de chaque élément.**

## `map2()`

- **Applique une fonction à deux vecteurs ou listes en parallèle.**
- **Très utile pour combiner deux colonnes d'un data frame (ex. produit pondéré entre taille et poids).**
- **Variantes disponibles : `map2_dbl()`, `map2_chr()`, etc.**

# Programmation fonctionnelle avec `purrr`

## `pmap()`

- **Applique une fonction à plusieurs vecteurs (plus de deux) simultanément.**
- **Nécessite une liste de colonnes de même longueur.**
- **Permet de combiner plusieurs variables dans une opération personnalisée.**

## `imap()`

- **Semblable à `map()`, mais fournit aussi l'indice ou le nom de chaque élément.**
- **Permet de générer des chaînes personnalisées ou des résumés nommés.**

# Programmation fonctionnelle avec `purrr`

## `keep()` et `discard()`

- `keep()` : conserve les éléments pour lesquels une condition est vraie.
- `discard()` : supprime les éléments pour lesquels une condition est vraie.
- S'utilisent avec des fonctions prédicats (retournant `TRUE` ou `FALSE`).
- Ex. : filtrer les colonnes avec un taux de valeurs non manquantes ≥ 70 %.

# Merci pour votre attention