KIN6520 – Préparation au Laboratoire I

Introduction à R, tidyverse et Quarto

Jonathan Tremblay

2026-02-01

Table of contents

0.1 Objectif de ce tutoriel

Ce document vous prépare à l'activité en classe du Laboratoire I. Après l'avoir complété, vous serez en mesure de :

- Comprendre la syntaxe de base de R
- Manipuler des données avec le tidyverse
- Créer des graphiques avec ggplot2
- Rédiger des documents reproductibles avec Quarto

> Temps estimé
>
> Environ **60 à 90 minutes** pour compléter ce tutoriel. Prenez le temps d'exécuter chaque bloc de code et d'expérimenter!

---

1 Partie 1 : L'environnement de travail

1.1 Qu'est-ce que R?

R est un langage de programmation conçu pour l'analyse statistique et la visualisation de données. Pensez-y comme une calculatrice extrêmement puissante qui peut :

- Importer et manipuler des jeux de données

- Effectuer des analyses statistiques
- Créer des graphiques de qualité publication
- Automatiser des tâches répétitives

1.2 Qu'est-ce que Positron?

Positron est l'environnement de développement intégré (IDE) que nous utilisons. C'est l'interface graphique qui facilite l'écriture et l'exécution du code R. Ses composantes principales sont :

Zone	Fonction
Éditeur (centre)	Où vous écrivez votre code et vos documents
Console (bas)	Où R exécute les commandes et affiche les résultats
Environnement (droite)	Liste des objets (variables, données) en mémoire
Fichiers/Graphiques (droite)	Navigation dans les fichiers et affichage des figures

1.3 Qu'est-ce que Quarto?

Quarto (extension `.qmd`) est un format de document qui combine texte et code. C'est ce qui nous permet de créer des rapports reproductibles : le même fichier contient l'analyse ET le texte, garantissant que les résultats présentés correspondent exactement au code exécuté.

---

2 Partie 2 : Les bases de R

2.1 Exécuter du code

Dans Positron, vous pouvez exécuter du code de plusieurs façons :

1. **Bloc entier** : Cliquez sur ▸ à droite du bloc de code
2. **Ligne par ligne** : Placez le curseur et appuyez sur `Ctrl+Enter` (Windows) ou `Cmd+Enter` (Mac)
3. **Sélection** : Sélectionnez plusieurs lignes et appuyez sur `Ctrl+Enter`

> Essayez!
>
> Exécutez le bloc suivant pour vérifier que tout fonctionne.

```
# Ceci est un commentaire - R l'ignore
# Les commentaires commencent par #

print("Bienvenue dans R!")
```

```
[1] "Bienvenue dans R!"
```

2.2 Variables et assignation

Une **variable** est un conteneur qui stocke une valeur. En R, on utilise `<-` pour assigner une valeur à une variable.

```r
# Assigner la valeur 75 à la variable "masse"
masse <- 75

# Assigner la valeur 1.80 à la variable "taille"
taille <- 1.80

# Afficher le contenu d'une variable (tapez simplement son nom)
masse
```

```
[1] 75
```

> **i** Convention de nommage
>
> - Utilisez des noms descriptifs : `vo2_max` plutôt que `x`
> - Évitez les accents et les espaces
> - R est sensible à la casse : `Masse` ≠ `masse`

2.3 Opérations de base

R peut effectuer toutes les opérations arithmétiques standard :

```r
# Addition, soustraction, multiplication, division
10 + 5
```

```
[1] 15
```

```r
20 - 8
```

```
[1] 12
```

```r
4 * 7
```

```
[1] 28
```

```r
100 / 4
```

```
[1] 25
```

```r
# Puissance
2^3  # 2 à la puissance 3
```

```
[1] 8
```

```
# Calcul avec des variables
imc <- masse / taille^2
imc
```

```
[1] 23.14815
```

2.4 Fonctions

Une **fonction** est une commande qui effectue une action spécifique. Elle prend des **arguments** en entrée et retourne un **résultat**.

```
# Syntaxe : nom_fonction(argument1, argument2, ...)

# Racine carrée
sqrt(16)
```

```
[1] 4
```

```
# Arrondir à 2 décimales
round(3.14159, digits = 2)
```

```
[1] 3.14
```

```
# Valeur absolue
abs(-5)
```

```
[1] 5
```

```
# Moyenne d'un ensemble de valeurs
mean(c(10, 20, 30, 40, 50))
```

```
[1] 30
```

> **Aide sur une fonction**
>
> Pour obtenir de l'aide sur une fonction, tapez `?nom_fonction` dans la console :
>
> ```
> ?mean
> ```

2.5 Vecteurs

Un **vecteur** est une séquence de valeurs du même type. On le crée avec la fonction `c()` (pour "combiner").

```
# Créer un vecteur de fréquences cardiaques
fc <- c(72, 85, 110, 145, 178, 185)

# Longueur du vecteur
length(fc)
```

```
[1] 6
```

```
# Accéder à un élément (R commence à 1, pas à 0!)
fc[1]   # Premier élément
```

```
[1] 72
```

```
fc[3]   # Troisième élément
```

```
[1] 110
```

```
# Accéder à plusieurs éléments
fc[c(1, 3, 5)]  # 1er, 3e et 5e éléments
```

```
[1]  72 110 178
```

```
fc[2:4]         # Éléments 2 à 4
```

```
[1]  85 110 145
```

2.6 Fonctions sur les vecteurs

Plusieurs fonctions travaillent sur des vecteurs entiers :

```
# Statistiques descriptives
mean(fc)    # Moyenne
```

```
[1] 129.1667
```

```
sd(fc)      # Écart-type
```

```
[1] 47.62107
```

```
min(fc)     # Minimum
```

```
[1] 72
```

```
max(fc)     # Maximum
```

```
[1] 185
```

```
range(fc)   # Min et max ensemble
```

```
[1]  72 185
```

```
# Résumé complet
summary(fc)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  72.00   91.25  127.50  129.17  169.75  185.00 
```

2.7 Le pipe |>

Le **pipe** (|>) permet d'enchaîner des opérations de manière lisible. Il se lit comme "PUIS" ou "ensuite".

```
# Sans pipe (difficile à lire avec plusieurs opérations)
round(sqrt(mean(fc)), 2)
```

```
[1] 11.37
```

```
# Avec pipe (se lit de gauche à droite)
fc |>
  mean() |>
  sqrt() |>
  round(2)
```

```
[1] 11.37
```

! Lecture du pipe

`x |> f()` signifie “prends x, PUIS applique la fonction f”

Le pipe passe automatiquement le résultat précédent comme **premier argument** de la fonction suivante.

---

3 Partie 3 : Le tidyverse

3.1 Qu’est-ce que le tidyverse?

Le **tidyverse** est une collection de packages R conçus pour la science des données. Ces packages partagent une philosophie commune et fonctionnent bien ensemble.

```
# Charger le tidyverse (fait une seule fois au début du document)
library(tidyverse)
```

i Installer un package

Avant de pouvoir utiliser un package avec `library()`, il faut d’abord l’**installer** sur votre ordinateur. L’installation se fait une seule fois, tandis que le chargement avec `library()` doit être fait à chaque nouvelle session R.

```
# Installer un package (une seule fois)
install.packages("tidyverse")

# Installer plusieurs packages à la fois
install.packages(c("tidyverse", "readxl"))
```

Dans Positron, vous pouvez aussi cliquer sur l’onglet **Packages** dans le panneau de droite, puis sur **Install** pour installer un package via l’interface graphique.

Les principaux packages du tidyverse que nous utiliserons :

Package	Fonction
readr	Importer des données (CSV, etc.)
readxl	Importer des fichiers Excel
dplyr	Manipuler des tableaux de données
ggplot2	Créer des graphiques
tibble	Tableaux de données modernes

3.2 Les tableaux de données (data frames)

Un **data frame** (ou **tibble** dans le tidyverse) est un tableau où :

- Chaque **colonne** est une variable
- Chaque **ligne** est une observation

```
# Créer un petit tableau d'exemple
donnees_test <- tibble(
  temps_s = c(0, 60, 120, 180, 240, 300),
  vo2_ml = c(350, 890, 1250, 1800, 2300, 2450),
  fc_bpm = c(72, 95, 120, 145, 168, 182),
  puissance_w = c(0, 50, 100, 150, 200, 250)
)

# Afficher le tableau
donnees_test
```

```
# A tibble: 6 × 4
  temps_s vo2_ml fc_bpm puissance_w
    <dbl>  <dbl>  <dbl>       <dbl>
1       0    350     72           0
2      60    890     95          50
3     120   1250    120         100
4     180   1800    145         150
5     240   2300    168         200
6     300   2450    182         250
```

3.3 Importer des données avec readr

La fonction `read_csv()` importe des fichiers CSV (valeurs séparées par des virgules) :

```
# Syntaxe de base
data <- read_csv("chemin/vers/fichier.csv")

# En classe, nous utiliserons :
data <- read_csv("donnees_vo2max.csv")
```

3.4 Importer des données Excel avec readxl

Le package **readxl** permet d'importer des fichiers Excel (`.xlsx` ou `.xls`) directement dans R, sans avoir besoin de les convertir en CSV.

```
# Charger le package readxl (une seule fois)
library(readxl)

# Importer un fichier Excel
data <- read_excel("chemin/vers/fichier.xlsx")

# Spécifier une feuille particulière (par nom ou numéro)
data <- read_excel("fichier.xlsx", sheet = "Feuille1")
data <- read_excel("fichier.xlsx", sheet = 2)  # Deuxième feuille

# Spécifier une plage de cellules
```

```r
data <- read_excel("fichier.xlsx", range = "A1:D50")

# Ignorer les premières lignes (ex: si l'en-tête n'est pas à la ligne 1)
data <- read_excel("fichier.xlsx", skip = 2)
```

Différences entre read_csv() et read_excel()

Aspect	`read_csv()`	`read_excel()`
Format	Fichiers `.csv`	Fichiers `.xlsx`, `.xls`
Package	readr (inclus dans tidyverse)	readxl (à charger séparément)
Feuilles	Un seul tableau	Peut avoir plusieurs feuilles
Formules	Non applicable	Lit les valeurs, pas les formules

Voir les feuilles disponibles

Pour connaître les noms des feuilles d'un fichier Excel :

```r
excel_sheets("fichier.xlsx")
```

3.5 Explorer les données

Plusieurs fonctions permettent d'examiner un tableau :

```r
# Aperçu de la structure (colonnes, types, premières valeurs)
glimpse(donnees_test)
```

```
Rows: 6
Columns: 4
$ temps_s     <dbl> 0, 60, 120, 180, 240, 300
$ vo2_ml      <dbl> 350, 890, 1250, 1800, 2300, 2450
$ fc_bpm      <dbl> 72, 95, 120, 145, 168, 182
$ puissance_w <dbl> 0, 50, 100, 150, 200, 250
```

```r
# Premières lignes
head(donnees_test)
```

```
# A tibble: 6 × 4
  temps_s vo2_ml fc_bpm puissance_w
    <dbl>  <dbl>  <dbl>       <dbl>
1       0    350     72           0
2      60    890     95          50
3     120   1250    120         100
```

```
4     180   1800    145         150
5     240   2300    168         200
6     300   2450    182         250
```

```
# Dernières lignes
tail(donnees_test)
```

```
# A tibble: 6 × 4
  temps_s vo2_ml fc_bpm puissance_w
    <dbl>  <dbl>  <dbl>       <dbl>
1       0    350     72           0
2      60    890     95          50
3     120   1250    120         100
4     180   1800    145         150
5     240   2300    168         200
6     300   2450    182         250
```

```
# Dimensions (lignes x colonnes)
dim(donnees_test)
```

```
[1] 6 4
```

```
# Noms des colonnes
names(donnees_test)
```

```
[1] "temps_s"     "vo2_ml"      "fc_bpm"      "puissance_w"
```

```
# Résumé statistique
summary(donnees_test)
```

```
    temps_s        vo2_ml         fc_bpm       puissance_w
 Min.   :  0   Min.   : 350   Min.   : 72.0   Min.   :  0.0
 1st Qu.: 75   1st Qu.: 980   1st Qu.:101.2   1st Qu.: 62.5
 Median :150   Median :1525   Median :132.5   Median :125.0
 Mean   :150   Mean   :1507   Mean   :130.3   Mean   :125.0
 3rd Qu.:225   3rd Qu.:2175   3rd Qu.:162.2   3rd Qu.:187.5
 Max.   :300   Max.   :2450   Max.   :182.0   Max.   :250.0
```

3.6 Manipuler les données avec dplyr

Le package **dplyr** fournit des “verbes” pour manipuler les données. Les plus courants sont :

3.6.a `select()` – Choisir des colonnes

```r
# Garder seulement certaines colonnes
donnees_test |>
  select(temps_s, vo2_ml)
```

```
# A tibble: 6 × 2
  temps_s vo2_ml
    <dbl>  <dbl>
1       0    350
2      60    890
3     120   1250
4     180   1800
5     240   2300
6     300   2450
```

```r
# Exclure une colonne
donnees_test |>
  select(-puissance_w)
```

```
# A tibble: 6 × 3
  temps_s vo2_ml fc_bpm
    <dbl>  <dbl>  <dbl>
1       0    350     72
2      60    890     95
3     120   1250    120
4     180   1800    145
5     240   2300    168
6     300   2450    182
```

3.6.b `filter()` – Filtrer des lignes

```r
# Garder les lignes où la puissance > 100 W
donnees_test |>
  filter(puissance_w > 100)
```

```
# A tibble: 3 × 4
  temps_s vo2_ml fc_bpm puissance_w
    <dbl>  <dbl>  <dbl>       <dbl>
1     180   1800    145         150
2     240   2300    168         200
3     300   2450    182         250
```

```r
# Plusieurs conditions (ET)
donnees_test |>
  filter(puissance_w > 100, fc_bpm < 170)
```

```
# A tibble: 2 × 4
  temps_s vo2_ml fc_bpm puissance_w
    <dbl>  <dbl>  <dbl>       <dbl>
1     180   1800    145         150
2     240   2300    168         200
```

```
# Conditions OU
donnees_test |>
  filter(puissance_w < 50 | puissance_w > 200)
```

```
# A tibble: 2 × 4
  temps_s vo2_ml fc_bpm puissance_w
    <dbl>  <dbl>  <dbl>       <dbl>
1       0    350     72           0
2     300   2450    182         250
```

3.6.c `mutate()` – Créer ou modifier des colonnes

```
# Ajouter une nouvelle colonne
donnees_test |>
  mutate(
    vo2_L = vo2_ml / 1000,  # Convertir mL en L
    temps_min = temps_s / 60  # Convertir s en min
  )
```

```
# A tibble: 6 × 6
  temps_s vo2_ml fc_bpm puissance_w vo2_L temps_min
    <dbl>  <dbl>  <dbl>       <dbl> <dbl>     <dbl>
1       0    350     72           0  0.35         0
2      60    890     95          50  0.89         1
3     120   1250    120         100  1.25         2
4     180   1800    145         150  1.8          3
5     240   2300    168         200  2.3          4
6     300   2450    182         250  2.45         5
```

```
# Modifier une colonne existante
donnees_test |>
  mutate(vo2_ml = round(vo2_ml, -1))  # Arrondir à la dizaine
```

```
# A tibble: 6 × 4
  temps_s vo2_ml fc_bpm puissance_w
    <dbl>  <dbl>  <dbl>       <dbl>
1       0    350     72           0
2      60    890     95          50
3     120   1250    120         100
4     180   1800    145         150
```

```
5     240   2300    168         200
6     300   2450    182         250
```

3.6.d `arrange()` – Trier les lignes

```
# Trier par VO₂ croissant
donnees_test |>
  arrange(vo2_ml)
```

```
# A tibble: 6 × 4
  temps_s vo2_ml fc_bpm puissance_w
    <dbl>  <dbl>  <dbl>       <dbl>
1       0    350     72           0
2      60    890     95          50
3     120   1250    120         100
4     180   1800    145         150
5     240   2300    168         200
6     300   2450    182         250
```

```
# Trier par VO₂ décroissant
donnees_test |>
  arrange(desc(vo2_ml))
```

```
# A tibble: 6 × 4
  temps_s vo2_ml fc_bpm puissance_w
    <dbl>  <dbl>  <dbl>       <dbl>
1     300   2450    182         250
2     240   2300    168         200
3     180   1800    145         150
4     120   1250    120         100
5      60    890     95          50
6       0    350     72           0
```

3.6.e `summarise()` – Résumer les données

```
# Calculer des statistiques résumées
donnees_test |>
  summarise(
    vo2_moyen = mean(vo2_ml),
    vo2_max = max(vo2_ml),
    fc_max = max(fc_bpm)
  )
```

```
# A tibble: 1 × 3
  vo2_moyen vo2_max fc_max
      <dbl>   <dbl>  <dbl>
1     1507.    2450    182
```

3.6.f Enchaîner les opérations

La vraie puissance du tidyverse vient de la possibilité d'enchaîner les opérations :

```
donnees_test |>
  filter(puissance_w > 0) |>           # Exclure le repos
  mutate(vo2_L = vo2_ml / 1000) |>     # Convertir en L
  select(temps_s, puissance_w, vo2_L)  # Garder ces colonnes
```

```
# A tibble: 5 × 3
  temps_s puissance_w vo2_L
    <dbl>       <dbl> <dbl>
1      60          50  0.89
2     120         100  1.25
3     180         150  1.8
4     240         200  2.3
5     300         250  2.45
```

---

4 Partie 4 : Visualisation avec ggplot2

4.1 La grammaire des graphiques

ggplot2 utilise une "grammaire des graphiques" où un graphique est construit couche par couche :

1. **Données** : le tableau à visualiser
2. **Esthétiques** (`aes`) : quelles variables sur quels axes
3. **Géométries** (`geom_`) : comment représenter les données
4. **Autres couches** : titres, thèmes, échelles...

4.2 Structure de base

```
# Structure : ggplot(données, aes(x = ..., y = ...)) + geom_xxx()
ggplot(donnees_test, aes(x = temps_s, y = vo2_ml)) +
  geom_point()
```

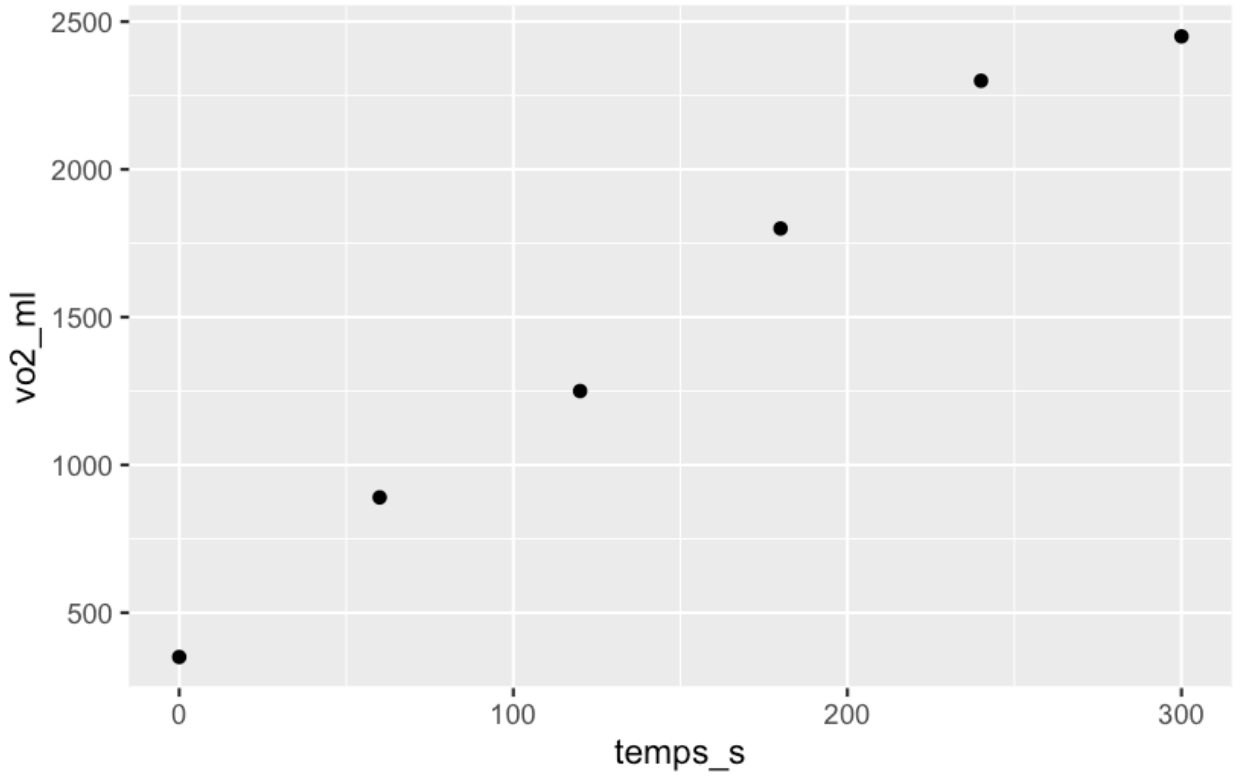


Figure 1: Structure de base d'un graphique ggplot2

4.3 Types de géométries courantes

4.3.a Points : `geom_point()`

```
ggplot(donnees_test, aes(x = puissance_w, y = vo2_ml)) +
  geom_point(size = 3, color = "darkblue")
```

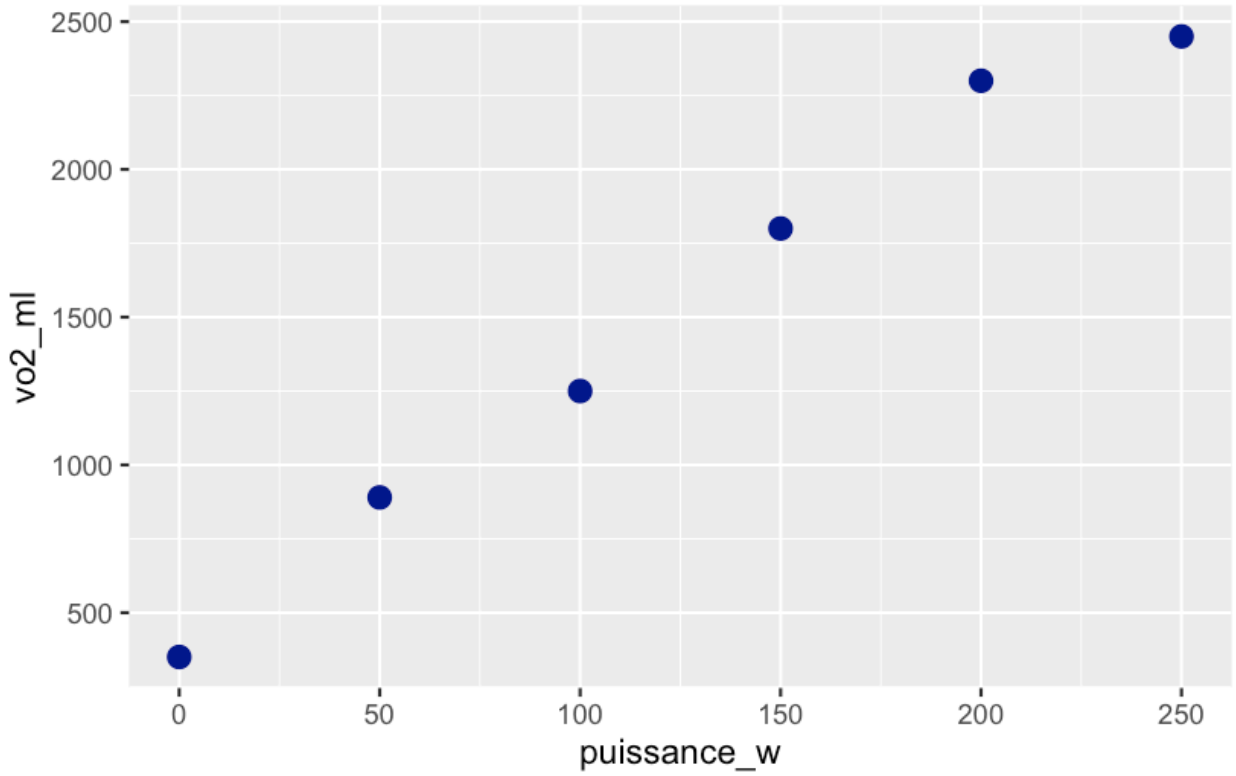


Figure 2: Nuage de points : VO_2 en fonction de la puissance

4.3.b Lignes : `geom_line()`

```
ggplot(donnees_test, aes(x = temps_s, y = vo2_ml)) +
  geom_line(linewidth = 1, color = "darkred")
```

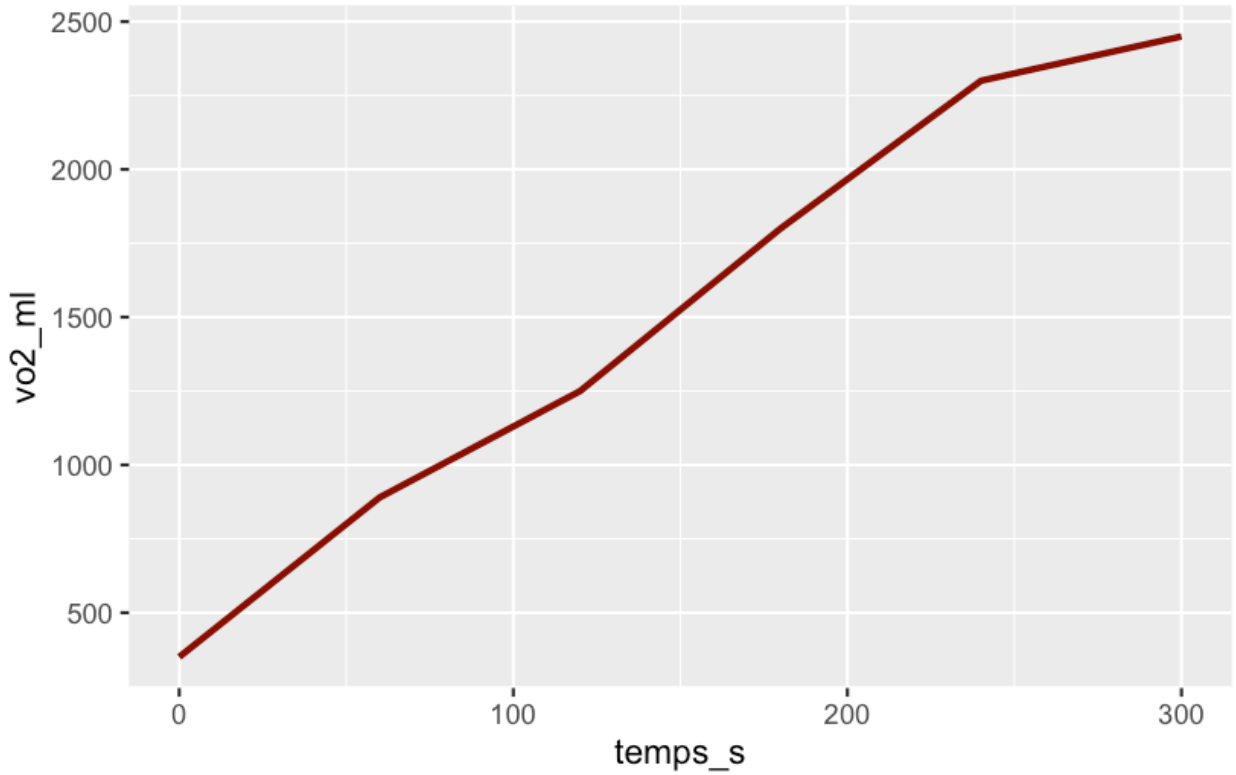


Figure 3: Courbe : évolution du VO_2 dans le temps

4.3.c Combiner points et lignes

```
ggplot(donnees_test, aes(x = temps_s, y = vo2_ml)) +
  geom_line(color = "gray50") +
  geom_point(size = 3, color = "darkblue")
```

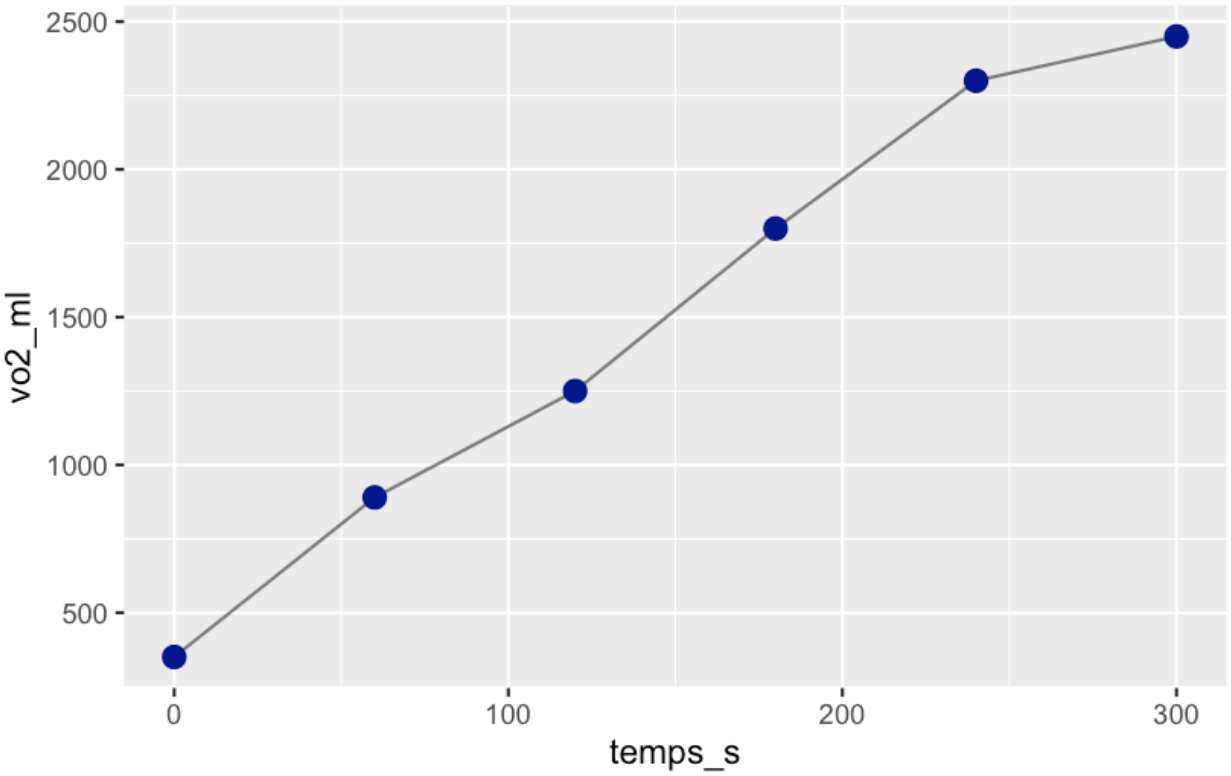


Figure 4: Combinaison de points et de ligne

4.4 Personnaliser les graphiques

4.4.a Ajouter des titres et étiquettes avec `labs()`

```
ggplot(donnees_test, aes(x = temps_s, y = vo2_ml)) +
  geom_line() +
  geom_point() +
  labs(
    title = "Évolution du VO₂ pendant le test",
    subtitle = "Test incrémental maximal",
    x = "Temps (s)",
    y = "VO₂ (mL/min)",
    caption = "Source : données simulées"
  )
```

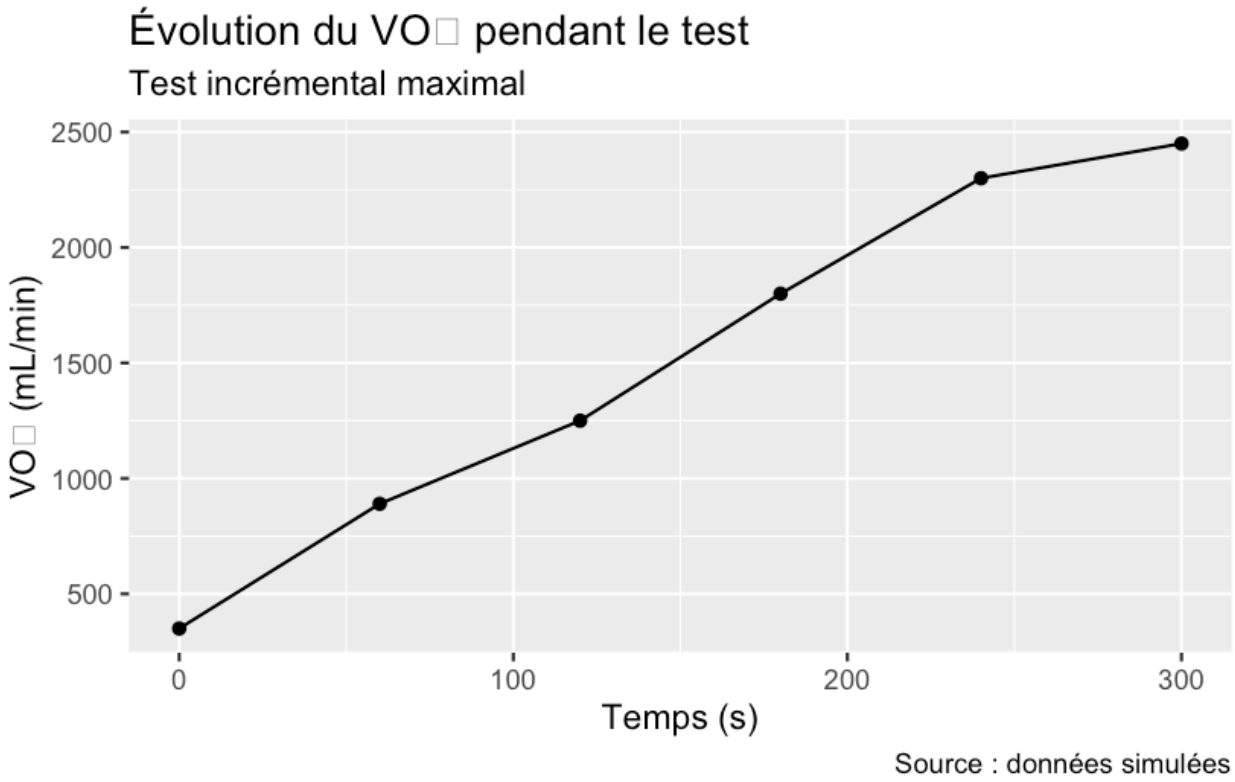


Figure 5: Graphique avec titres et étiquettes

4.4.b Utiliser la couleur pour une variable

```
ggplot(donnees_test, aes(x = temps_s, y = vo2_ml, color = fc_bpm)) +
  geom_point(size = 4) +
  scale_color_viridis_c() +  # Échelle de couleur continue
  labs(
    x = "Temps (s)",
    y = "VO₂ (mL/min)",
    color = "FC (bpm)"
  )
```

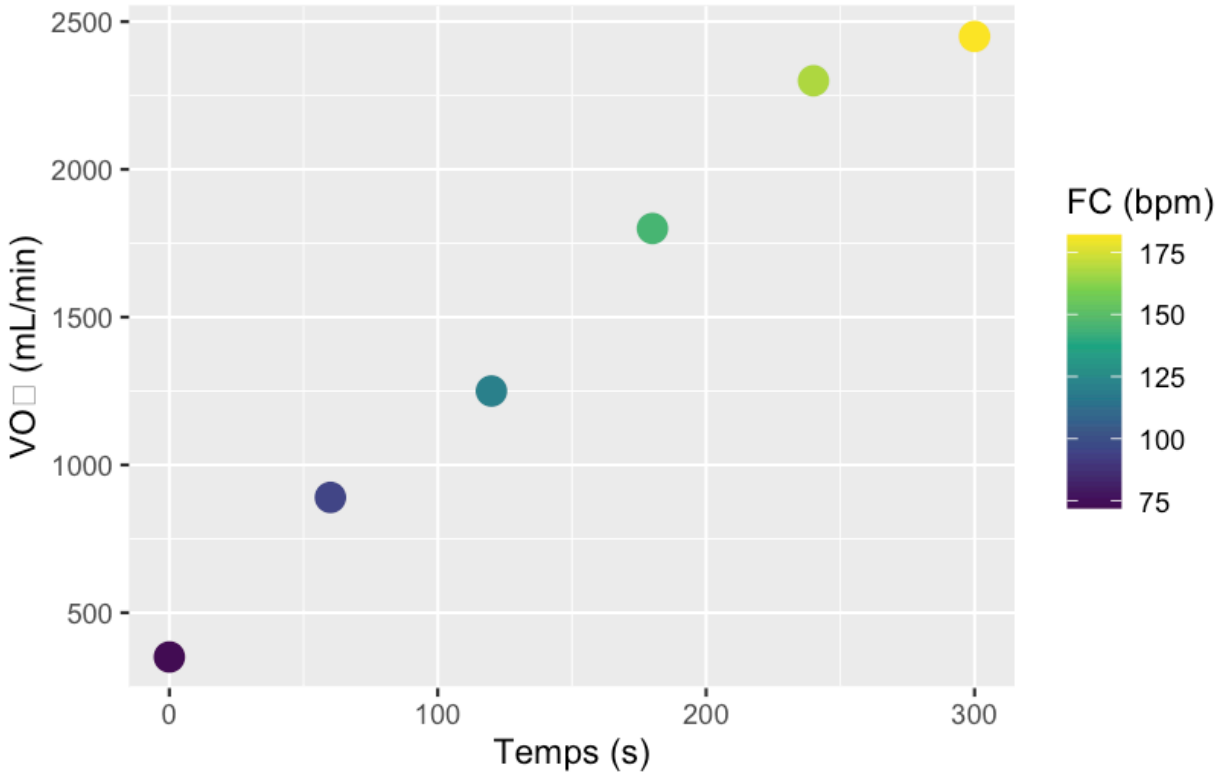


Figure 6: Couleur représentant la fréquence cardiaque

4.4.c Transparence avec `alpha`

```
# Créer des données avec plus de bruit pour illustrer
set.seed(42)
donnees_bruit <- tibble(
  temps = rep(1:100, each = 3),
  vo2 = 2000 + temps * 10 + rnorm(300, 0, 150)
)

ggplot(donnees_bruit, aes(x = temps, y = vo2)) +
  geom_point(alpha = 0.3) +  # Points semi-transparents
  labs(x = "Temps (s)", y = "VO₂ (mL/min)")
```

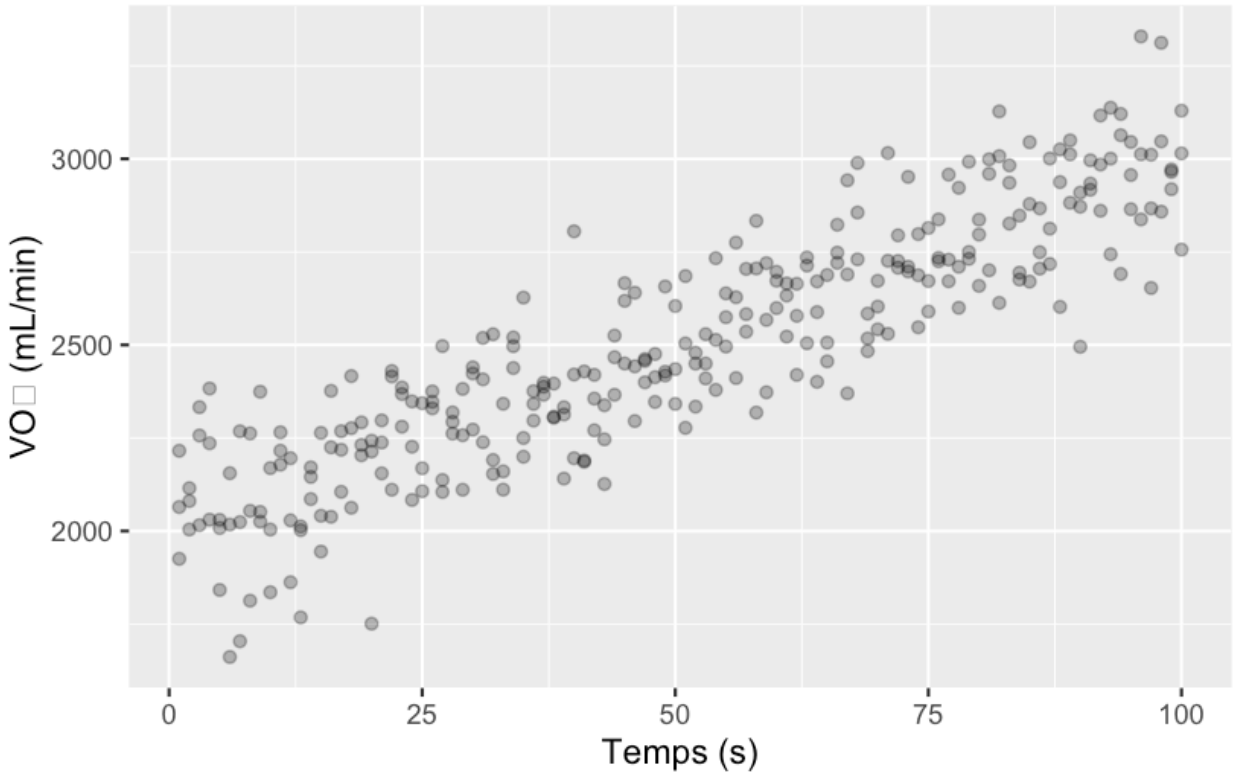


Figure 7: Utilisation de la transparence

4.4.d Thèmes prédéfinis

```r
# Graphique de base
p <- ggplot(donnees_test, aes(x = temps_s, y = vo2_ml)) +
  geom_line() +
  geom_point() +
  labs(x = "Temps (s)", y = "VO₂ (mL/min)")

# Thème minimaliste (recommandé pour les publications)
p + theme_minimal()
```

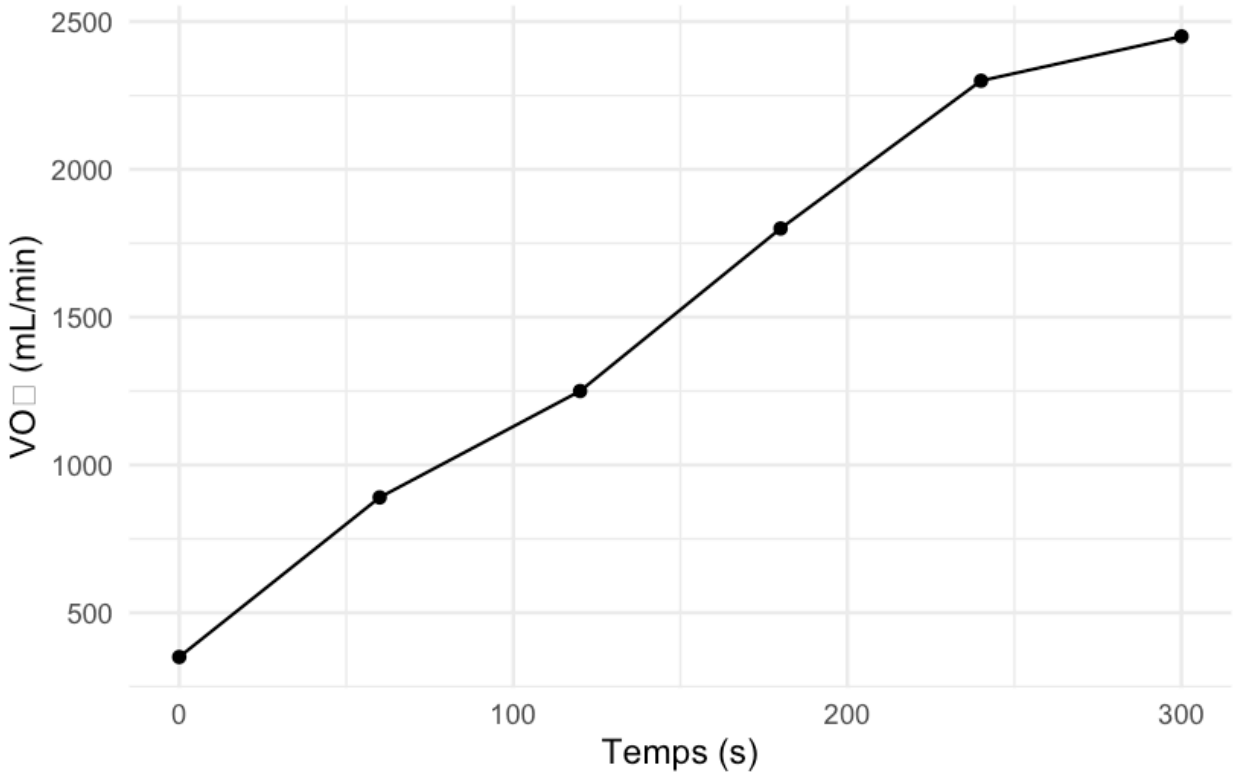


Figure 8: Différents thèmes disponibles

4.5 Superposer des données

Un patron très utile : superposer données brutes et données traitées.

```
# Simuler des données brutes vs lissées
set.seed(123)
exemple <- tibble(
  temps = 1:50,
  vo2_brut = 1500 + temps * 20 + rnorm(50, 0, 100),
  vo2_lisse = 1500 + temps * 20
)

ggplot(exemple, aes(x = temps)) +
  geom_line(aes(y = vo2_brut), alpha = 0.3, color = "gray40") +
  geom_line(aes(y = vo2_lisse), color = "blue", linewidth = 1.2) +
  labs(
    title = "Comparaison données brutes vs lissées",
    x = "Temps (s)",
    y = "VO₂ (mL/min)"
  ) +
  theme_minimal()
```

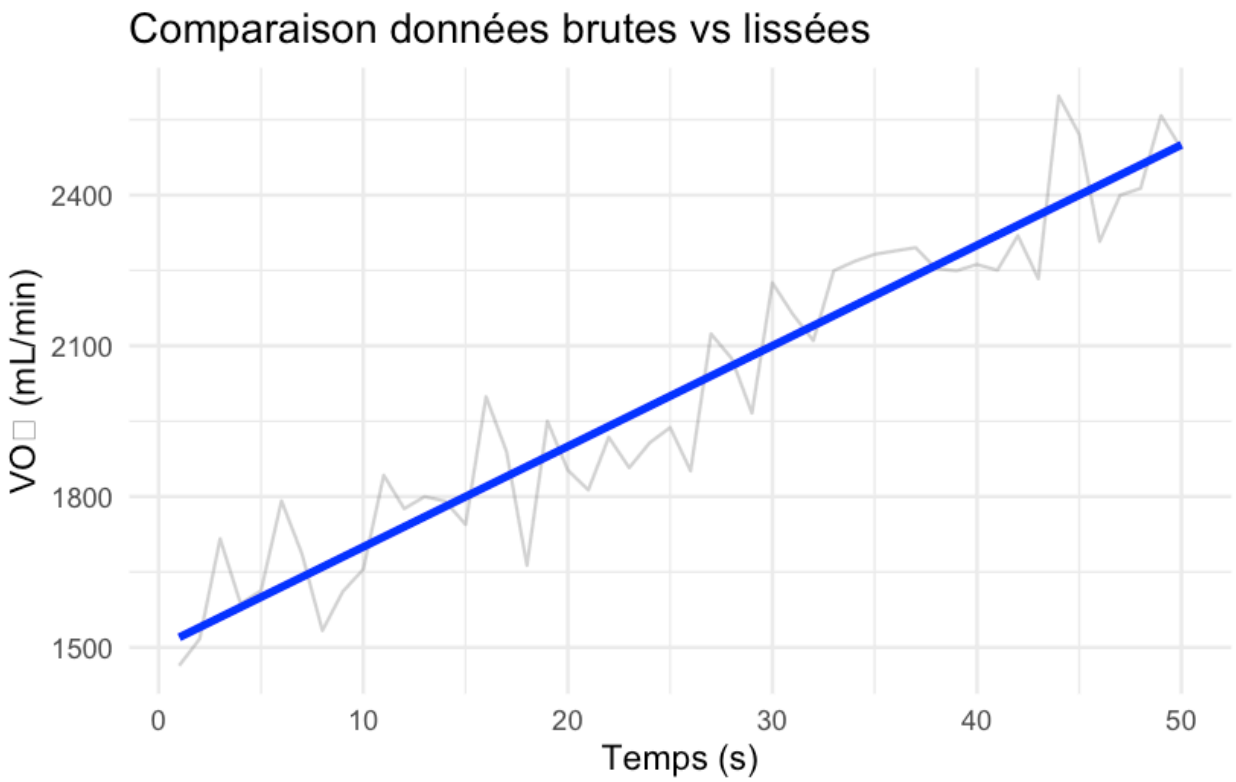


Figure 9: Superposition de données brutes et lissées

5 Partie 5 : Documents Quarto

5.1 Structure d'un document Quarto

Un document `.qmd` contient trois éléments :

1. **En-tête YAML** (entre `---`) : métadonnées du document
2. **Texte Markdown** : contenu narratif
3. **Blocs de code** : analyses R

5.1.a L'en-tête YAML

```yaml
---
title: "Mon titre"
author: "Mon nom"
date: today
format:
  html:
    toc: true
execute:
  echo: true
---
```

Options courantes :

Option	Description
`toc: true`	Table des matières
`code-fold: true`	Code masquable
`echo: true/false`	Afficher/masquer le code
`eval: true/false`	Exécuter/ignorer le code

5.1.b Texte Markdown

```
# Titre de niveau 1
## Titre de niveau 2
### Titre de niveau 3

Texte normal avec *italique* et **gras**.

- Liste à puces
- Autre élément

1. Liste numérotée
2. Deuxième élément

[Lien vers un site](https://www.umontreal.ca)
```

5.1.c Blocs de code R

Les blocs de code sont délimités par trois accents graves :

````
::: {.cell}

```{.r .cell-code}
# Votre code R ici
mean(c(1, 2, 3))
```

::: {.cell-output .cell-output-stdout}

```
[1] 2
```



:::
:::
````

5.2 Options des blocs de code

Chaque bloc peut avoir ses propres options :

```
{{r}}
#| label: mon-graphique
#| fig-cap: "Ma légende de figure"
#| echo: false
#| eval: true

ggplot(data, aes(x, y)) + geom_point()
```

Option	Description
`label`	Nom unique du bloc (pour références)
`echo`	Afficher le code (`true`/`false`)
`eval`	Exécuter le code (`true`/`false`)
`fig-cap`	Légende de la figure
`warning`	Afficher les avertissements
`message`	Afficher les messages

5.3 Générer le document (Render)

Pour produire votre document final :

1. Cliquez sur **Render** (ou `Ctrl+Shift+K`)
2. Choisissez le format : HTML, PDF (via Typst), ou Word

Formats de sortie

- **HTML** : interactif, idéal pour l'exploration
- **Typst/PDF** : mise en page fixe, idéal pour impression
- **Word** : si révision/commentaires nécessaires

5.4 Callouts (encadrés)

Quarto offre des encadrés colorés pour attirer l'attention :

Note

Pour des informations complémentaires.

Astuce

Pour des conseils pratiques.

Attention

Pour des mises en garde.

! Important

Pour des points essentiels.

Syntaxe :

```
::: {.callout-note}
## Titre de l'encadré
Contenu de l'encadré.
:::
```

---

6 Partie 6 : Exercices de préparation

Complétez ces exercices pour vérifier votre compréhension.

6.1 Exercice 1 : Manipulations de base

```
# 1. Créez un vecteur "puissances" contenant : 50, 100, 150, 200, 250, 300
puissances <- ___

# 2. Calculez la moyenne de ce vecteur
moyenne_puissance <- ___

# 3. Trouvez la valeur maximale
puissance_max <- ___

# 4. Créez un nouveau vecteur avec chaque valeur divisée par 10
puissances_kw <- ___
```

Solution

```
puissances <- c(50, 100, 150, 200, 250, 300)
moyenne_puissance <- mean(puissances)
puissance_max <- max(puissances)
puissances_kw <- puissances / 10

# Vérification
moyenne_puissance
```

```
[1] 175
```

```
puissance_max
```

```
[1] 300
```

```
puissances_kw
```

```
[1]  5 10 15 20 25 30
```

6.2 Exercice 2 : Manipulation de tableau

```
# Données pour l'exercice
exercice_data <- tibble(
  sujet = c("A", "B", "C", "D", "E"),
  vo2max_ml = c(3200, 4100, 2800, 3600, 4500),
  masse_kg = c(70, 82, 58, 75, 90),
  age = c(25, 32, 28, 22, 35)
)
```

```
# 1. Affichez un aperçu des données avec glimpse()


# 2. Filtrez les sujets avec un VO₂max > 3500 mL/min


# 3. Ajoutez une colonne vo2max_relatif (mL/kg/min) = vo2max_ml / masse_kg


# 4. Triez par vo2max_relatif décroissant
```

Solution

```
# 1. Aperçu
glimpse(exercice_data)
```

```
Rows: 5
Columns: 4
$ sujet     <chr> "A", "B", "C", "D", "E"
$ vo2max_ml <dbl> 3200, 4100, 2800, 3600, 4500
$ masse_kg  <dbl> 70, 82, 58, 75, 90
$ age       <dbl> 25, 32, 28, 22, 35
```

```
# 2. Filtrer
exercice_data |>
  filter(vo2max_ml > 3500)
```

```
# A tibble: 3 × 4
  sujet vo2max_ml masse_kg   age
  <chr>     <dbl>    <dbl> <dbl>
1 B          4100       82    32
2 D          3600       75    22
3 E          4500       90    35
```

```
# 3. Ajouter colonne
exercice_data |>
  mutate(vo2max_relatif = vo2max_ml / masse_kg)
```

```
# A tibble: 5 × 5
  sujet vo2max_ml masse_kg   age vo2max_relatif
  <chr>     <dbl>    <dbl> <dbl>          <dbl>
1 A          3200       70    25           45.7
2 B          4100       82    32           50
3 C          2800       58    28           48.3
4 D          3600       75    22           48
5 E          4500       90    35           50
```

```
# 4. Trier
exercice_data |>
  mutate(vo2max_relatif = vo2max_ml / masse_kg) |>
  arrange(desc(vo2max_relatif))
```

```
# A tibble: 5 × 5
  sujet vo2max_ml masse_kg   age vo2max_relatif
  <chr>     <dbl>    <dbl> <dbl>          <dbl>
1 B          4100       82    32           50
2 E          4500       90    35           50
3 C          2800       58    28           48.3
4 D          3600       75    22           48
5 A          3200       70    25           45.7
```

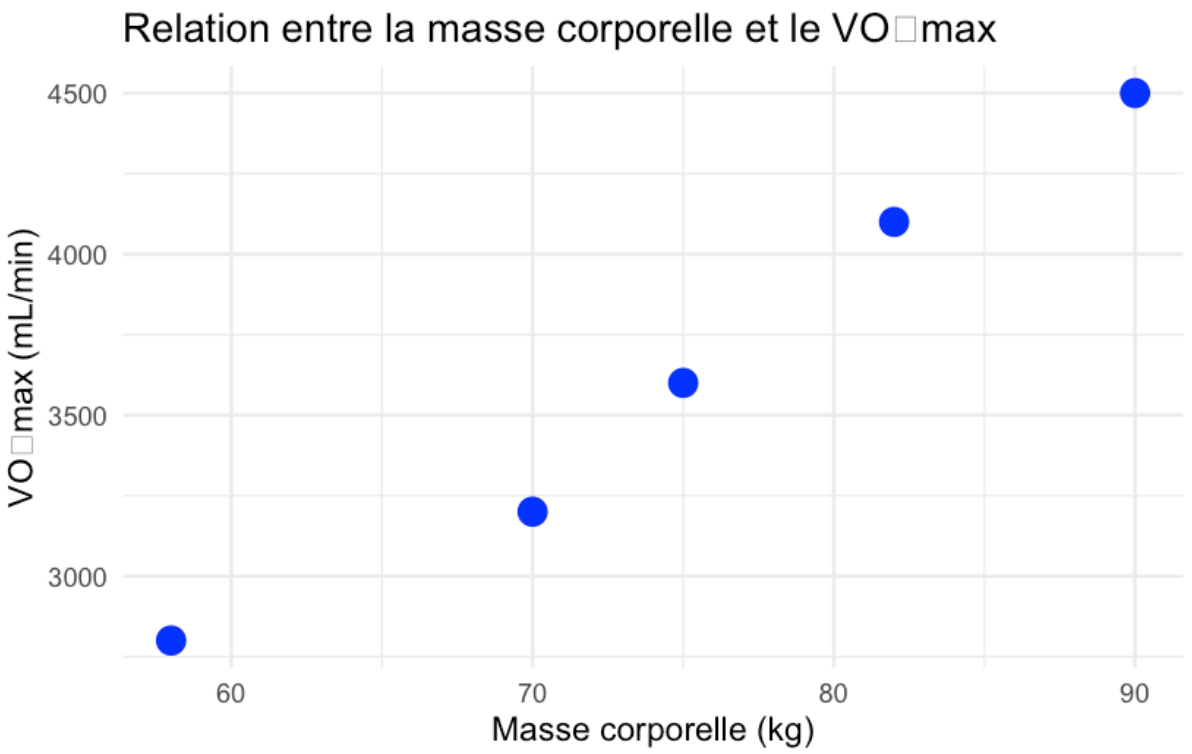
6.3 Exercice 3 : Graphique

```
# Créez un graphique montrant la relation entre masse_kg (axe x)
# et vo2max_ml (axe y) avec :
# - Des points de taille 4
# - Une couleur bleue
# - Un titre approprié
# - Des étiquettes d'axes en français

ggplot(exercice_data, aes(___)) +
  geom____() +
  labs(
    ___
  )
```

Solution

```
ggplot(exercice_data, aes(x = masse_kg, y = vo2max_ml)) +
  geom_point(size = 4, color = "blue") +
  labs(
    title = "Relation entre la masse corporelle et le VO₂max",
    x = "Masse corporelle (kg)",
    y = "VO₂max (mL/min)"
  ) +
  theme_minimal()
```



7 Résumé des fonctions essentielles

7.1 Fonctions R de base

Fonction	Description	Exemple
`<-`	Assignation	`x <- 5`
`c()`	Créer un vecteur	`c(1, 2, 3)`
`mean()`	Moyenne	`mean(x)`
`max()`	Maximum	`max(x)`
`min()`	Minimum	`min(x)`
`sd()`	Écart-type	`sd(x)`
`sqrt()`	Racine carrée	`sqrt(16)`
`round()`	Arrondir	`round(3.14, 1)`
`length()`	Longueur	`length(x)`

7.2 Fonctions tidyverse

Fonction	Package	Description
`read_csv()`	readr	Importer un CSV
`read_excel()`	readxl	Importer un fichier Excel
`glimpse()`	dplyr	Aperçu des données
`filter()`	dplyr	Filtrer les lignes
`select()`	dplyr	Choisir des colonnes
`mutate()`	dplyr	Créer/modifier colonnes
`arrange()`	dplyr	Trier les lignes
`summarise()`	dplyr	Résumer les données
`tibble()`	tibble	Créer un tableau

7.3 Fonctions ggplot2

Fonction	Description
`ggplot()`	Initialiser un graphique
`aes()`	Définir les esthétiques
`geom_point()`	Nuage de points
`geom_line()`	Ligne
`labs()`	Titres et étiquettes
`theme_minimal()`	Thème épuré

8 Ressources supplémentaires

- R for Data Science (2e édition) — Le livre de référence (gratuit en ligne)
- Quarto Documentation — Guide officiel de Quarto
- ggplot2 Cheat Sheet — Aide-mémoire visuel

! Prêt pour le laboratoire?

Si vous avez complété ce tutoriel et les exercices, vous êtes prêt pour l'activité en classe! N'hésitez pas à revenir consulter ce document comme référence pendant le laboratoire.