Un document Quarto

Une petite introduction

Votre nom

14 octobre 2025

1 Introduction

Voici la première section de notre document. Notez l'étiquette entre `{}`, qui nous permet de nous référer à une section plus tard. Par exemple, la section suivante a l'identifiant `#sec-problemes`. On peut donc écrire `@sec-problemes` dans le texte pour dire « lisez la Section 2 pour savoir comment utiliser Quarto dans les problèmes du cours ». Voici un mot en *italique* et un mot en **gras**. Pour le code (fonctions, bibliothèque, etc.), on utilise une police à espacement fixe : `tidyverse` (utilisez les accents graves autour du mot ou de la séquence).

2 Comment répondre aux questions des problèmes ?

Le problème comprendra un fichier `.qmd`, donc vous n'aurez pas besoin de créer un nouveau fichier de zéro. La majorité des questions exigeront un peu de code. C'est en effet l'un des avantages de Quarto : la possibilité d'ajouter nos codes *dans* le texte de façon automatisée. Il y a deux façons de travailler avec le code : en ligne ou en bloc. Le code en ligne vous permet d'imprimer dans le texte quelques informations qui sont dans un fichier, ou de calculer quelque chose (c'est comme une console R). Par exemple, le fichier `villes.csv` est chargé ci-dessous. Dans le `PDF`, c'est invisible, parce que le bloc contient la ligne `|# echo: false`. Donc, vous ne voyez que le résultat du code. Vu que le code ne génère pas de résultat, le `PDF` n'affiche rien. Toutefois, l'extension `tidyverse` et le fichier de données sont effectivement chargés maintenant. Cela nous permet de calculer, par exemple, la note moyennes dans le fichier *en ligne*. Le voici : la note moyenne est de 71.88, à partir de 100 observations. Si jamais vous changez le fichier, le texte sera mis à jour lorsque vous le compilerez.

2.1 Les blocs de code

Un bloc de code doit commencer avec trois accents graves suivis par `r`. Dans le bloc, on ajoute quelques options, qui seront déjà dans le modèle des problèmes. Bref, ici on ajoute `message: false` et `warning: false` pour éviter ces éléments dans notre `PDF`. Lorsque le bloc est responsable pour générer une figure, il y a des arguments pour l'ajuster aussi.

````
```{r}
#| message: false
#| warning: false

# Votre code ici
```
````

2.2 Exemples

Générons des tableaux et une figure avec nos blocs de code. On utilise `villes.csv`, déjà chargé, ainsi que `tidyverse`. Pour notre problème, vous verrez que les blocs auront la ligne `echo: true`. Cela imprimera **le code** ainsi que son résultat (le cas échéant) dans le `PDF`. Toutefois, dans un article typique, on utiliserai `echo: false`, pour cacher le code.

```r
ggplot(data = v, aes(x = ville, y = note)) +
  geom_boxplot() +
  stat_summary(color = "red") +
  theme_classic() +
  labs(
    x = "Ville",
    y = "Note"
  )
```

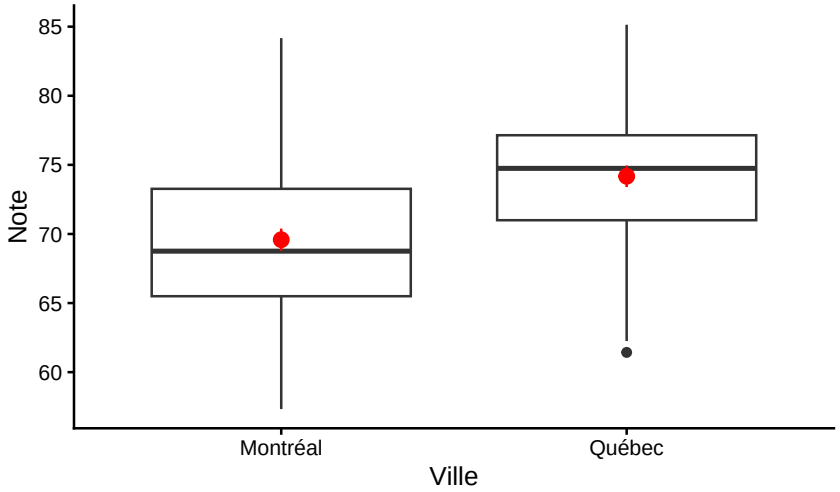

FIGURE 1 : Un graphique

Puisqu'on utilise `label`, on peut maintenant nous référer à la figure en utilisant `@fig-maFigure1` : « la Figure 1 montre qu'il y a probablement une différence entre les villes ».

Vous pouvez également générer des tableaux avec la bibliothèque `knitr` et la fonction `kable()`. C'est facile et le résultat sera un tableau déjà mis en forme. Vous verrez dans le code ci-dessous que j'utilise les accents graves pour les noms des colonnes. La raison est simple : il y a un espace et un trait d'union dans les noms en question.

```
v |>
  summarize(
    `note moyenne` = mean(note) |> round(2),
    `écart-type` = sd(note) |> round(2),
    .by = ville
  ) |>
  kable()
```

TABLEAU 1 : Les notes moyennes des villes

ville	note moyenne	écart-type
Québec	74.17	5.51
Montréal	69.58	5.67

Maintenant, on peut nous référer au Tableau 1 aussi. Finalement, vous pouvez imprimer un tableau à partir d'une régression en utilisant la bibliothèque `sjPlot`, déjà discutée dans le cours, ou la bibliothèque `modelsummary`, utilisé pour créer le Tableau 2.

```r
fit <- lm(note ~ ville, data = v)
tab_model(fit) # version avec sjPlot
# L'option ci-dessous exige la bibliothèque modelsummary,
# qui ne semble pas fonctionner correctement dans la version
# en ligne (posit.cloud). Utilisez plutôt tab_model().
# fit |> modelsummary(
#   shape = term ~ model + statistic,
#   statistic = c("conf.int", "p.value"),
#   gof_map = c("adj.r.squared", "nobs")
# )
```

Tableau 2 : Régression linéaire

		note	
Predictors	Estimates	CI	p
(Intercept)	69.58	68.02 – 71.15	**<0.001**
ville [Québec]	4.59	2.37 – 6.81	**<0.001**
Observations	100		
R^2 / R^2 adjusted	0.147 / 0.138		

3 Gestion bibliographique

Bien que les références bibliographiques ne joueront pas un rôle essentiel dans notre cours, il est utile d'apprendre comment citer des articles ou des bibliothèque dans notre document. Par exemple, on a utilisé `tidyverse` ci-dessous. Si vous exécuter `citation("tidyverse")` dans votre console, vous verrez la référence en format `bib` pour la bibliothèque. Copiez cette référence et collez-la dans le fichier `references.bib`. Cela vous permettra de citer `tidyverse` (Wickham et al. 2019)—notez la référence complète à la fin du document. Consultez les fichiers supplémentaires et examiner `references.bib` attentivement.

Références

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Cha ng, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. « Welcome to the tidyverse ». *Journal of Open Source Software* 4 (43) : 1686. https://doi.org/10.21105/joss.01686.