

RAPPORTS ÉPIDÉMIOLOGIQUES

AVEC

MEILLEURES PRATIQUES DES TABLEAUX
AUX SÉRIES TEMPORELLES



The GRAPH Courses, le Fonds Mondial & l'OMS

Ce livre est une compilation de matériaux de formation créée par The GRAPH Network, soutenue par le Fonds Mondial. Les matériaux visent à développer des compétences globales dans l'analyse des données épidémiologiques.

Pyramides démographiques pour l'analyse épidémiologique

Introduction
Objectifs d'Apprentissage
Introduction aux Pyramides Démographiques
L'utilisation des Pyramides Démographiques en Épidémiologie
Conceptualisation des Pyramides Démographiques
Paquets
Préparation des Données
Introduction au Jeu de Données
Importation des Données
Inspection des Données
Création d'un sous-ensemble de données agrégées
Création du graphique
Utilisation de <code>geom_col()</code> pour les pyramides démographiques
Personnalisation du Graphique
Axes
Étiquettes Personnalisées
Schéma de Couleurs et les Thèmes
En Résumé !
Solutions
References

Introduction

Une pyramide démographique, également appelée pyramide des âges ou pyramide par sexe, aide à visualiser la distribution d'une population selon deux variables démographiques importantes : **l'âge** et le **sexé**.

Aujourd'hui, vous apprendrez l'importance d'utiliser des pyramides démographiques pour visualiser la distribution d'une maladie par âge et par sexe, ainsi que comment en créer une avec `{ggplot2}`.

Allons-y !

Objectifs d'Apprentissage

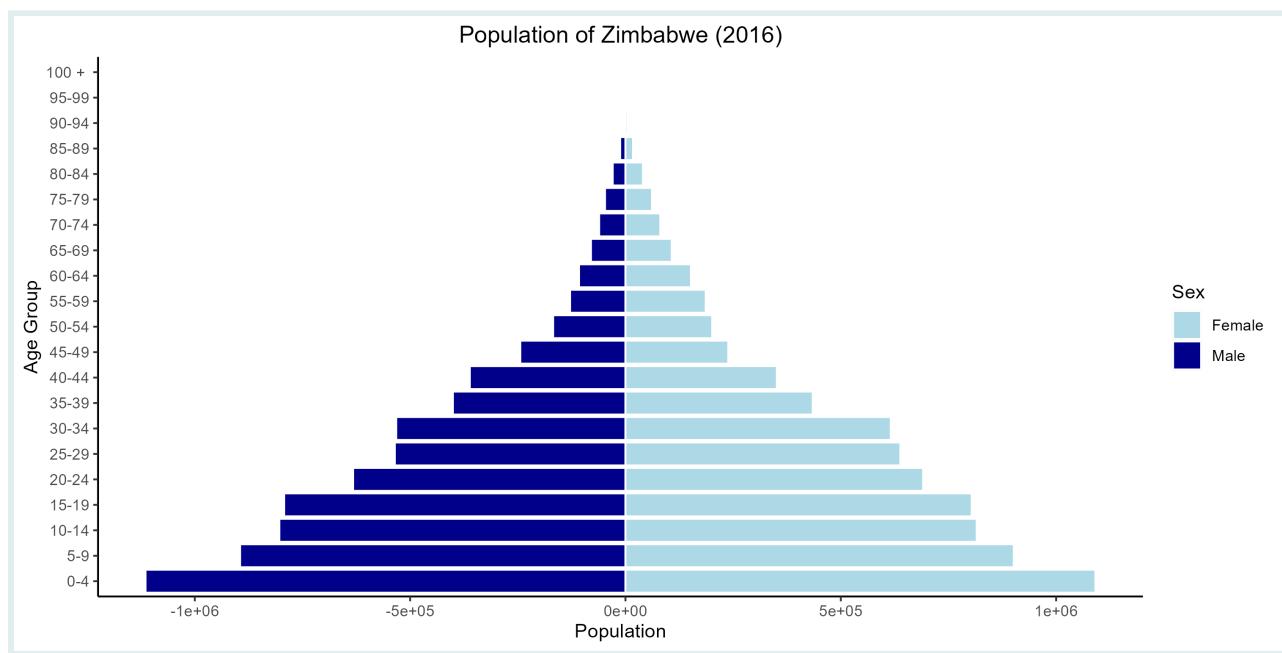
À la fin de cette leçon, vous serez en mesure de :

- Expliquer **l'importance des pyramides démographiques** pour communiquer les **patterns spécifiques à l'âge et au sexe** de la distribution d'une maladie.
- Comprendre les **composants d'une pyramide démographique** et la conceptualiser comme une version modifiée d'un **graphique à barres empilées**.

- **Résumer et préparer les données** dans le format approprié pour le tracé avec les fonctions `{dplyr}`.
- Utiliser le code `{ggplot2}` pour **tracer une pyramide démographique à l'aide de `geom_col()`**, montrant les totaux ou pourcentages sur l'axe x.
- **Personnaliser le graphique** en changeant le schéma de couleurs, les étiquettes et les axes.

Introduction aux Pyramides Démographiques

Les pyramides démographiques sont des graphiques qui montrent la distribution des âges au sein d'une population. Le graphique est divisé en deux parties entre les membres masculins et féminins de la population, où l'axe des y montre les groupes d'âge et l'axe des x le sexe.



Pyramide des âges pour un pays avec une population jeune ayant une forme de pyramide.

Le graphique global prend souvent la forme d'une pyramide, d'où son nom.

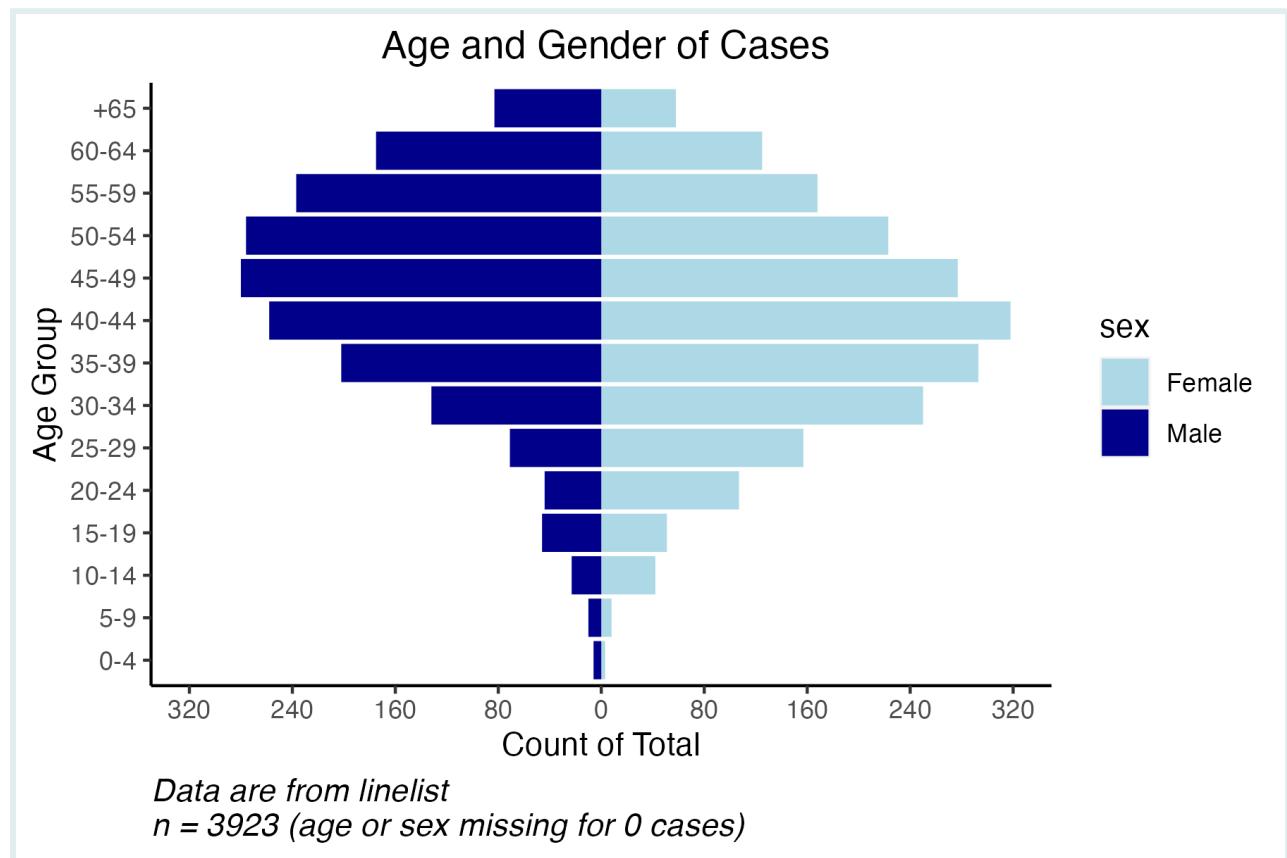
Trois caractéristiques clés d'une pyramide démographique sont :

1. **Groupes d'Âge** : La pyramide est divisée en barres horizontales, chacune représentant un groupe d'âge, souvent par tranches de cinq ans (0-4, 5-9, 10-14, etc.).
2. **Représentation par Sexe** : Le côté gauche de la pyramide représente généralement les hommes, et le côté droit représente les femmes, permettant

une comparaison rapide de la distribution entre les sexes dans chaque groupe d'âge.

3. Taille de la Population : La longueur des barres indique la taille de la population dans chaque groupe d'âge. Une barre plus longue suggère une plus grande population dans ce groupe d'âge.

À l'aide de `{ggplot2}`, nous sommes en mesure de créer des pyramides tout en les adaptant à nos besoins spécifiques. À la fin de cette leçon, notre graphique final ressemblera à ceci :



L'utilisation des Pyramides Démographiques en Épidémiologie

Les pyramides démographiques sont utiles pour décrire et comprendre l'épidémiologie de diverses maladies, en visualisant la distribution de la maladie par âge et par sexe.

Vulnérabilité Spécifique à l'Âge et au Sexe

Différentes maladies peuvent affecter différemment les groupes d'âge. Nous savons que l'incidence de certaines maladies transmissibles peut varier en fonction de l'âge. Dans le cas de la tuberculose (TB) en Afrique, les adolescents et les jeunes adultes sont principalement touchés dans la région. Cependant, dans des pays où l'incidence de la TB a diminué de manière significative, comme aux États-Unis, elle est principalement observée chez les personnes plus âgées ou immunodéprimées. Une autre maladie qui démontre une variation liée à l'âge est le paludisme, où les

enfants de moins de 5 ans représentent une grande majorité des décès en Afrique. Le VIH a été démontré pour affecter davantage les femmes que les hommes, en particulier dans les groupes d'âge plus jeunes. Cela pourrait être dû à une vulnérabilité biologique ou à des facteurs sociaux.

Par conséquent, lors de la description de l'épidémiologie des maladies transmissibles telles que le VIH, le paludisme et la TB, il est important d'observer la distribution des cas ou des décès par groupe d'âge et par sexe. Ces informations aident à informer les programmes nationaux de surveillance en identifiant les groupes d'âge et de sexe qui connaissent la plus grande charge, et qui cibler pour l'intervention.

Utilisation de la Distribution Démographique pour l'Évaluation de la Qualité des Données

Les pyramides démographiques peuvent également jouer un rôle crucial dans l'évaluation de la qualité des données des systèmes de surveillance de routine en aidant à évaluer la cohérence interne et externe.

SIDE NOTE

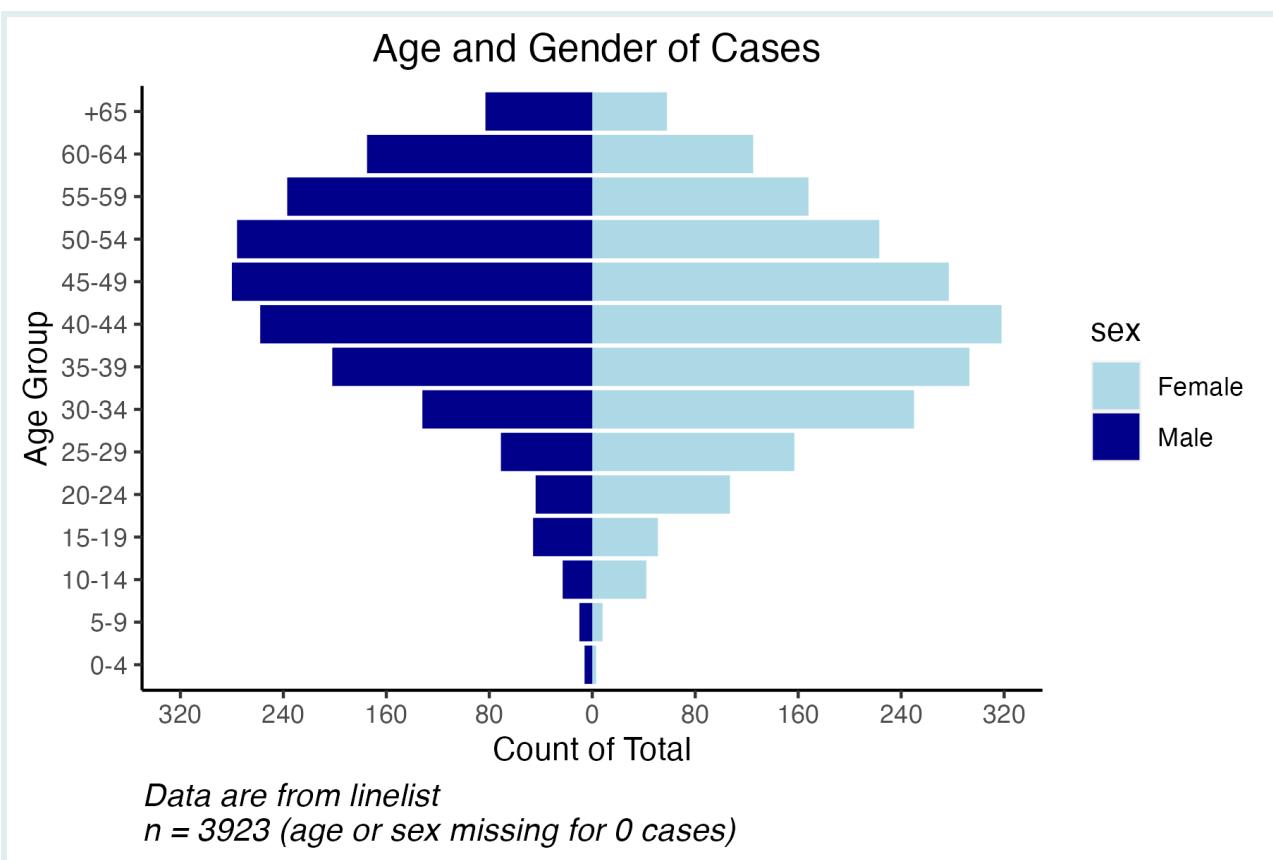


Lors de la tentative d'évaluation des normes de qualité des données de surveillance de certaines maladies, la cohérence externe peut être évaluée en comparant les données de surveillance nationales avec l'épidémiologie mondiale de cette maladie. Les calculs de données basées sur des variables démographiques telles que le groupe d'âge sont parfois utilisés.

Dans le cas des données de surveillance de la TB, la cohérence externe peut être évaluée en calculant le pourcentage d'enfants diagnostiqués avec la TB dans le programme et en le comparant avec la moyenne mondiale des cas.

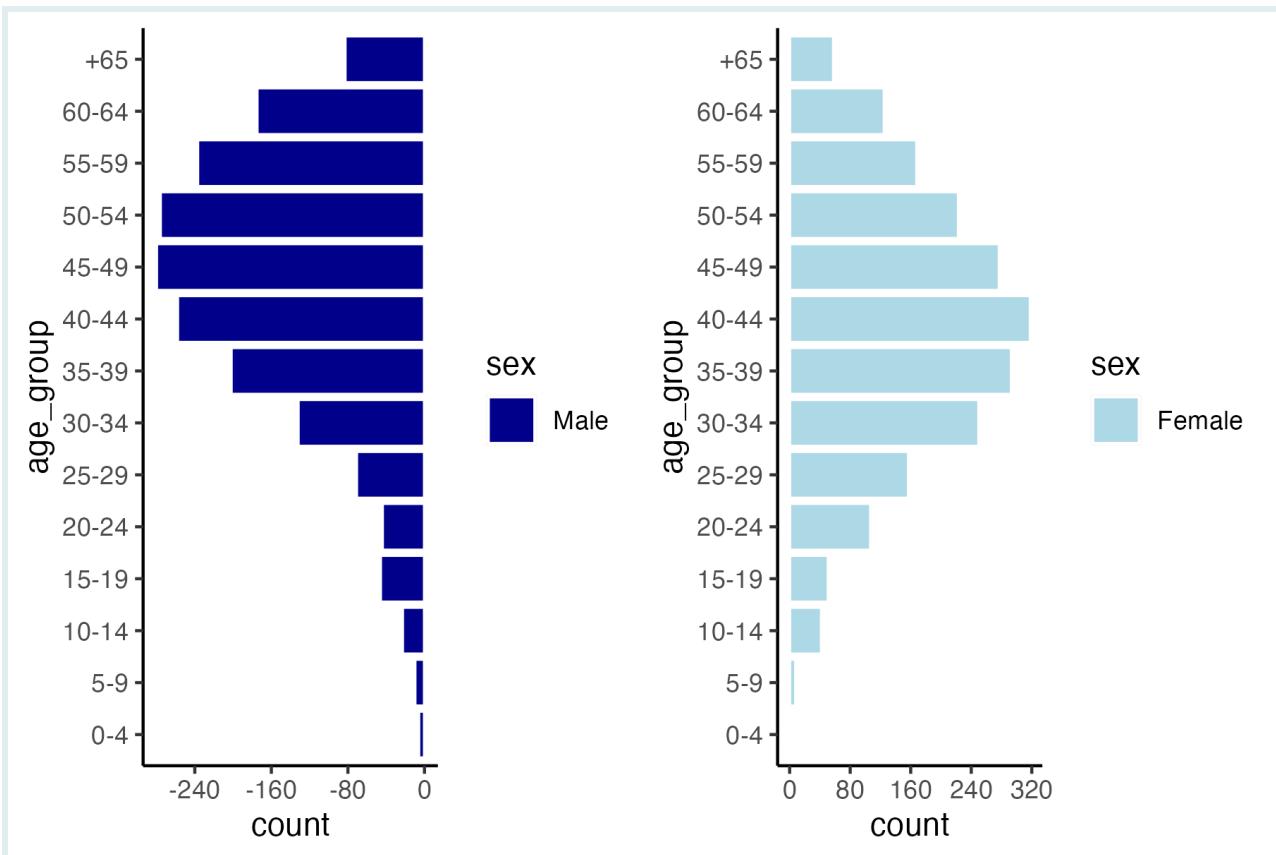
Conceptualisation des Pyramides Démographiques

Jetons un coup d'œil plus attentif à notre pyramide démographique cible, et décomposons comment elle peut être tracée à l'aide de `geom_col()` de `{ggplot2}`.



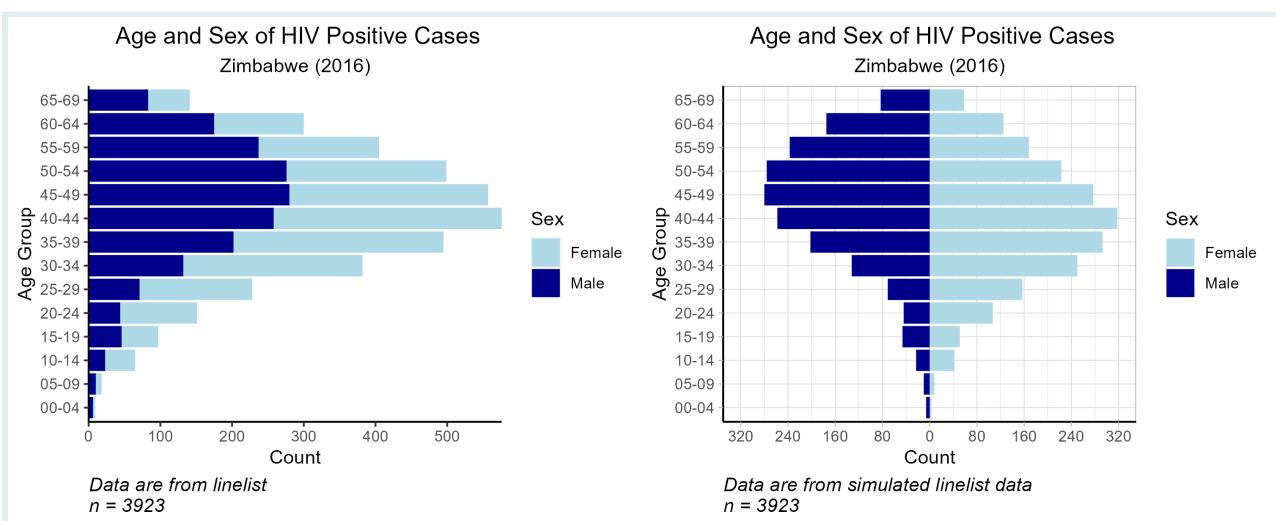
Comme vous pouvez le voir, l'axe des x est divisé en deux moitiés (hommes et femmes), qui sont tracées dans des directions opposées à partir de 0. Les unités sur l'axe des x sont symétriques de chaque côté, et les groupes d'âge sont étiquetés le long de l'axe des y.

En d'autres termes, nous pouvons le considérer comme deux **graphiques à barres** - un pour les hommes et un pour les femmes.



Une pyramide démographique peut également être conceptualisée comme une forme spécialisée d'un graphique à barres empilées. Dans un graphique à barres empilées traditionnel, les segments sont empilés les uns sur les autres, à partir d'une ligne de base de zéro et s'étendant vers l'extérieur.

En revanche, une pyramide démographique aligne les barres dos à dos le long d'un axe central. Cet axe représente le point de division entre deux catégories (populations masculines et féminines) avec les barres s'étendant dans des directions opposées pour illustrer la proportion de chaque groupe d'âge au sein de ces catégories.



KEY POINT

Une pyramide démographique ressemble à une version modifiée d'un graphique à barres empilées avec deux distinctions clés :

1. **Axe Central** : Au lieu de superposer les segments de barres les uns sur les autres, les barres sont alignées dos à dos le long d'un **axe central**, qui sert de point de division entre les populations masculines et féminines.
2. **Orientation des Barres** : Contrairement aux barres empilées traditionnelles qui partent de zéro vers l'extérieur dans une seule direction, les barres de la pyramide démographique s'étendent dans des **directions opposées** pour représenter la taille de chaque groupe d'âge au sein des catégories.

Paquets

Cette leçon nécessitera l'installation et le chargement des paquets suivants :

```
# Charger les paquets
pacman::p_load(here,          # pour localiser les fichiers
                tidyverse,      # pour manipuler et tracer les données (inclut
                ggplot2)        # paquet dédié à la création de pyramides par âge
                apyramid)
```

REMINDER

Chaque leçon de The GRAPH Courses est accompagnée d'un script RMarkdown pour la programmation, que vous devriez télécharger et utiliser pour suivre la leçon vidéo ou les notes de cours.

Préparation des Données

Introduction au Jeu de Données

Pour cette leçon, nous utiliserons un jeu de données simulées sur le VIH imitant une liste linéaire des cas de VIH au Zimbabwe en 2016. Pour cette leçon spécifique,

nous nous concentrerons sur les variables **âge** et **sex** pour créer notre pyramide démographique.

Importation des Données

Commençons par importer nos données dans notre environnement RStudio et les examiner de plus près pour mieux comprendre les variables que nous utiliserons pour la création de notre pyramide démographique.

```
hiv_data <- read_csv(here::here("data/hiv_zw_linelist_2016.csv"))

hiv_data
```

Notre jeu de données importé contient **28000** lignes et **3** colonnes contenant les variables `age_group` et `sex` que nous utiliserons pour la création de notre pyramide démographique. Chaque ligne (ligne) correspond à un patient, tandis que chaque colonne représente différentes variables d'intérêt. La liste linéaire ne contient que des variables démographiques et liées au VIH (statut VIH). De plus, la variable `hiv_status` nous fournit des informations sur le statut des individus (*positif* ou *négatif*).

Comme nous sommes intéressés par la création d'une pyramide démographique sur la prévalence du VIH, nous devons d'abord filtrer pour les individus VIH positifs.

Filtrons les données !

```
# Créez un sous-ensemble avec seulement les individus VIH positifs
hiv_cases <- hiv_data %>%
  filter(hiv_status == "positive")

hiv_cases
```

Remarquez que nous avons maintenant un sous-ensemble de données de **3923** lignes et **3** colonnes où tous les individus sont **VIH positifs** !

Inspection des Données

Maintenant, avant de passer à la création de notre pyramide démographique, examinons les données en créant un tableau récapitulatif des colonnes `age_group` et `sex` !

Pour cette étape, nous utiliserons `count()` de `{dplyr}`.

```
hiv_cases %>%
  count(age_group, sex)
```

Nous pouvons voir que les données sont propres et que la colonne `age_group` est correctement organisée dans l'ordre croissant (du plus jeune au plus vieux).

WATCH OUT

Vérifiez toujours l'ordre de vos données avant de tracer un diagramme pyramidal démographique !

WATCH OUT



Avant de créer votre pyramide démographique, assurez-vous que vos données sont propres et correctement organisées dans un **ordre croissant** ! Cela est important lors de l'utilisation de variables catégorielles car l'ordre de votre `age_group` affectera l'ordre dans lequel il sera tracé dans votre pyramide.

Dans le cas des pyramides démographiques, nous voulons que le groupe d'âge le plus jeune soit situé en bas de l'axe des y et le groupe d'âge le plus vieux en haut de l'axe des y.

Création d'un sous-ensemble de données agrégées

Avant de commencer, nous devons créer un cadre de données agrégées qui calcule le nombre total de cas par groupe d'âge, divisé par sexe. Nous voulons que le cadre de données agrégées ressemble à ceci :

# A tibble: 28 × 5					
	age_group	sex	total	axis_counts	axis_percent
	<chr>	<chr>	<int>	<int>	<dbl>
1	00-04	female	3	3	0.1
2	00-04	male	6	-6	-0.2
3	05-09	female	8	8	0.2
4	05-09	male	10	-10	-0.3
5	10-14	female	42	42	1.1
6	10-14	male	23	-23	-0.6
7	15-19	female	51	51	1.3
8	15-19	male	46	-46	-1.2
9	20-24	female	107	107	2.7
10	20-24	male	44	-44	-1.1

(Nous avons 14 groupes d'âge et deux sexes, soit 28 lignes. Nous calculerons d'abord la somme de chaque groupe - c'est la colonne `total`, puis nous nierons les valeurs masculines et créerons une nouvelle colonne appelée `axis_counts`, puis ajouterons la dernière colonne, `axis_percent`, qui présente les totaux en pourcentage. Ils sont appelés `axis counts` car c'est purement à des fins de traçage. Nous n'avons pas réellement -6 cas, ce n'est pas possible.)

La raison pour laquelle les valeurs *masculines* sont niées est de tracer les barres masculines sur le *côté gauche* de l'axe !

KEY POINT

Lors de l'utilisation de la fonction `geom_col()`, le compte de chaque groupe doit être préalablement calculé et spécifié dans `aes()` comme variable x ou y. En d'autres termes, vous devrez convertir les données de la liste en un tableau récapitulatif avec le nombre agrégé d'occurrences pour chaque niveau catégorique. Si vos données sont pré-agrégées, vous pouvez sauter cette étape d'agrégation.

Commençons par utiliser `{dplyr}` pour créer le cadre de données résumées, avec les 28 lignes et 5 colonnes que nous avons vues ci-dessus ! Utilisons les fonctions `group_by()` et `summarise()` pour cela.

REMINDER

N'oubliez pas de nier les comptes masculins afin d'obtenir le graphique des barres masculines du côté gauche du graphique ! Nous pouvons le faire avec la fonction `mutate()`.

# A tibble: 28 × 5					
	age_group	sex	total	axis_counts	axis_percent
	<chr>	<chr>	<int>	<int>	<dbl>
1	00-04	female	3	3	0.1
2	00-04	male	6	-6	-0.2
3	05-09	female	8	8	0.2
4	05-09	male	10	-10	-0.3
5	10-14	female	42	42	1.1
6	10-14	male	23	-23	-0.6
7	15-19	female	51	51	1.3
8	15-19	male	46	-46	-1.2
9	20-24	female	107	107	2.7
10	20-24	male	44	-44	-1.1

```

# Créer un nouveau sous-ensemble
pyramid_data <-
  hiv_cases %>%

# Compter les cas totaux par groupe d'âge et par sexe
count(age_group, sex, name = "total") %>%

# Créer de nouvelles colonnes pour les valeurs de l'axe x sur le graphique
mutate(
  # Nouvelle colonne avec les valeurs de l'axe - convertir les comptes
  # masculins en négatifs
  axis_counts = ifelse(sex == "male", -total, total),
  # Nouvelle colonne pour les valeurs de l'axe en pourcentage
  axis_percent = round(100 * (axis_counts / nrow(hiv_cases)),
  digits = 1))

head(pyramid_data)

```

Notez que les valeurs masculines dans les colonnes `axis_counts` et `axis_percent` sont *négatives* !

Maintenant que les données sont résumées et dans le format approprié, nous pouvons utiliser notre nouveau cadre de données `pyramid_data` pour tracer la pyramide avec `{ggplot2}` !



Testons votre compréhension avec la question à choix multiples suivante :

- 1. Lors de la préparation des données pour le traçage avec `geom_col()`, quelle modification doit être apportée aux valeurs de comptage ?**
 - a. Tous les comptes totaux doivent être niés.
 - b. Les comptes doivent être multipliés par 2.
 - c. Les comptes doivent être convertis en pourcentages.
 - d. Les comptes masculins doivent être niés (multipliés par -1).

La clé de réponse peut être trouvée à la fin de ce document.

Création du graphique

Comme mentionné précédemment, un pyramide démographique peut être considéré comme une version modifiée d'un graphique à barres empilées.

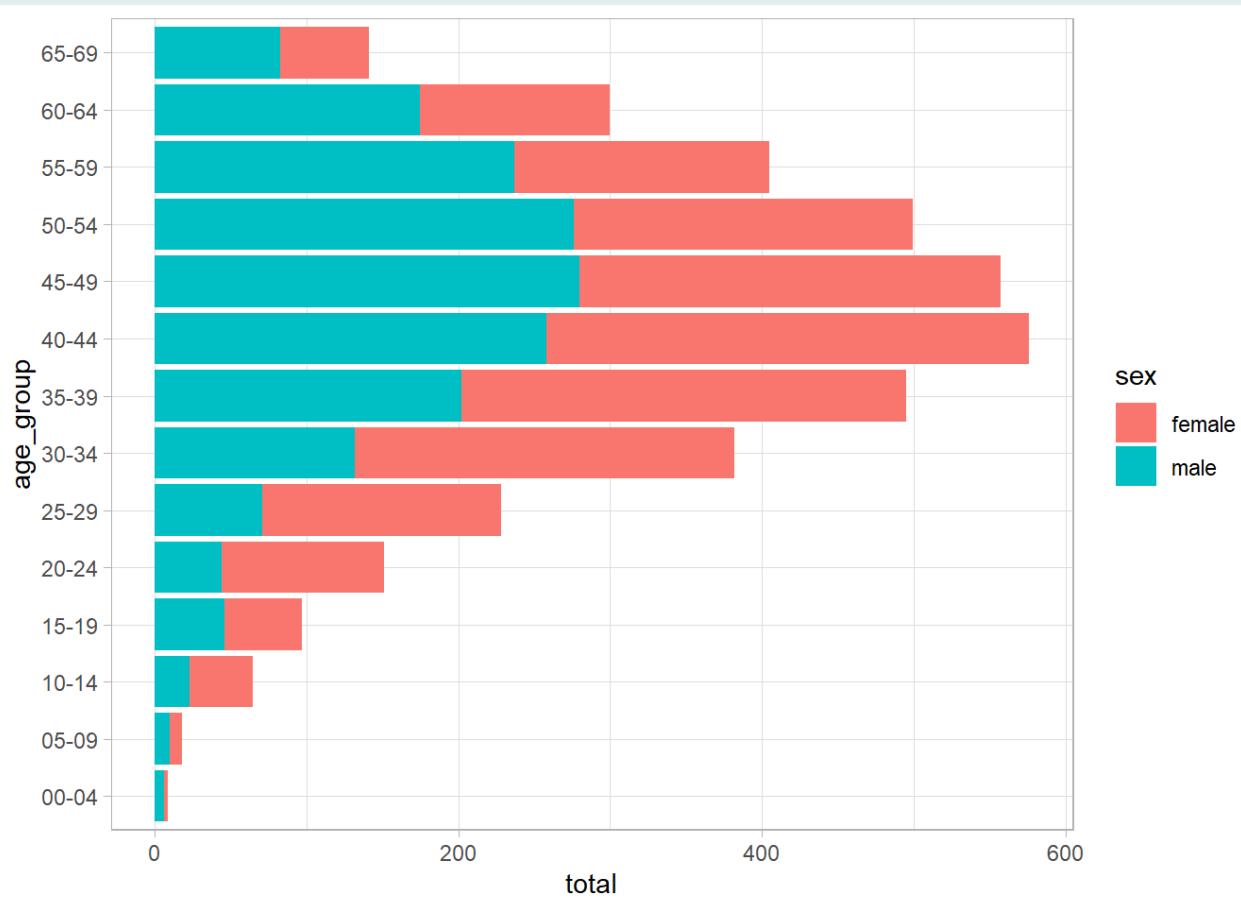
Pour créer un graphique à barres empilées de base avec `geom_col()`, nous traçons une variable catégorielle (par exemple, `age_group`) par rapport à une variable continue (par exemple, `total`), et nous définissons la couleur de remplissage (`fill`) sur une deuxième variable catégorielle (par exemple, `sex`).

```
# Graphique à barres empilées de base

# Commencer ggplot
ggplot() +

# Créer un graphique à barres avec geom_col()
  geom_col(data = pyramid_data,      # spécifier les données à graphiquer
            aes(x = age_group,       # indiquer la variable x catégorielle
                y = total,           # indiquer la variable y continue
                fill = sex)) +     # remplir par la deuxième variable
            catégorielle
# Modifier le thème
  theme_light() +

# Inverser les axes X et Y
  coord_flip()
```



Ici, nous avons utilisé la variable `total` de `pyramid_data`, où toutes les valeurs sont positives.

	age_group	sex	total	axis_counts	axis_percent
	<chr>	<chr>	<int>	<int>	<dbl>
1	00-04	female	3	3	0.1
2	00-04	male	6	-6	-0.2
3	05-09	female	8	8	0.2
4	05-09	male	10	-10	-0.3
5	10-14	female	42	42	1.1
6	10-14	male	23	-23	-0.6
7	15-19	female	51	51	1.3
8	15-19	male	46	-46	-1.2
9	20-24	female	107	107	2.7
10	20-24	male	44	-44	-1.1

Pour utiliser la fonction `geom_col()` pour un graphique à barres empilées, votre ensemble de données doit inclure les sommes totales agrégées par chaque variable catégorielle (`age_group` et `sex`) !

RECAP



Les bases de la création de graphiques à barres avec `{ggplot2}` sont couvertes dans notre cours introductif sur la visualisation des données, Data on Display. Des applications supplémentaires des graphiques à barres pour la déclaration d'épidémies sont enseignées dans notre leçon sur la visualisation des comparaisons et des compositions. Tout le contenu est disponible sur notre [site web](#).

Utilisation de `geom_col()` pour les pyramides démographiques

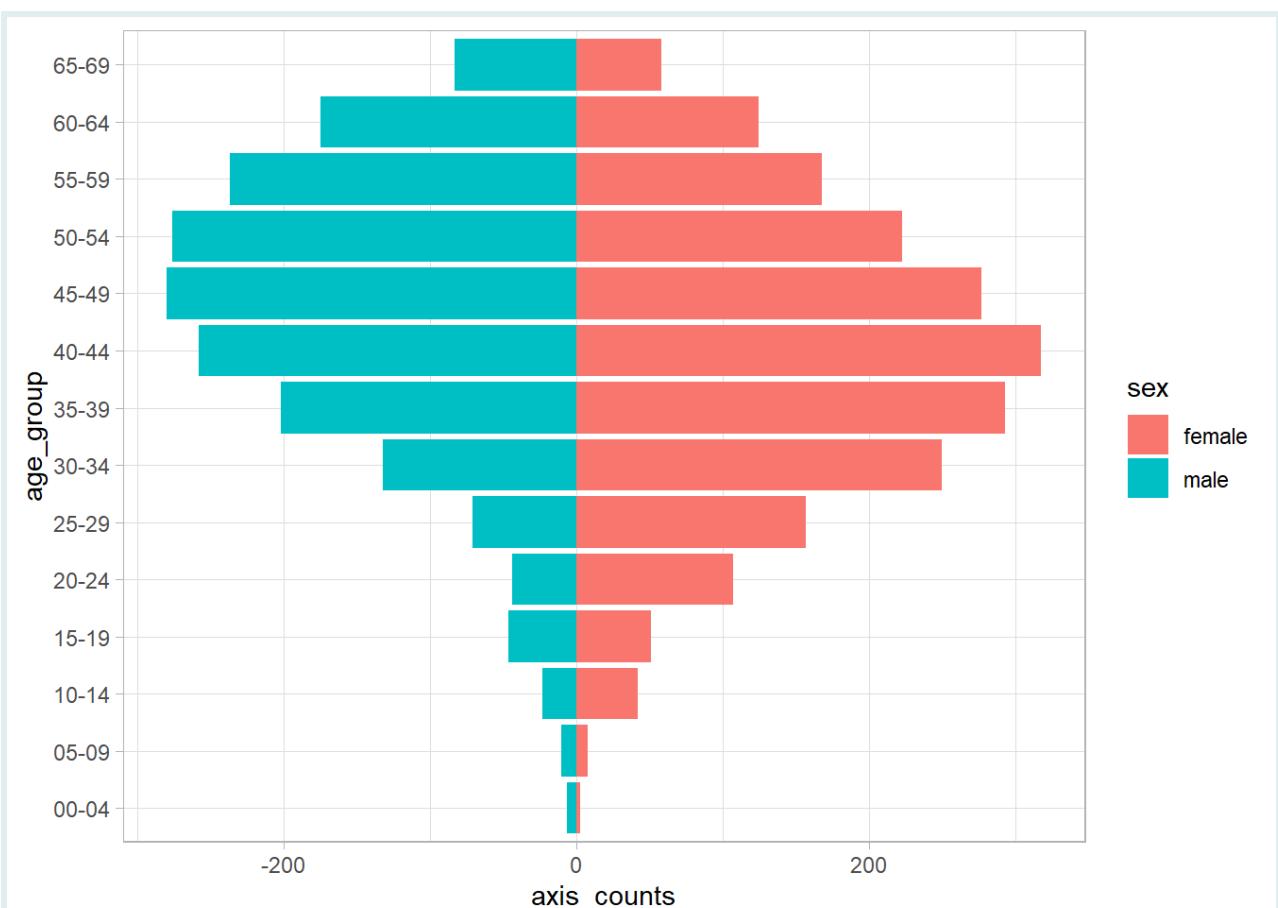
Maintenant, nous allons construire sur le code de base du graphique à barres empilées ci-dessus pour créer une pyramide démographique.

Cette fois, nous utilisons `axis_counts` pour l'axe des y, qui contient des valeurs négatives pour les hommes.

	age_group	sex	total	axis_counts	axis_percent
	<chr>	<chr>	<int>	<int>	<dbl>
1	00-04	female	3	3	0.1
2	00-04	male	6	-6	-0.2
3	05-09	female	8	8	0.2
4	05-09	male	10	-10	-0.3
5	10-14	female	42	42	1.1
6	10-14	male	23	-23	-0.6
7	15-19	female	51	51	1.3
8	15-19	male	46	-46	-1.2
9	20-24	female	107	107	2.7
10	20-24	male	44	-44	-1.1

```
demo_pyramid <-
  ggplot() +
    geom_col(data = pyramid_data, # spécifier les données à graphiquer
              aes(
                x = age_group,      # indiquer la variable x
                y = axis_counts,   # indiquer la variable y NÉGATIVE
                fill = sex)) +    # remplir par le sexe
    theme_light() +
    coord_flip()

demo_pyramid
```



KEY POINT



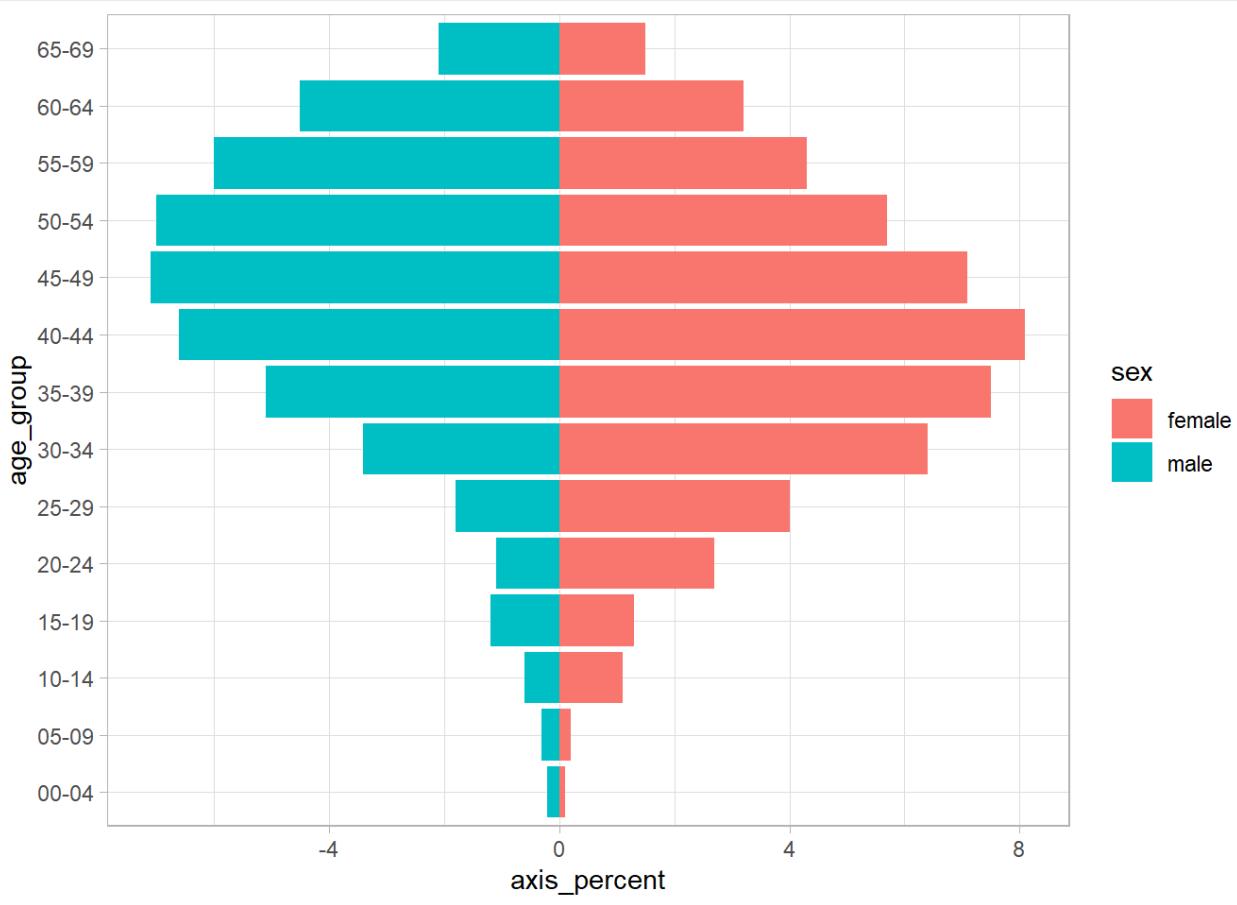
Une pyramide démographique est une version d'un graphique à barres empilées, avec l'alignement horizontal des barres ajusté le long de l'axe x.

Nous pouvons également créer la même pyramide démographique en utilisant les pourcentages totaux sur notre axe des y.

	age_group	sex	total	axis_counts	axis_percent
	<chr>	<chr>	<int>	<int>	<dbl>
1	00-04	female	3	3	0.1
2	00-04	male	6	-6	-0.2
3	05-09	female	8	8	0.2
4	05-09	male	10	-10	-0.3
5	10-14	female	42	42	1.1
6	10-14	male	23	-23	-0.6
7	15-19	female	51	51	1.3
8	15-19	male	46	-46	-1.2
9	20-24	female	107	107	2.7
10	20-24	male	44	-44	-1.1

```
demo_pyramid_percent <-
  ggplot() +
  geom_col(data = pyramid_data,
            aes(x = age_group,
                 y = axis_percent, # utiliser les pourcentages précalculés
                 fill = sex)) +
  coord_flip() +
  theme_light()

demo_pyramid_percent
```

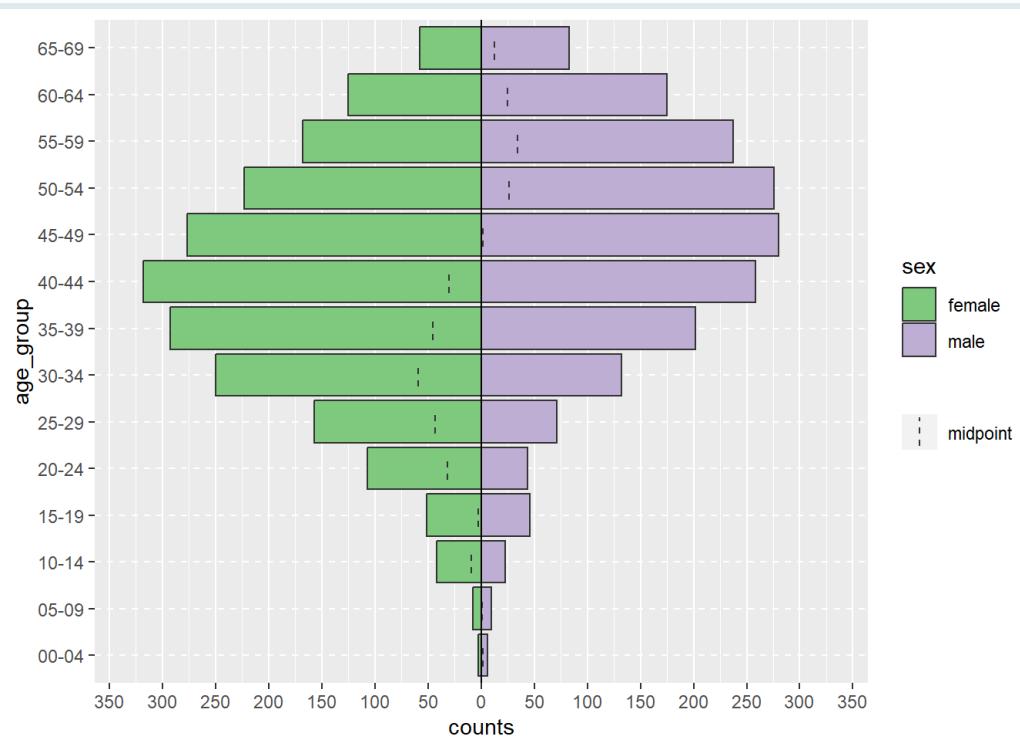


SIDE NOTE



Plusieurs packages sont disponibles pour faciliter l'analyse de données et la visualisation de données. Dans le cas des pyramides démographiques, le package `{apyramid}` peut être un outil utile. Ce package du projet **R4Epis** contient une fonction `age_pyramid()`, qui permet la création rapide de pyramides démographiques :

```
hiv_cases %>% # peut utiliser la liste originale
  # La variable de regroupement doit être un facteur
  mutate(age_group = factor(age_group)) %>%
  apyramid::age_pyramid(# 2 arguments requis :
    age_group = "age_group",
    split_by = "sex")
```

SIDE NOTE

Cependant, ce package a des fonctionnalités limitées et peu d'options de personnalisation. `{ggplot2}` offre une approche beaucoup plus polyvalente et utilise une syntaxe avec laquelle nous sommes déjà familiers.

Des informations supplémentaires sur la fonction peuvent être lues [ici](#) ou en entrant `?age_pyramid` dans votre console R.

PRACTICE

(in RMD)

Testons votre compréhension avec les questions à choix multiples suivantes (la clé des réponses se trouve à la fin) :

2. Lors de l'utilisation de `geom_col()`, quel type de variable x votre ensemble de données doit-il inclure ?
 - a. Variables continues
 - b. Variables catégorielles
 - c. Variables binaires
 - d. Variables ordinaires
3. Quelle fonction de `{ggplot2}` pouvez-vous utiliser pour inverser les axes x et y ?
 - a. `coord_flip()`
 - b. `x_y_flip()`
 - c. `geom_flip()`



d. Toutes les options ci-dessus

Maintenant, testons votre compréhension avec la question de pratique suivante :

Nous allons utiliser un ensemble de données nettoyé et préparé contenant la population totale du Zimbabwe en 2016 regroupée par groupe d'âge et par sexe.

Commencez par charger l'ensemble de données préparé comme suit :

```
zw_2016 <- readRDS(here::here("data/population_zw_2016.rds"))

zw_2016
```



```
## # A tibble: 28 × 3
## # Groups:   age_group [14]
##       age_group sex    total_count
##       <fct>     <fct>      <int>
## 1 0-4        Female    1091129
## 2 0-4        Male     -1114324
## 3 10-14      Female    815362
## 4 10-14      Male     -803657
## 5 15-19      Female    803754
## 6 15-19      Male     -792474
## 7 20-24      Female    690940
## 8 20-24      Male     -632342
## 9 25-29      Female    638192
## 10 25-29     Male     -535444
## # ... 18 more rows
```

Notez que le total des hommes est déjà négatif !

5. Créez une pyramide démographique pour la population totale du Zimbabwe en 2016 en utilisant la fonction `geom_col()` du package `ggplot2`. Assurez-vous d'ajouter une bordure blanche autour de chaque barre !

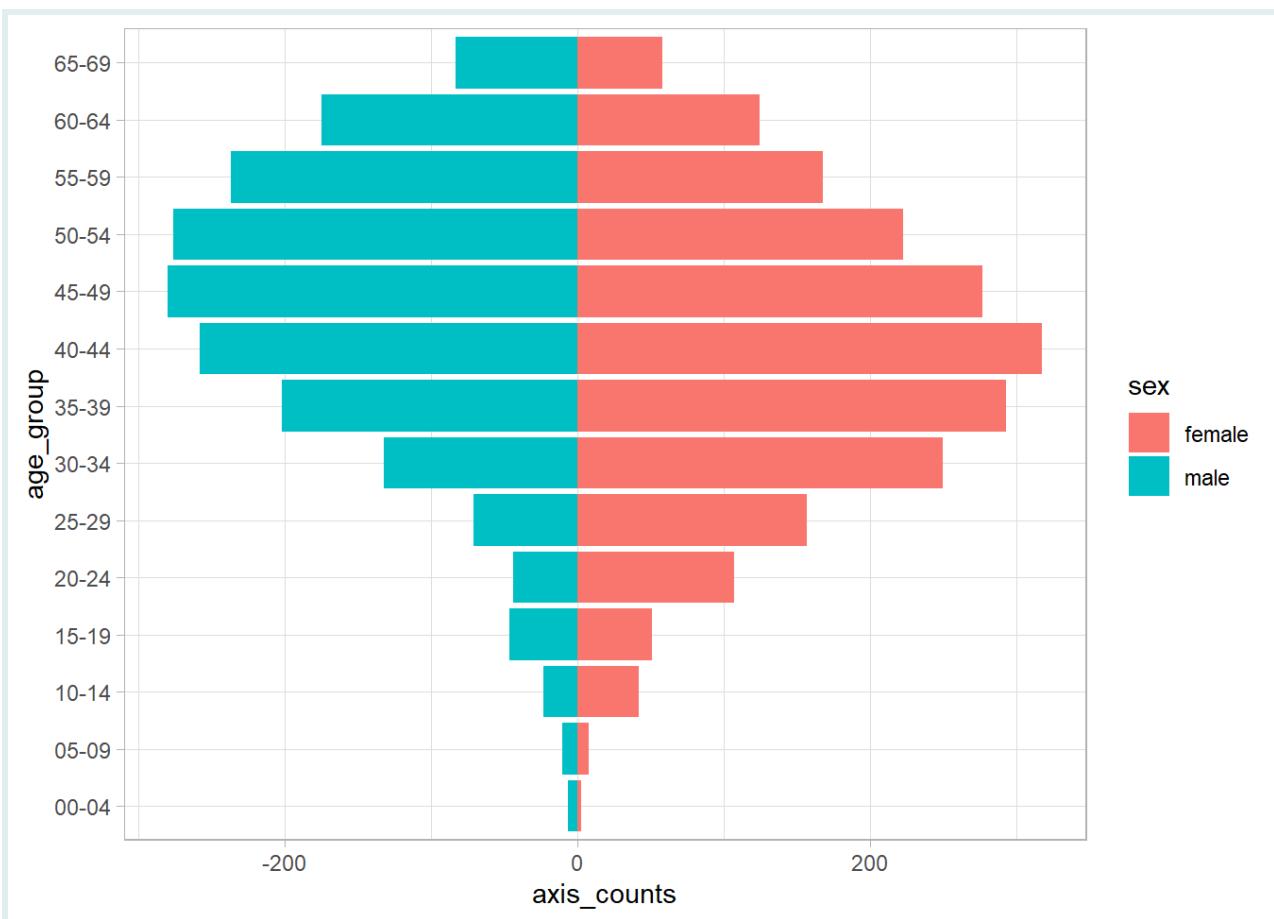


```
Q4_pyramid_zw_2016 <-
# Commencer ggplot
ggplot() +
# Créer un graphique à barres avec geom_bar
geom_col(data = ____,
          aes(x = ____,
              y = ____,
              fill = ____),
          color = ____) +
# Inverser les axes x et y
_____
```

Personnalisation du Graphique

Jusqu'à présent, vous avez appris comment créer un pyramide démographique en utilisant `{ggplot2}` comme illustré ci-dessous :

```
demo_pyramid
```



Cependant, afin de créer un graphique informatif, un certain niveau de personnalisation du graphique est nécessaire. Par exemple, il est important d'inclure des étiquettes informatives et de redimensionner les axes x et y pour une meilleure visualisation.

Apprenons quelques personnalisations utiles de `{ggplot2}` !

Axes

Les paramètres actuels des axes résultent en une représentation asymétrique, avec le côté droit (féminin) s'étendant plus loin que le côté gauche (masculin) en raison d'un nombre de cas plus élevé dans le plus grand groupe d'âge féminin. Pour une comparaison précise, il est essentiel que les deux côtés se reflètent symétriquement.

Nous allons ajuster les limites des axes pour qu'elles soient égales en valeurs positives et négatives, garantissant ainsi que les données soient visualisées de manière symétrique pour une comparaison claire et équilibrée.

Nous commencerons par redimensionner l'axe du *nombre total*, ou dans le cas de notre graphique, l'**axe y**. Pour cela, nous commencerons par identifier la valeur maximale et la sauvegarder comme un objet.

```
max_count <- max(pyramid_data$total)  
max_count
```

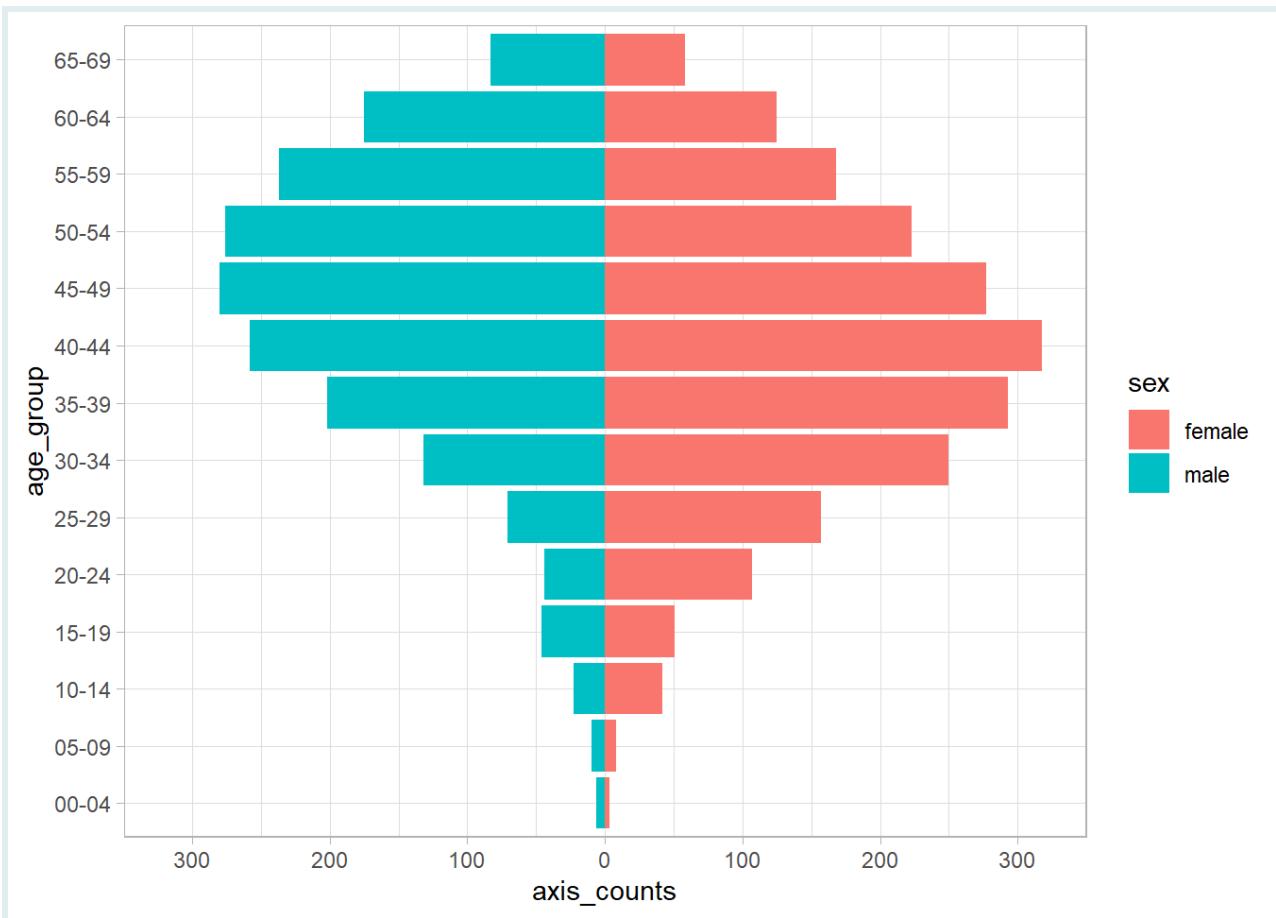
```
## [1] 318
```

Maintenant que nous avons identifié que la valeur maximale pour le *nombre total* est **318**, nous pouvons l'utiliser pour redimensionner notre axe y en conséquence.

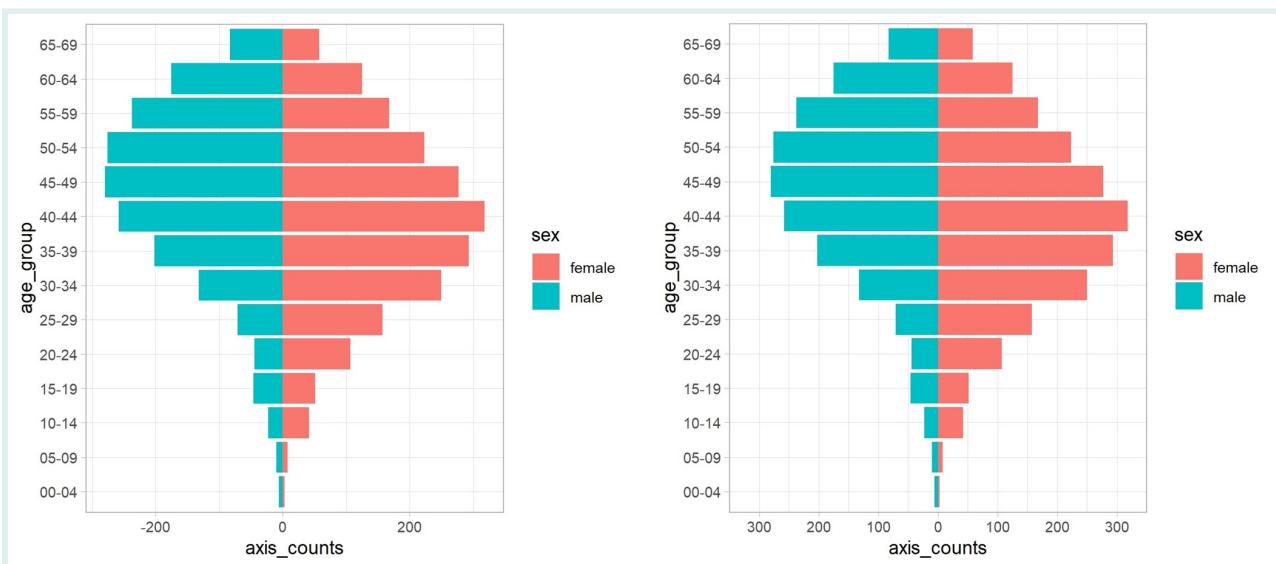
Dans ce cas particulier, nous voulons redimensionner notre axe y de manière symétrique. Nous prendrons donc la plus grande valeur absolue et l'utiliserons comme limite pour les côtés *positif* et *négatif*.

Dans ce cas, nous utiliserons notre valeur maximale.

```
custom_axes <-  
  
# Utiliser le graphique précédent  
demo_pyramid +  
  
# Ajuster l'axe y (nombre total)  
scale_y_continuous(  
  
  # Spécifier la limite de l'axe y en utilisant la valeur maximale et en la  
  # rendant positive et négative  
  limits = c(-max_count, max_count),  
  
  # Spécifier l'espacement entre les étiquettes d'axe  
  breaks = scales::breaks_width(100),  
  
  # Rendre les étiquettes d'axe absolues pour que les étiquettes masculines  
  # apparaissent positives  
  labels = abs)  
  
custom_axes
```



Ajuster les limites des axes pour qu'elles soient égales des deux côtés garantit une comparaison visuelle symétrique et précise, facilitant l'interprétation correcte des données.



Notez que nous avons également changé les étiquettes d'axe pour qu'elles soient leur valeur absolue, de sorte que les nombres de cas masculins n'apparaissent plus comme des nombres négatifs.

Étiquettes Personnalisées

Utilisons la pyramide de population que nous avons précédemment créée à l'aide de la fonction `geom_col()` et construisons dessus.

Nous pouvons commencer par ajouter un titre informatif, des axes et une légende à notre graphique :

```
custom_labels <-
  # Commencer avec la pyramide démographique précédente
  custom_axes +
  # Ajuster les étiquettes
  labs(
    title = "Cas VIH Positifs par Âge et Sexe",
    subtitle = "Zimbabwe (2016)",
    x = "Groupe d'Âge",
    y = "Nombre",
    fill = "Sexe",
    caption = stringr::str_glue("Les données proviennent de données de liste
      de simulation \nn = {nrow(hiv_cases)}"))
  custom_labels
```

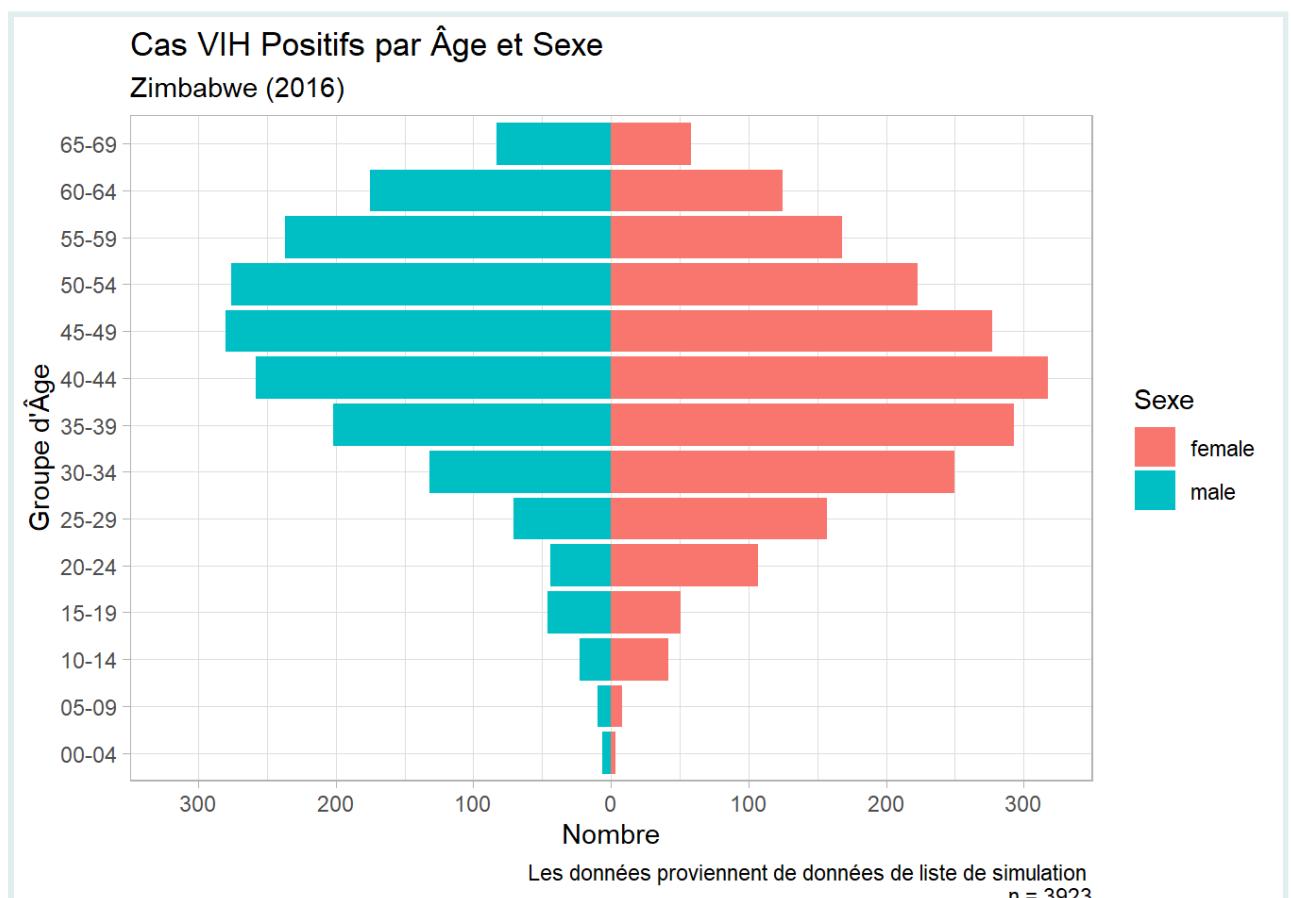


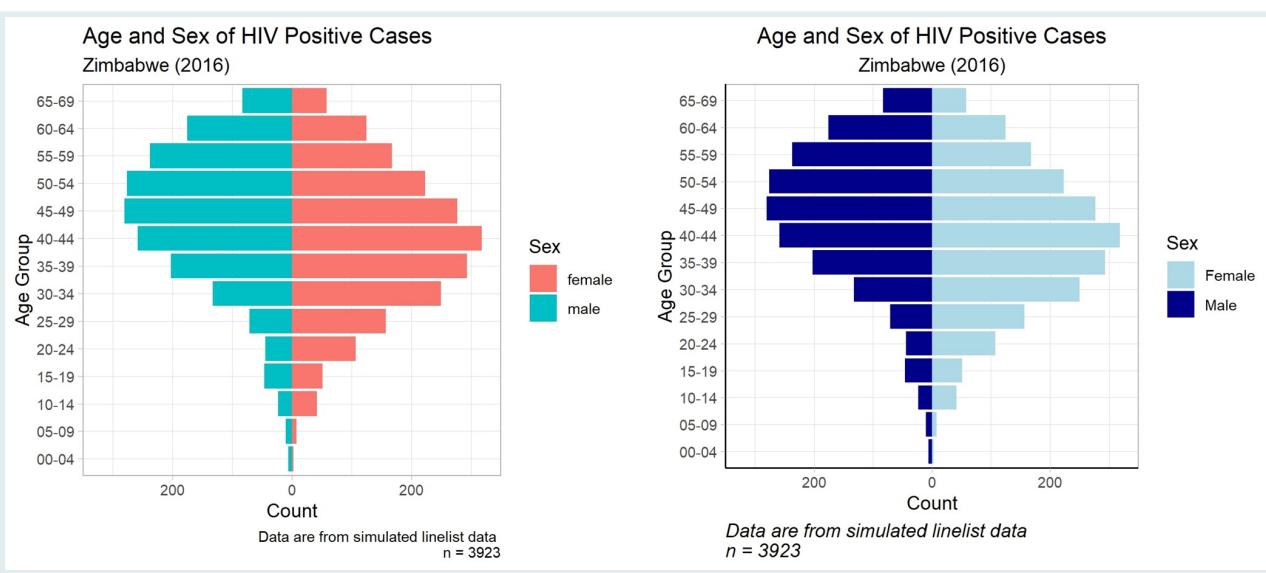
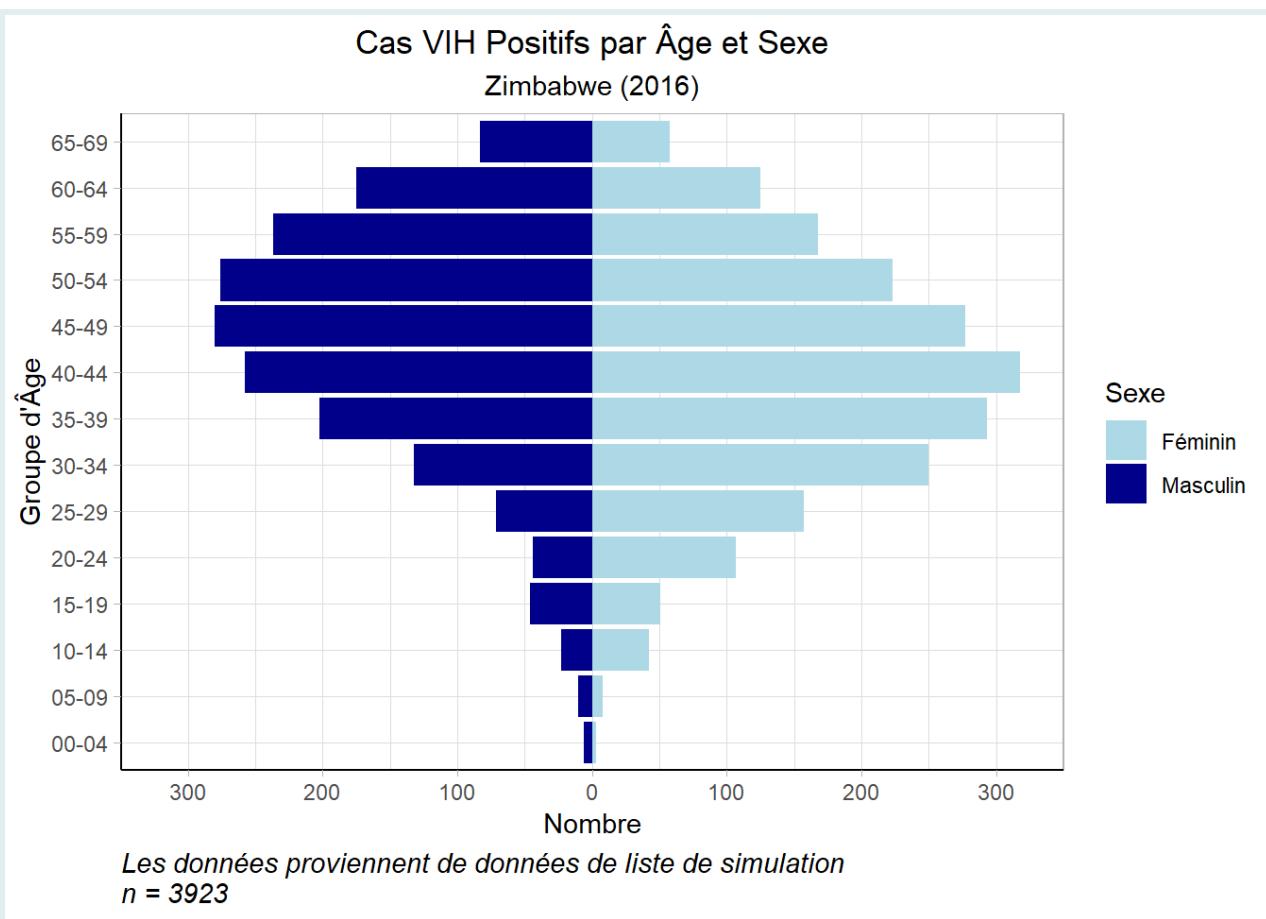
Schéma de Couleurs et les Thèmes

Nous pouvons également apporter des ajustements nécessaires au schéma de couleurs et au thème du graphique.

Ci-dessous est un exemple de certains changements que nous pouvons effectuer :

```
custom_color_theme <-
# Utiliser le graphique précédent
custom_labels +
# Désigner manuellement les couleurs et les légendes
scale_fill_manual(
  # Sélectionner la couleur de remplissage du sexe
  values = c("female" = "lightblue",
             "male" = "darkblue"),
  # Mettre en majuscules les légendes
  labels = c("Féminin", "Masculin")) +
# Ajuster les paramètres du thème
theme(
  axis.line = element_line(colour = "black"), # rendre la ligne d'axe noire
  plot.title = element_text(hjust = 0.5), # centrer le titre
  plot.subtitle = element_text(hjust = 0.5), # centrer le sous-titre
  plot.caption = element_text(hjust = 0, # formater le texte de la
                               légende
                               size = 11,
                               face = "italic"))

custom_color_theme
```



- titre et sous-titre centrés
- légende à gauche et augmentation de la taille de la police
- texte de légende en majuscules
- couleurs
- lignes d'axe en gras

En Résumé !

Comme vous pouvez le voir, les pyramides démographiques sont un outil de visualisation essentiel pour comprendre la répartition de maladies spécifiques selon les groupes d'âge et le sexe.

Les concepts appris dans cette leçon peuvent également être appliqués pour créer d'autres types de graphiques nécessitant des résultats à la fois négatifs et positifs, tels que le pourcentage de changement dans les taux de notification de cas, et bien plus encore.

Maintenant que vous avez compris le concept de création de pyramides démographiques, les possibilités sont infinies ! De la représentation graphique des **cas** par groupe d'âge et sexe sur la population **de référence/réelle**, à la représentation graphique des changements (positifs et négatifs) des interventions dans une population, vous devriez être en mesure d'appliquer ces concepts pour créer des graphiques épidémiologiques informatifs.

Félicitations pour avoir terminé cette leçon. Nous espérons que vous pourrez maintenant appliquer les connaissances acquises lors de la leçon d'aujourd'hui lors de l'analyse et de la création de rapports d'examen épidémiologique.

Solutions

1. d
2. b
3. a
4. c
- 5.

```
Q4_pyramid_zw_2016 <-
  ggplot() +
  geom_col(data = zw_2016,
            aes(x = age_group,
                 y = total_count,
                 fill = sex),
            color = "white") +
  coord_flip()
```

Contributors

Les membres de l'équipe suivants ont contribué à cette leçon :



SABINA RODRIGUEZ VELÁSQUEZ

Project Manager and Scientific Collaborator, The GRAPH Network
Infectiously enthusiastic about microbes and Global Health



JOY VAZ

R Developer and Instructor, the GRAPH Network
Loves doing science and teaching science

References

1. Contenu de la leçon adapté de : Batra, Neale, et al. Le Manuel de l'épidémiologiste R. 2021. <https://doi.org/10.5281/zenodo.4752646>
2. Contenu de la leçon adapté de : OMS. Comprendre et utiliser les données sur la tuberculose. 2014. https://apps.who.int/iris/bitstream/handle/10665/129942/9789241548786_eng.pdf
3. Package référencé à partir de : <https://r4epis.netlify.app/>

Présentation des données de santé avec les tableaux gt dans R: Fondamentaux

Introduction
Packages
Présentation du jeu de données
Création de tables simples avec {gt}
Personnalisation des tableaux {gt}
En-tête et pied de tableau
Stub
Colonnes Spanner & sous-colonnes
Renommage des colonnes du tableau
Lignes de résumé
Conclusion

Introduction

Les tables sont un outil puissant pour visualiser les données de manière claire et concise. Avec R et le package `gt`, nous pouvons exploiter l'attrait visuel des tables pour communiquer efficacement des informations clés. Dans cette leçon, nous apprendrons comment construire des tables esthétiquement agréables et personnalisables qui soutiennent les objectifs d'analyse de données.

Objectifs d'apprentissage

- Utiliser la fonction `gt()` pour créer des tables basiques
- Regrouper les colonnes sous des en-têtes spanner
- Renommer les noms des colonnes
- Ajouter des lignes récapitulatives pour les groupes

À la fin, vous serez capable de générer des tables polies et reproductibles comme celle-ci :

Sum of HIV Tests in Malawi

from Q1 2019 to Q4 2019

	New tests		Previous tests	
	Positive	Negative	Positive	Negative
Central Region				
2019 Q1	2004	123018	3682	2562
2019 Q2	1913	116443	3603	1839
2019 Q3	1916	127799	4002	2645
2019 Q4	1691	124728	3754	1052
sum	7524.00	491988.00	15041.00	8098.00
mean	1881.00	122997.00	3760.25	2024.50
Northern Region				
2019 Q1	664	36196	1197	675
2019 Q2	582	35315	1084	590
2019 Q3	570	36850	1191	542
2019 Q4	519	34322	1132	346
sum	2335.00	142683.00	4604.00	2153.00
mean	583.75	35670.75	1151.00	538.25

Tableau récapitulatif exemple

Packages

Nous utiliserons ces packages :

- {gt} pour créer des tables
- {tidyverse} pour la manipulation des données
- {here} pour les chemins de fichiers

```
# Charger les packages
pacman::p_load(tidyverse, gt, here)
```

Présentation du jeu de données

Nos données proviennent du ** Programme VIH du Malawi ** et couvrent les soins prénataux et le traitement du VIH en 2019. Nous nous concentrerons sur les agrégats régionaux et au niveau des installations par trimestre (disponibles [ici](#)).

```
# Importer les données
hiv_malawi <- read_csv(here::here("data/clean/hiv_malawi.csv"))
```

Explorons les variables :

```
# Les 6 premières lignes
head(hiv_malawi)
```

```
## # A tibble: 6 × 29
##   region           zone      district traditional_autho...¹
##   <chr>            <chr>     <chr>    <chr>
## 1 Northern Region Northern Zone Chitipa Senior TA Bulambya...
## 2 Northern Region Northern Zone Chitipa Senior TA Bulambya...
## 3 Northern Region Northern Zone Chitipa Senior TA Bulambya...
## 4 Northern Region Northern Zone Chitipa Senior TA Bulambya...
## 5 Northern Region Northern Zone Chitipa Senior TA Bulambya...
## 6 Northern Region Northern Zone Chitipa Senior TA Bulambya...
## # i abbreviated name: ¹traditional_authority
## # i 25 more variables: facility_name <chr>, ...
```

```
# Les noms et les types de variables
glimpse(hiv_malawi)
```

```

## Rows: 17,235
## Columns: 29
## $ region <chr> "Northern R...
## $ zone <chr> "Northern Z...
## $ district <chr> "Chitipa", ...
## $ traditional_authority <chr> "Senior TA ...
## $ facility_name <chr> "Kapenda He...
## $ datim_code <chr> "K9u9BIAaJJ...
## $ system <chr> "e-masterca...
## $ hsector <chr> "Public", ...
## $ period <chr> "2019 Q1", ...
## $ reporting_period <chr> "1st month ...
## $ sub_groups <chr> "All patien...
## $ new_women_registered <dbl> 45, NA, 40, ...
## $ total_women_in_booking_cohort <dbl> NA, 55, NA, ...
## $ not_tested_for_syphilis <dbl> NA, 45, NA, ...
## $ syphilis_negative <dbl> NA, 10, NA, ...
## $ syphilis_positive <dbl> NA, 0, NA, ...
## $ hiv_status_not_ascertained <dbl> 4, 7, 9, 4, ...
## $ previous_negative <dbl> 0, 0, 0, 0, ...
## $ previous_positive <dbl> 0, 0, 0, 1, ...
## $ new_negative <dbl> 40, 47, 30, ...
## $ new_positive <dbl> 1, 1, 1, 1, ...
## $ not_on_cpt <dbl> NA, 0, NA, ...
## $ on_cpt <dbl> NA, 1, NA, ...
## $ no_ar_vs <dbl> 0, 0, 0, 0, ...
## $ already_on_art_when_starting_anc <dbl> 0, 1, 0, 1, ...
## $ started_art_at_0_27_weeks_of_pregnancy <dbl> 1, 0, 1, 1, ...
## $ started_art_at_28_weeks_of_preg <dbl> 0, 0, 0, 0, ...
## $ no_ar_vs_dispensed_for_infant <dbl> NA, 0, NA, ...
## $ ar_vs_dispensed_for_infant <dbl> NA, 1, NA, ...

```

Les données couvrent les régions géographiques, les établissements de santé, les périodes de temps, les données démographiques des patients, les résultats des tests, les thérapies préventives, les médicaments antirétroviraux, et plus encore. Plus d'informations sur le jeu de données sont dans la section des annexes.

Les variables clés que nous examinerons sont :

1. **previous_negative**: Le nombre de patients qui ont visité l'établissement de santé au cours de ce trimestre et qui avaient auparavant des tests VIH négatifs.
2. **previous_positive**: Le nombre de patients (comme ci-dessus) avec des tests VIH positifs précédents.
3. **new_negative**: Le nombre de patients testant nouvellement négatif pour le VIH.
4. **new_positive**: Le nombre de patients testant nouvellement positif pour le VIH.

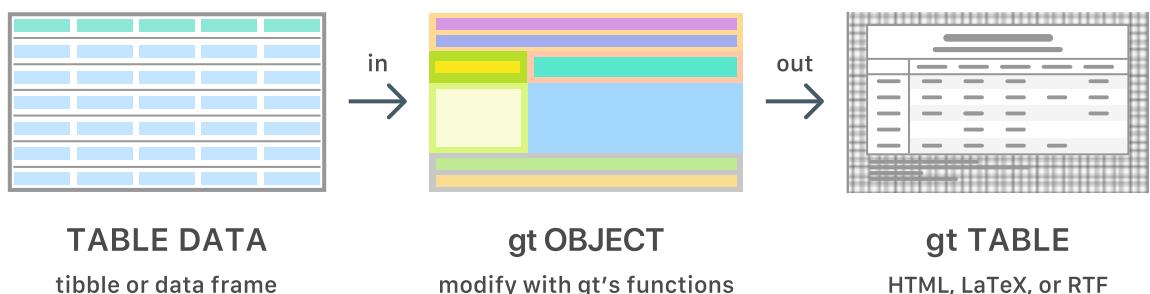
Dans cette leçon, nous allons agréger les données par trimestre et résumer les changements dans les résultats des tests VIH.

Création de tables simples avec {gt}

La flexibilité, l'efficacité et la puissance de {gt} en font un package redoutable pour la création de tables dans R. Nous explorerons certaines de ses principales caractéristiques dans cette leçon.



A Typical gt Workflow



Le package {gt} contient un ensemble de fonctions qui prennent des données brutes en entrée et produisent une table joliment formatée pour une analyse et un rapport ultérieurs.

Pour utiliser efficacement le package {gt}, nous devons d'abord transformer nos données dans un format résumé approprié.

Dans le bloc de code ci-dessous, nous utilisons les fonctions de {dplyr} pour résumer les tests de VIH dans certains centres de dépistage du Malawi par trimestre. Nous regroupons d'abord les données par période, puis nous additionnons les cas sur plusieurs variables en utilisant `across()` :

```
# Variables à résumer
cols <- c("new_positive", "previous_positive", "new_negative",
"previous_negative")

# Créer un résumé par trimestre
hiv_malawi_summary <- hiv_malawi %>%
  group_by(period) %>%
  summarize(
    across(all_of(cols), sum) # Résumer toutes les colonnes
  )

hiv_malawi_summary
```

```

## # A tibble: 4 × 5
##   period new_positive previous_positive new_negative
##   <chr>      <dbl>            <dbl>        <dbl>
## 1 2019 Q1     6199            14816       284694
## 2 2019 Q2     6132            15101       282249
## 3 2019 Q3     5907            15799       300529
## 4 2019 Q4     5646            15700       291622
## # i 1 more variable: previous_negative <dbl>

```

Ceci agrège les données de manière appropriée pour les passer à `{gt}` afin de générer un tableau résumé propre.

Pour créer une table simple à partir des données agrégées, nous pouvons ensuite appeler la fonction `gt()` :

```

hiv_malawi_summary %>%
  gt()

```

period	new_positive	previous_positive	new_negative	previous_negative
2019 Q1	6199	14816	284694	6595
2019 Q2	6132	15101	282249	5605
2019 Q3	5907	15799	300529	6491
2019 Q4	5646	15700	291622	6293

Comme vous pouvez le voir, le formatage de table par défaut est assez simple et non raffiné. Cependant, `{gt}` offre de nombreuses options pour personnaliser et embellir la sortie de table. Nous approfondirons ces aspects dans la prochaine section.

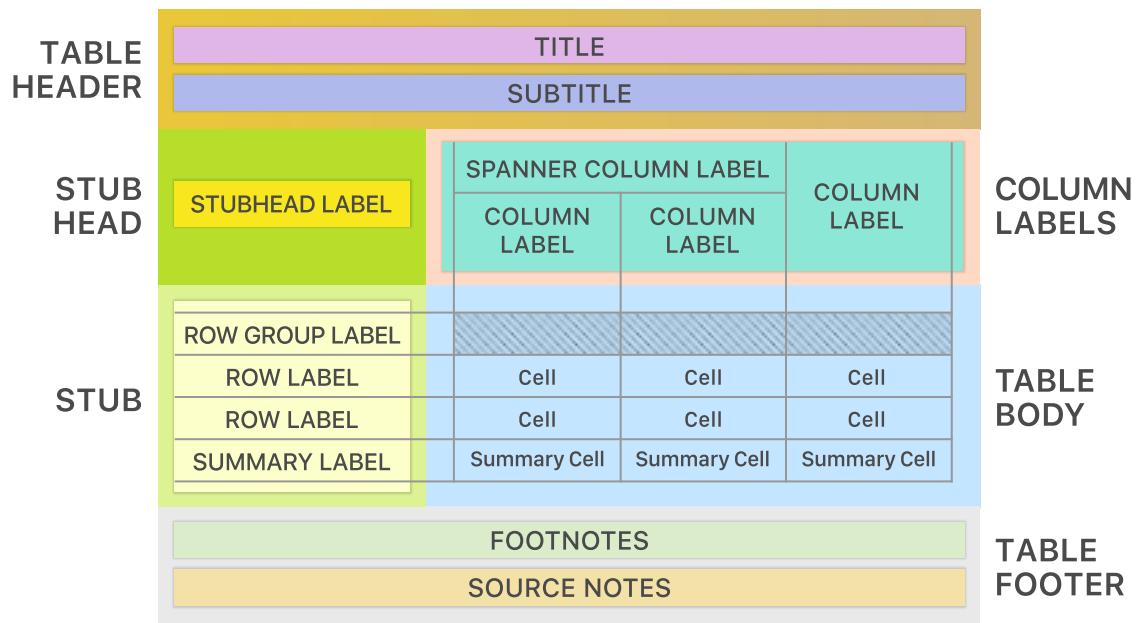
Voici la traduction en français du fragment de tutoriel Rmd, en gardant la syntaxe Rmd et le code valide :

Personnalisation des tableaux `{gt}`

Le package `{gt}` permet une personnalisation complète des tableaux grâce à son cadre de « grammaire des tableaux ». C'est similaire à la façon dont la grammaire graphique de `{ggplot2}` fonctionne pour les graphiques.

Pour tirer pleinement parti de `{gt}`, il est utile de comprendre certains éléments clés de sa grammaire.

The Parts of a gt Table



Comme on le voit dans la figure du site du package, les principaux composants d'un tableau {gt} sont :

- **En-tête du tableau**: Contient un titre et un sous-titre facultatifs
- **Colonne d'étiquette**: Étiquettes de lignes qui identifient chaque ligne
- **En-tête de colonne d'étiquette**: Regroupement et étiquettes facultatifs pour les lignes de colonne d'étiquette
- **Étiquettes de colonne**: En-têtes pour chaque colonne
- **Corps du tableau**: Les principales cellules de données du tableau
- **Pied de tableau**: Notes de bas de page et notes de source facultatives

Comprendre cette anatomie nous permet de construire systématiquement des tableaux {gt} en utilisant sa grammaire.

En-tête et pied de tableau

Le tableau de base que nous avions peut maintenant être mis à jour avec plus de composants.

Les tableaux deviennent plus informatifs et professionnels avec l'ajout d'en-têtes, de notes de source et de notes de bas de page. Nous pouvons facilement améliorer le tableau de base d'avant en ajoutant ces éléments à l'aide des fonctions {gt}.

Pour créer un en-tête, nous utilisons `tab_header()` et spécifions un `title` et `subtitle`. Cela donne au lecteur le contexte de ce que montre le tableau.

```
hiv_malawi_summary %>%
  gt() %>%
  tab_header(
    title = "Dépistage du VIH au Malawi",
    subtitle = "T1 à T4 2019"
  )
```

period	new_positive	previous_positive	new_negative	previous_negative
2019 Q1	6199	14816	284694	6595
2019 Q2	6132	15101	282249	5605
2019 Q3	5907	15799	300529	6491
2019 Q4	5646	15700	291622	6293

Nous pouvons ajouter un pied de page avec la fonction `tab_source_note()` pour citer la provenance des données :

```
hiv_malawi_summary %>%
  gt() %>%
  tab_header(
    title = "Dépistage du VIH au Malawi",
    subtitle = "T1 à T4 2019"
  ) %>%
  tab_source_note("Source : Programme VIH du Malawi")
```

period	new_positive	previous_positive	new_negative	previous_negative
2019 Q1	6199	14816	284694	6595
2019 Q2	6132	15101	282249	5605
2019 Q3	5907	15799	300529	6491
2019 Q4	5646	15700	291622	6293

Source : Programme VIH du Malawi

Les notes de bas de page sont utiles pour fournir des détails supplémentaires sur certains points de données ou étiquettes. La fonction `tab_footnote()` attache les notes de bas de page aux cellules de tableau indiquées. Par exemple, nous pouvons annoter les colonnes de diagnostic :

```
hiv_malawi_summary %>%
  gt() %>%
  tab_header(
    title = "Dépistage du VIH au Malawi",
    subtitle = "T1 à T2 2019"
  ) %>%
  tab_source_note("Source : Programme VIH du Malawi") %>%
  tab_footnote(
    footnote = "Nouveau diagnostic",
    locations = cells_column_labels(columns = c(new_positive,
new_negative))
  )
```

Dépistage du VIH au Malawi

T1 à T2 2019

period	new_positive ¹	previous_positive	new_negative ¹	previous_negative
2019 Q1	6199	14816	284694	6595
2019 Q2	6132	15101	282249	5605
2019 Q3	5907	15799	300529	6491
2019 Q4	5646	15700	291622	6293

¹ Nouveau diagnostic

Source : Programme VIH du Malawi

Ces petits ajouts améliorent grandement l'apparence professionnelle et informative des tableaux.

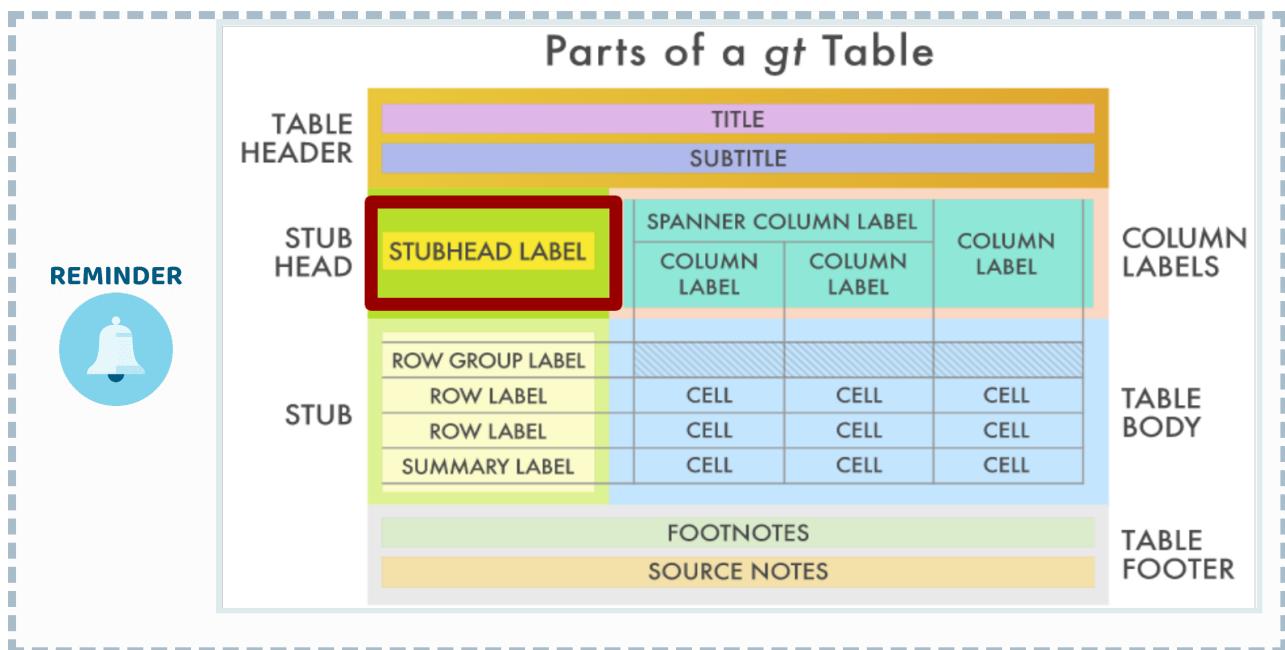
Stub

Le stub est la section gauche d'un tableau contenant les étiquettes de ligne. Elles fournissent un contexte pour les données de chaque ligne.

REMINDER



Cette image montre le composant stub d'un tableau `{gt}`, marqué par un carré rouge.



Dans notre tableau de cas de VIH, la colonne period contient les étiquettes de ligne que nous voulons utiliser. Pour générer un stub, nous spécifions cette colonne dans `gt()` en utilisant l'argument `rowname_col`:

```
hiv_malawi_summary %>%
  gt(rowname_col = "period") %>%
  tab_header(
    title = "Dépistage du VIH au Malawi",
    subtitle = "T1 à T2 2019"
  ) %>%
  tab_source_note("Source: Programme VIH du Malawi")
```

Dépistage du VIH au Malawi

T1 à T2 2019

	new_positive	previous_positive	new_negative	previous_negative	
2019 Q1	6199	14816	284694	6595	
2019 Q2	6132	15101	282249	5605	
2019 Q3	5907	15799	300529	6491	
2019 Q4	5646	15700	291622	6293	

Source: Programme VIH du Malawi

Notez que le nom de la colonne passé à `rowname_col` doit être entre guillemets.

Pour plus de commodité, sauvegardons le tableau dans une variable `t1`:

```
t1 <- hiv_malawi_summary %>%
  gt(rownames_col = "period") %>%
  tab_header(
    title = "Dépistage du VIH au Malawi",
    subtitle = "T1 à T2 2019"
  ) %>%
  tab_source_note("Source: Programme VIH du Malawi")

t1
```

Dépistage du VIH au Malawi

T1 à T2 2019

	new_positive	previous_positive	new_negative	previous_negative
2019 Q1	6199	14816	284694	6595
2019 Q2	6132	15101	282249	5605
2019 Q3	5907	15799	300529	6491
2019 Q4	5646	15700	291622	6293

Source: Programme VIH du Malawi

Colonnes Spinner & sous-colonnes

Pour mieux structurer notre tableau, nous pouvons regrouper des colonnes liées sous des “spanners”. Les spanners sont des titres qui couvrent plusieurs colonnes, fournissant une organisation catégorielle de niveau supérieur. Nous pouvons le faire avec la fonction `tab_spinner()`.

Créons deux colonnes spinner pour les tests nouveaux et précédents. Nous commencerons par le spinner “Nouveaux tests” afin que vous puissiez observer la syntaxe :

```
t1 %>%
  tab_spinner(
    label = "Nouveaux tests",
    columns = starts_with("new") # sélectionne les colonnes commençant
par "new"
  )
```

Dépistage du VIH au Malawi

T1 à T2 2019

	Nouveaux tests			
	new_positive	new_negative	previous_positive	previous_negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293

Source: Programme VIH du Malawi

L'argument `columns` nous permet de sélectionner les colonnes pertinentes, et l'argument `label` prend en entrée l'étiquette du spanner.

Ajoutons maintenant les deux spanners :

```
# Enregistre le tableau dans t2 pour un accès facile
t2 <- t1 %>%
  # Premier spanner pour "Nouveaux tests"
  tab_spinner(
    label = "Nouveaux tests",
    columns = starts_with("new"))
  ) %>%
  # Second spanner pour "Tests précédents"
  tab_spinner(
    label = "Tests précédents",
    columns = starts_with("prev"))
  )

t2
```

Dépistage du VIH au Malawi

T1 à T2 2019

	Nouveaux tests		Tests précédents	
	new_positive	new_negative	previous_positive	previous_negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293

Source: Programme VIH du Malawi

Notez que la fonction `tab_spanner` a automatiquement réarrangé les colonnes de manière appropriée.

Question 1 : Le but des Spanners

Quel est le but de l'utilisation de "colonnes spanner" dans un tableau `gt` ?

- A. Appliquer des styles CSS personnalisés à des colonnes spécifiques.
- B. Créer des colonnes de groupe et augmenter la lisibilité.
- C. Formater la taille de la police de toutes les colonnes de manière uniforme.
- D. Trier les données par ordre croissant.



Question 2 : Crédit de Spanners

En utilisant le cadre de données `hiv_malawi`, créez un tableau `gt` qui affiche un résumé de la `somme` des cas "nouveaux_positifs" et "précédents_positifs" pour chaque région. Créez des en-têtes spanner pour étiqueter ces deux colonnes de résumé. Pour y parvenir, remplissez les parties manquantes du code ci-dessous :



```
region_summary <- hiv_malawi %>%
  group_by(region) %>%
  summarise(
    _____(
      c(nouveaux_positifs, précédents_positifs),
      _____
    )
  )

# Créer un tableau gt avec des en-têtes spanner
summary_table_spanners <- region_summary %>%
  _____ %>%
  _____(
    label = "Cas positifs",
    _____ = c(nouveaux_positifs,
    précédents_positifs)
  )
```

Renommage des colonnes du tableau

Les noms des colonnes contiennent actuellement des préfixes inutiles comme “new_” et “previous_”. Pour une meilleure lisibilité, nous pouvons les renommer en utilisant `cols_label()`.

`cols_label()` prend un ensemble d’anciens noms à apparter (du côté gauche d’une tilde, ~) et de nouveaux noms pour les remplacer (du côté droit de la tilde). Nous pouvons utiliser `contains()` pour sélectionner des colonnes avec “positive” ou “negative” :

```
t3 <- t2 %>%
  cols_label(
    contains("positive") ~ "Positive",
    contains("negative") ~ "Negative"
  )

t3
```

Dépistage du VIH au Malawi

T1 à T2 2019

	Nouveaux tests		Tests précédents	
	Positive	Negative	Positive	Negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293

Source: Programme VIH du Malawi

Ceci renomme les colonnes d'une manière plus propre.

`cols_label()` accepte plusieurs aides à la sélection de colonnes comme `contains()`, `starts_with()`, `ends_with()` etc. Celles-ci proviennent du package `tidyselect` et offrent une flexibilité dans le renommage.

`cols_label()` a d'autres fonctions d'identification comme `contains()` qui fonctionnent de manière similaire et sont identiques aux aides de `tidyselect`, celles-ci incluent également :

PRO TIP



- `starts_with()`: Commence par un préfixe exact.
- `ends_with()`: Se termine par un suffixe exact.
- `contains()`: Contient une chaîne de caractères littérale.
- `matches()`: Correspond à une expression régulière.
- `num_range()`: Correspond à une plage numérique comme x01, x02, x03.

Ces aides sont utiles, en particulier dans le cas de la sélection de plusieurs colonnes.

Pour en savoir plus sur la fonction `cols_label()`, vous pouvez consulter ici : https://rstudio.com/reference/cols_label.html

Question 3 : étiquettes de colonnes

Quelle fonction est utilisée pour changer les étiquettes ou les noms des colonnes dans un tableau **gt** ?

PRACTICE



- A. `tab_header()`
- B. `tab_style()`
- C. `tab_options()`
- D. `tab_relabel()`

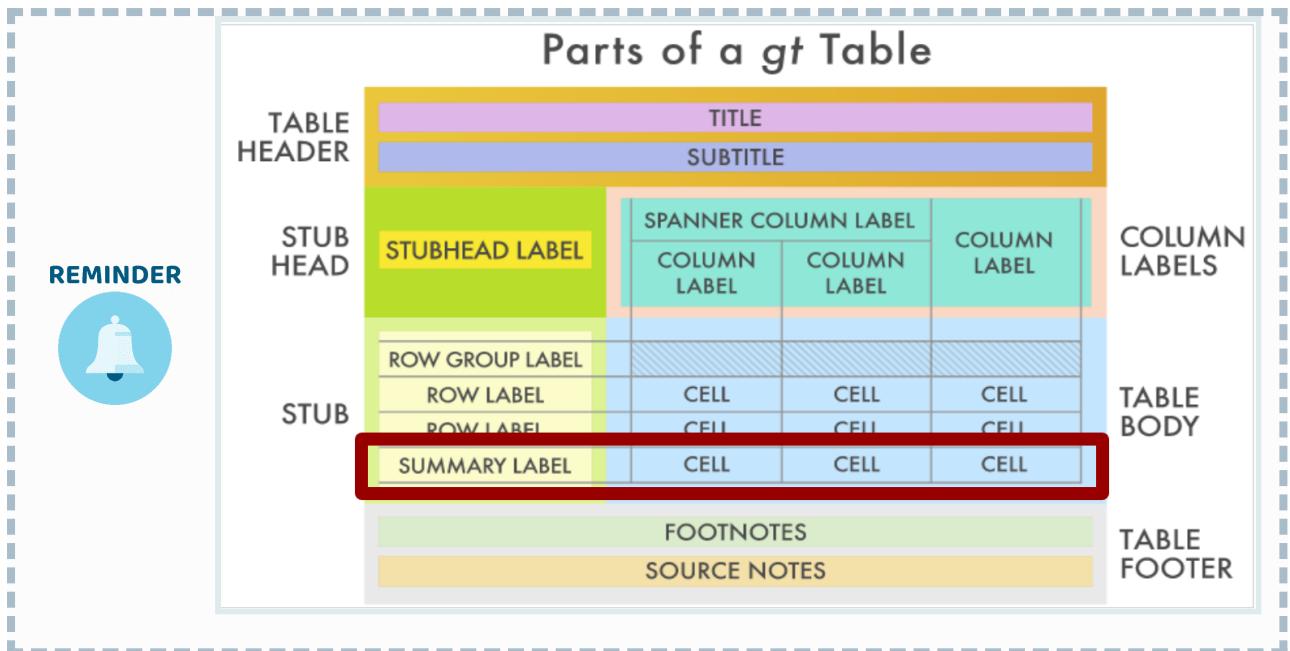
Lignes de résumé

Prenons les mêmes données avec lesquelles nous avons commencé au début de cette leçon et, au lieu de regrouper uniquement par période (trimestres), regroupons à la fois par période et par région. Nous faisons cela pour illustrer la puissance des fonctionnalités de résumé dans **gt** : les tableaux récapitulatifs.

REMINDER



Rappel {gt} - Lignes de résumé Cette image montre le composant des lignes de résumé d'un tableau **{gt}**, clairement indiqué dans un carré rouge. Les lignes de résumé fournissent des données agrégées ou des résumés statistiques des données contenues dans les colonnes correspondantes.



Tout d'abord, recréons les données :

```
summary_data_2 <- hiv_malawi %>%
  group_by(
    # Notez l'ordre des variables par lesquelles nous regroupons.
    region,
    period
  ) %>%
  summarise(
    across(all_of(cols), sum)
  ) %>%
  gt()
```

`summarise()` has grouped output by 'region'. You can override using the `groups` argument.

```
summary_data_2
```

period	new_positive	previous_positive	new_negative	previous_negative
Central Region				
2019 Q1	2004	3682	123018	2562
2019 Q2	1913	3603	116443	1839
2019 Q3	1916	4002	127799	2645
2019 Q4	1691	3754	124728	1052
Northern Region				
2019 Q1	664	1197	36196	675
2019 Q2	582	1084	35315	590
2019 Q3	570	1191	36850	542
2019 Q4	519	1132	34322	346
Southern Region				
2019 Q1	3531	9937	125480	3358
2019 Q2	3637	10414	130491	3176
2019 Q3	3421	10606	135880	3304
2019 Q4	3436	10814	132572	4895

WATCH OUT



L'ordre dans la fonction `group_by()` affecte les groupes de lignes dans le tableau `gt`.

Deuxièmement, réincorporons tous les changements que nous avons précédemment effectués dans ce tableau :

```

# sauvegardant les progrès dans l'objet t4

t4 <- summary_data_2 %>%
  tab_header(
    title = "Somme des tests VIH au Malawi",
    subtitle = "du T1 2019 au T4 2019"
  ) %>%
  tab_source_note("Source des données : Programme VIH du Malawi") %>%
tab_spinner(
  label = "Nouveaux tests",
  columns = starts_with("new") # sélectionne les colonnes commençant
par "new"
) %>%
  # création du premier spinner pour les tests précédents
  tab_spinner(
    label = "Tests précédents",
    columns = starts_with("prev") # sélectionne les colonnes
commençant par "prev"
) %>%
  cols_label(
    # localiser ### assigner
    contains("positive") ~ "Positif",
    contains("negative") ~ "Négatif"
  )

t4

```

Somme des tests VIH au Malawi

du T1 2019 au T4 2019

period	Nouveaux tests		Tests précédents	
	Positif	Négatif	Positif	Négatif
Central Region				
2019 Q1	2004	123018	3682	2562
2019 Q2	1913	116443	3603	1839
2019 Q3	1916	127799	4002	2645
2019 Q4	1691	124728	3754	1052
Northern Region				
2019 Q1	664	36196	1197	675
2019 Q2	582	35315	1084	590
2019 Q3	570	36850	1191	542
2019 Q4	519	34322	1132	346
Southern Region				
2019 Q1	3531	125480	9937	3358
2019 Q2	3637	130491	10414	3176
2019 Q3	3421	135880	10606	3304
2019 Q4	3436	132572	10814	4895
Source des données : Programme VIH du Malawi				

Maintenant, que faire si nous voulons visualiser sur le tableau un résumé de chaque variable pour chaque groupe de régions ? Plus précisément, nous voulons voir la somme et la moyenne pour les 4 colonnes que nous avons pour chaque région.

N'oubliez pas que nos 4 colonnes d'intérêt sont : "new_positive", "previous_positive", "new_negative", et "previous_negative". Nous n'avons changé les labels de ces colonnes que dans la table gt et non dans le jeu de données lui-même, nous pouvons donc utiliser les noms de ces colonnes pour indiquer à gt où appliquer la fonction de résumé. De plus, nous avons déjà stocké les noms de ces 4 colonnes dans l'objet cols donc nous allons l'utiliser à nouveau ici.

Pour atteindre cet objectif, nous allons utiliser la fonction pratique `summary_rows` où nous fournissons explicitement les colonnes que nous voulons résumer, et les fonctions avec lesquelles nous voulons résumer, dans notre cas c'est `sum` et `mean`. Notez que nous assignons le nom de la nouvelle ligne (non citée) à un nom de fonction ("cité").

```
t5 <- t4 %>%
  summary_rows(
    columns = cols, #using columns = 3:6 also works
    fns = list(
      TOTAL = "sum",
      MOYENNE = "mean"
    )
  )

t5
```

Somme des tests VIH au Malawi

du T1 2019 au T4 2019

period	Nouveaux tests		Tests précédents	
	Positif	Négatif	Positif	Négatif
Central Region				
2019 Q1	2004	123018	3682	2562
2019 Q2	1913	116443	3603	1839
2019 Q3	1916	127799	4002	2645
2019 Q4	1691	124728	3754	1052
sum	—	7524.00	491988.00	15041.00
mean	—	1881.00	122997.00	3760.25
Northern Region				
2019 Q1	664	36196	1197	675
2019 Q2	582	35315	1084	590
2019 Q3	570	36850	1191	542
2019 Q4	519	34322	1132	346
sum	—	2335.00	142683.00	4604.00
mean	—	583.75	35670.75	1151.00
Southern Region				
2019 Q1	3531	125480	9937	3358
2019 Q2	3637	130491	10414	3176
2019 Q3	3421	135880	10606	3304
2019 Q4	3436	132572	10814	4895
sum	—	14025.00	524423.00	41771.00
mean	—	3506.25	131105.75	10442.75

Source des données : Programme VIH du Malawi

Question 4 : lignes résumées

Quelle est la bonne réponse (ou les bonnes réponses) si vous deviez résumer l'écart type des lignes des colonnes "new_positive" et "previous_negative" uniquement?

- A. Utilisez **summary_rows()** avec l'argument **columns** défini sur "new_positive" et "previous_negative" et l'argument **fns** défini sur "sd".

```
# Option A
your_data %>%
  summary_rows(
    columns = c("new_positive", "previous_negative"),
    fns = "sd"
  )
```

- B. Utilisez **summary_rows()** avec l'argument **columns** défini sur "new_positive" et "previous_negative" et l'argument **fns** défini sur "summarize(sd)".

```
# Option B
your_data %>%
  summary_rows(
    columns = c("new_positive", "previous_negative"),
    fns = summarize(sd)
  )
```

- C. Utilisez **summary_rows()** avec l'argument **columns** défini sur "new_positive" et "previous_negative" et l'argument **fns** défini sur **list(ECART_TYPE = "sd")**.

```
# Option C
your_data %>%
  summary_rows(
    columns = c("new_positive", "previous_negative"),
    fns = list(ECART_TYPE = "sd")
  )
```

- D. Utilisez **summary_rows()** avec l'argument **columns** défini sur "new_positive" et "previous_negative" et l'argument **fns** défini sur "standard_deviation".

```
# Option D
your_data %>%
  summary_rows(
    columns = c("new_positive", "previous_negative"),
    fns = "standard_deviation"
  )
```

Conclusion

Dans la leçon d'aujourd'hui, nous nous sommes attaqués aux tables de données dans R en utilisant gt. Nous avons commencé par définir des objectifs clairs, présenté les packages que nous utiliserons et découvert notre jeu de données. Ensuite, nous avons mis la main à la pâte en créant des tables simples. Nous avons appris à organiser nos données proprement en utilisant des colonnes spanner et en ajustant les étiquettes des colonnes pour rendre les choses parfaitement claires et cohérentes. Nous avons ensuite conclu avec quelques résumés de table astucieux. Ce sont les bases de la création de tables dans R et gt, et elles seront très utiles à mesure que nous poursuivrons notre voyage pour créer des tableaux engageants et informatifs dans R.

Corrigé

1. B

2.

```
# Les solutions sont où les lignes sont numérotées

# résumer les données d'abord
district_summary <- hiv_malawi %>%
  group_by(region) %>%
  summarize(
    across( #1
      c(new_positive, previous_positive),
      sum #2
    )
  )

# Créer une table gt avec des en-têtes spanner
summary_table_spanners <- district_summary %>%
  gt() %>% #3
  tab_spanner( #4
    label = "Cas positifs",
    columns = c(new_positive, previous_positive) #5
  )
```

3. D

4. C

Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



BENNOUR HSIN

Data Science Education Officer
Data Visualization enthusiast



JOY VAZ

R Developer and Instructor, the GRAPH Network
Loves doing science and teaching science



GUY WAFEU

R Instructor and Public Health Physician
Committed to improving the quality of data analysis

Références

1. Tom Mock, "The Definite Cookbook of {gt}" (2021), The Mockup Blog, <https://themockup.blog/static/resources/gt-cookbook.html#introduction>.
2. Tom Mock, "The Grammar of Tables" (May 16, 2020), The Mockup Blog, <https://themockup.blog/posts/2020-05-16-gt-a-grammar-of-tables/#add-titles>.
3. RStudio, "Introduction to Creating gt Tables," Official {gt} Documentation, <https://gt.rstudio.com/articles/intro-creating-gt-tables.html>.
4. Fleming, Jessica A., Alister Munthali, Bagrey Ngwira, John Kadzandira, Monica Jamili-Phiri, Justin R. Ortiz, Philipp Lambach, et al. 2019. "Maternal Immunization in Malawi: A Mixed Methods Study of Community Perceptions, Programmatic Considerations, and Recommendations for Future Planning." *Vaccine* 37 (32): 4568-75. <https://doi.org/10.1016/j.vaccine.2019.06.020>.

This work is licensed under the Creative Commons Attribution Share Alike license.



Introduction
Objectifs d'apprentissage
Packages
Introduction aux étiquettes avec {ggplot2}
Données : Résultats du traitement de la tuberculose au Bénin
Étiquetage de diagrammes à barres simples
Étape 1 : Résumer les données
Étape 2 : Créer le graphique de base
Étape 3 : Annoter le graphique avec <code>geom_text()</code> ou <code>geom_label()</code>
Étape 4 : Ajuster le positionnement et le style du texte
Étiquetage des graphiques à barres empilées
Étape 1 : Résumer les données
Étape 2 : Créer le graphique de base
Étape 3 : Annoter le graphique avec <code>geom_text()</code> ou <code>geom_label()</code>
Étape 4 : Ajuster la position du texte pour l'aligner avec les barres
Étiquetage des graphiques à barres groupées
Étape 1 : Résumer les données
Étape 2 : Créer le graphique de base
Étape 3 : Annoter le graphique avec <code>geom_text()</code>
Étape 4 : Ajuster la position du texte pour l'aligner avec les barres
Étiquetage des graphiques à barres empilées normalisées
Étape 1 : Résumer les données
Étape 2 : Créer le graphique de base
Étape 3 : Annoter le graphique avec <code>geom_text()</code> ou <code>geom_label()</code>
Étape 4 : Ajuster la position du texte pour l'aligner avec les barres
Étiquetage de graphiques circulaires
Étape 1 : Résumer les données
Étape 2 : Créer les graphiques de base
Étape 3 : Annoter le graphique avec <code>geom_text()</code> ou <code>geom_label()</code>
Étape 4 : Ajuster la position du texte pour l'aligner avec les tranches du cercle et sections de l'anneau
Conclusion !

Introduction

Les graphiques à barres sont l'un des types de graphiques les plus courants et existent sous plusieurs variantes. Dans la leçon précédente, nous avons appris à créer des graphiques à barres et leurs équivalents circulaires avec `{ggplot2}`.



Cette leçon pivotera des comparaisons de groupes à la pratique de l'étiquetage dans la visualisation de données. Les étiquettes fournissent un contexte supplémentaire, clarifient les points de données et améliorent la lisibilité générale d'un graphique. Nous nous plongerons dans les détails de l'étiquetage dans ggplot2, en nous concentrant particulièrement sur les fonctions `geom_label()` et `geom_text()` de `{ggplot2}`.

Objectifs d'apprentissage

Après cette leçon, vous serez en mesure de :

1. Utiliser deux différentes façons pour étiqueter les graphiques ggplots :
 - `geom_text()` pour les étiquettes simples
 - `geom_label()` pour les étiquettes accentuées
2. Transformer et résumer les données de manière appropriée dans le format approprié pour les différents types de graphiques.
3. Ajuster le positionnement du texte pour positionner les étiquettes sur des graphiques à barres empilées, groupées et en pourcentage empilé.

-
4. Ajuster le positionnement du texte pour positionner les étiquettes sur les diagrammes circulaires et les diagrammes circulaires à anneau.
-

Packages

Nous utiliserons une combinaison de packages dans cette leçon pour améliorer nos visualisations de données :

1. `tidyverse` : Une collection de paquets R pour une manipulation et une visualisation de données efficaces, incluant `ggplot2`.
2. `glue` : Permet une interpolation de chaînes flexible pour du texte dynamique dans les graphiques.
3. `here` : Pour les chemins de fichiers relatifs au projet.

```
pacman::p_load(tidyverse, glue, here)
```

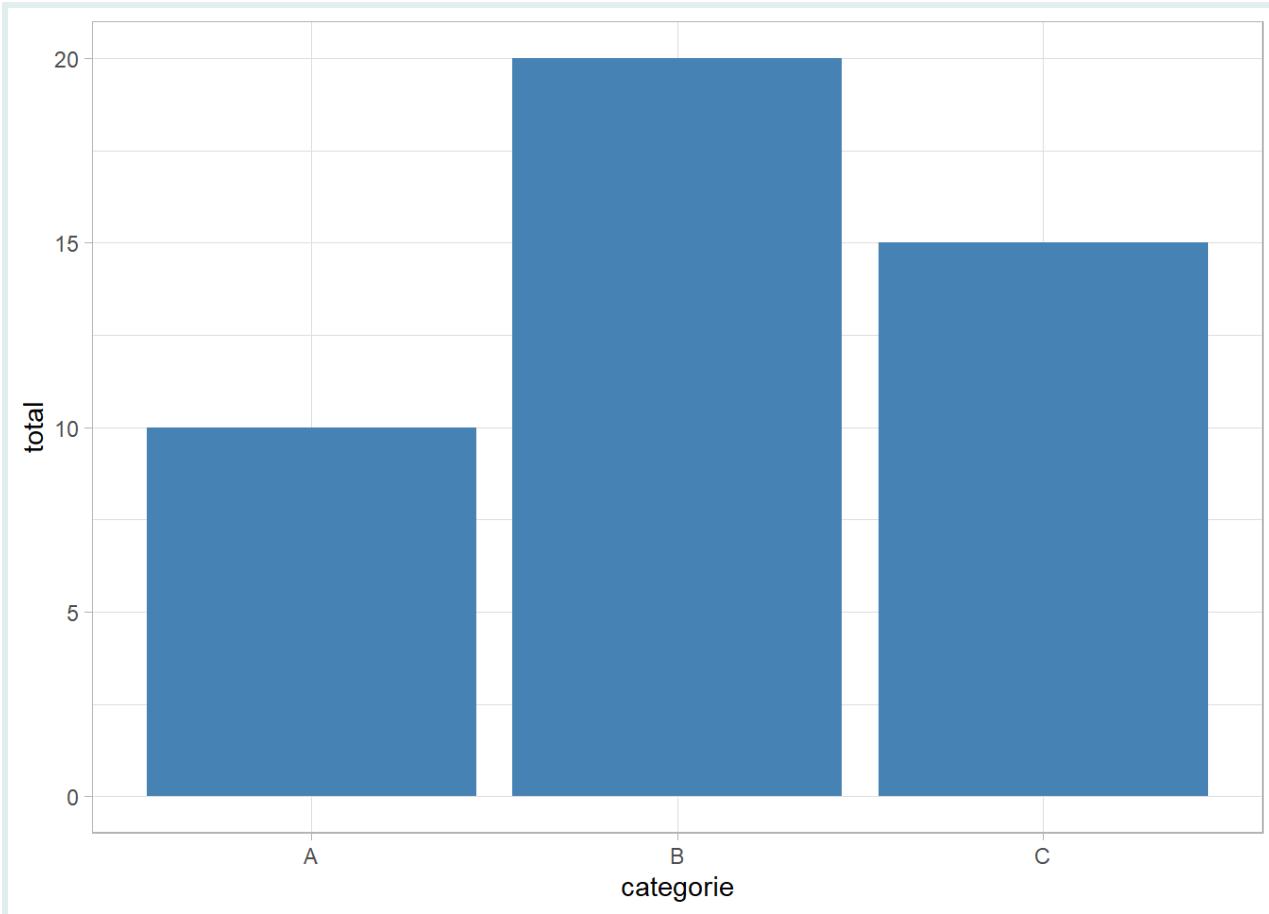
Introduction aux étiquettes avec {ggplot2}

Dans `{ggplot2}`, ajouter des étiquettes est un exercice de précision et d'esthétique. Nous commencerons par `geom_text()` pour un étiquetage simple, ensuite on passerons à `geom_label()` pour des étiquettes plus stylisé. Dans cette section, nous présenterons la différence entre ces deux fonctions avec un simple diagramme à barres, puis nous entrerons plus en détail sur comment les utiliser sur les barres empilées, les barres groupées, les barres empilées normalisées et les graphiques circulaires.

Commençons par pratiquer l'utilisation de ces fonctions sur un simple diagramme à barres fait avec des données fictives. Une fois que nous aurons couvert les bases de la syntaxe d'étiquetage, nous l'appliquerons à de vraies données épidémiologiques.

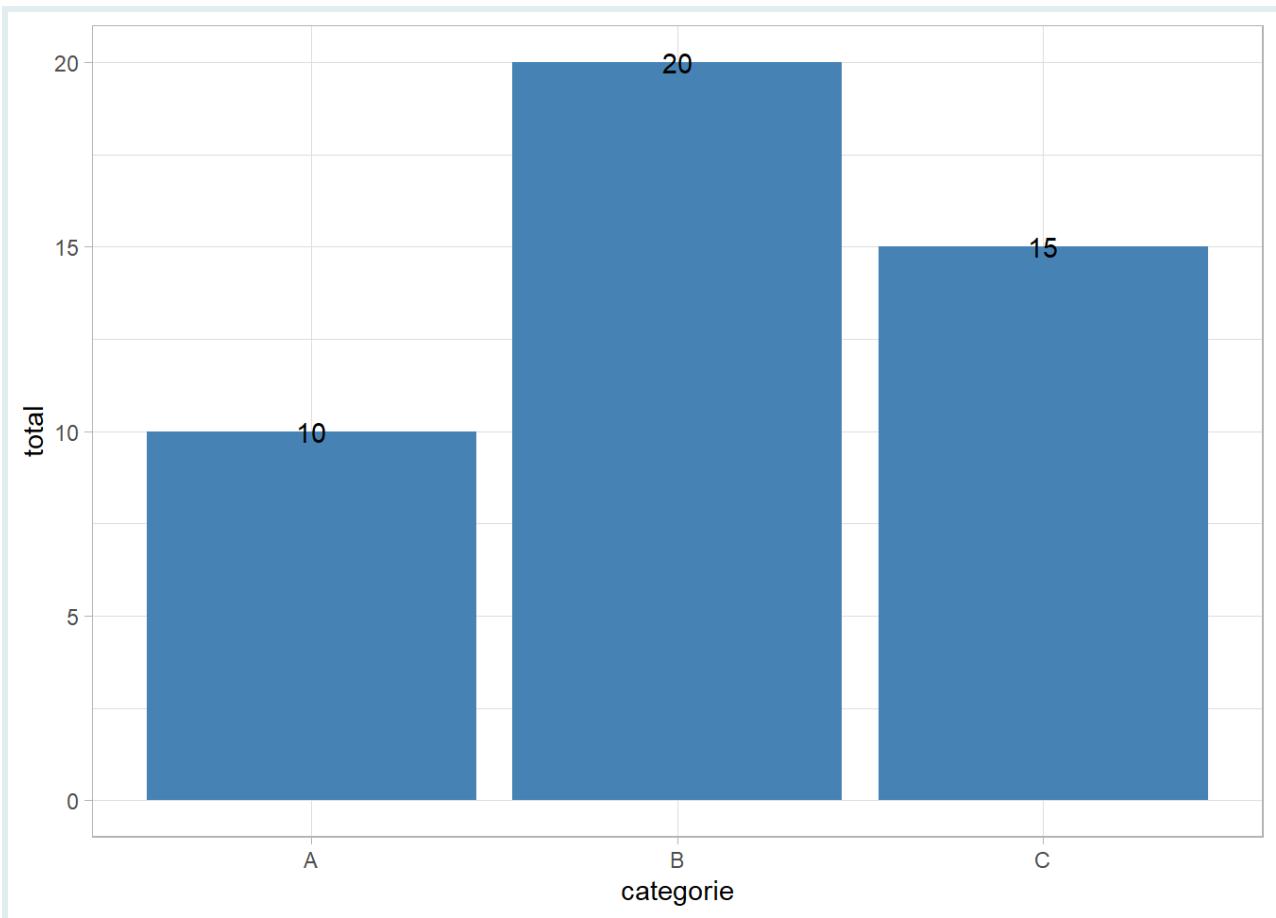
```
# Créer un exemple de dataframe
data <- data.frame(
  categorie = c("A", "B", "C"),
  total = c(10, 20, 15)
)

# Créer le diagramme à barres
ggplot(data, aes(x = categorie, y = total)) +
  geom_col(fill = "steelblue") +
  theme_light()
```



On peut facilement ajouter des étiquettes à notre diagramme. Pour cela, nous utilisons la fonction `geom_text()` et on va préciser à `aes()` quelle variable qu'il faut extraire pour l'étiquette en utilisant `label =:`

```
# Ajouter des étiquettes de texte sur les barres
ggplot(data, aes(x = categorie, y = total)) +
  geom_col(fill = "steelblue") +
  theme_light() +
  # AJOUTEZ JUSTE UNE COUCHE GEOM !
  geom_text(aes(label = total)) # doit fournir une variable à l'argument
    `label`
```

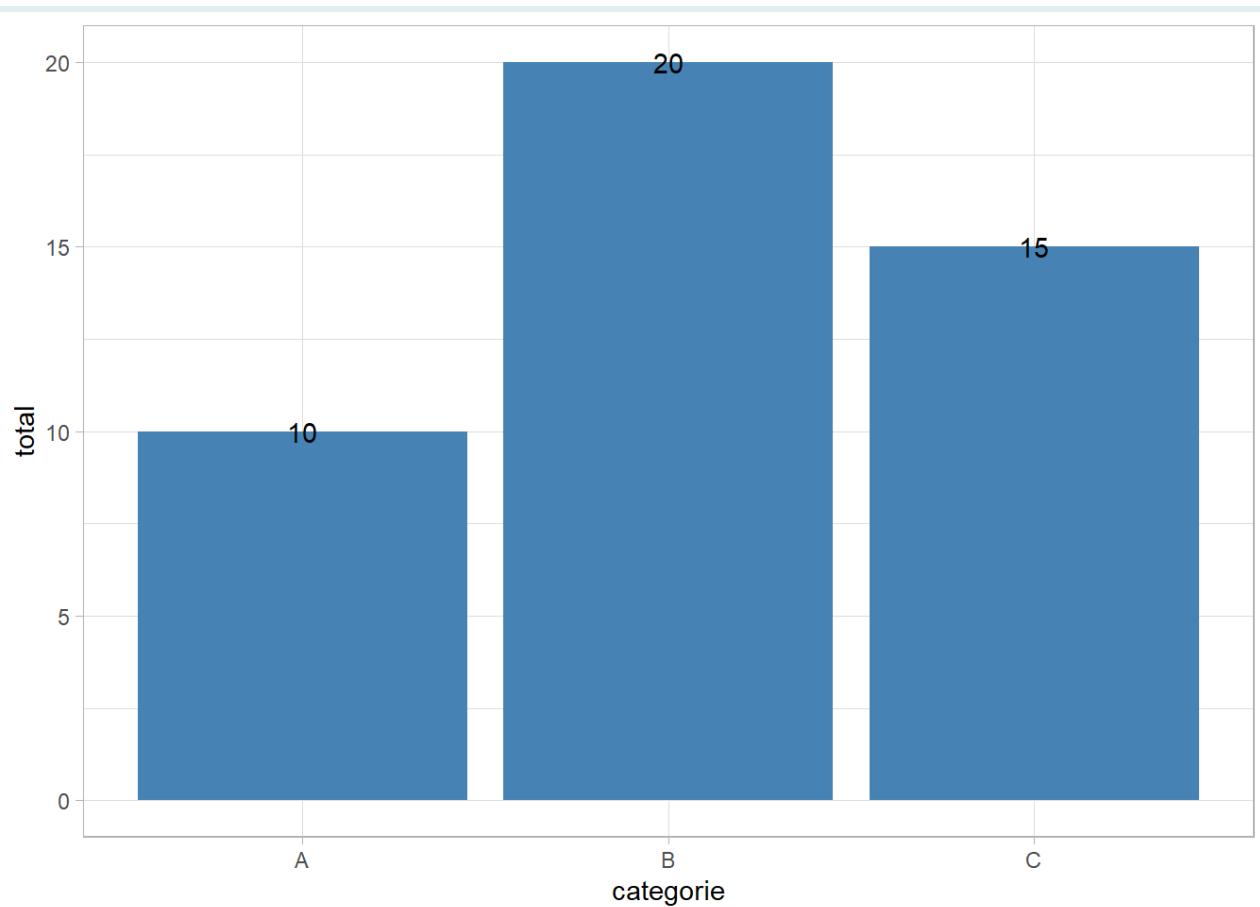


Comme vous pouvez le voir, c'est une façon très simple d'améliorer votre diagramme `ggplot` en quelques lignes de codes. Le reste de cette leçon vous apprendra comment le faire de plusieurs différentes manières.

Ensuite, nous modifierons le code simple ci-dessus pour apprendre `geom_text()` et `geom_label()`. Ces deux façons d'étiquetter offrent des approches distinctes pour ajouter du texte aux graphiques, chacune avec ses caractéristiques et cas d'utilisation uniques.

- `geom_text()` : cette fonction place du texte brut directement sur le graphique. Il est préférable de l'utiliser lorsque l'arrière-plan n'est pas trop chargé et que le texte n'a pas besoin de se démarquer excessivement.

```
# Etiquettes de base avec geom_text()
ggplot(data, aes(x = categorie, y = total)) +
  geom_col(fill = "steelblue") +
  theme_light() +
  geom_text(aes(label = total))
```



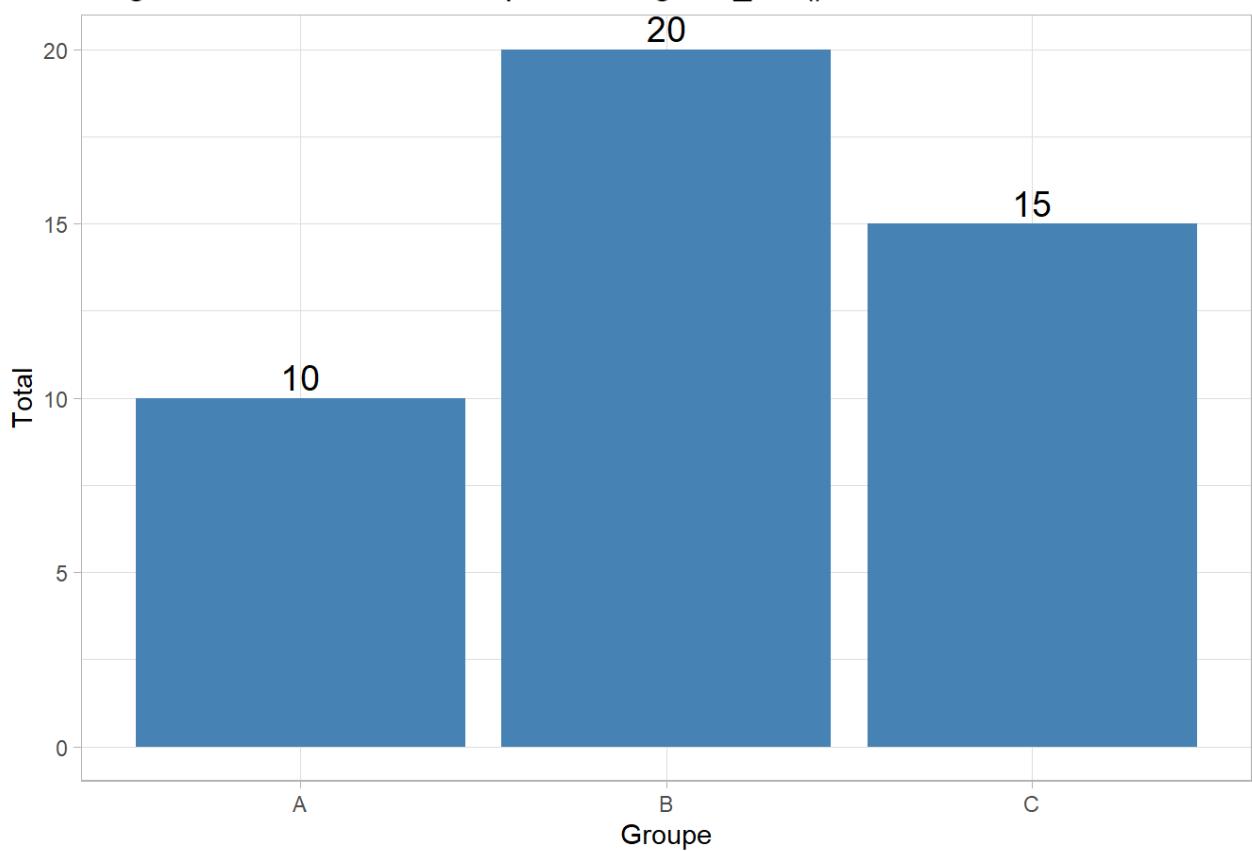
Bien que les étiquettes soient effectivement utiles, l'emplacement de notre texte est plutôt particulier, ni sur la barre, ni sous la barre. De plus, elles sont quelque peu petites et difficiles à discerner. Nous pouvons remédier à cela en agrandissant leur taille et en ajustant verticalement leur emplacement.

La fonction `geom_text()` a comme arguments `hjust` et `vjust` qui vous aide à aligner et positioner vos étiquettes.

Nous pouvons nous appuyer sur le code de l'exemple précédent pour ajouter des esthétiques supplémentaires à la fonction :

```
# Étiquettes avec geom_text()
ggplot(data, aes(x = categorie, y = total)) +
  geom_col(fill = "steelblue") +
  # PERSONNALISER LES ESTHÉTIQUES
  geom_text(aes(label = total),
            vjust = -0.3, # Déplacer le texte vers le haut
            size = 5) + # Augmenter la taille du texte
  theme_light() +
  labs(title = "Diagramme à barres avec étiquettes de geom_text()", 
       x = "Groupe", y = "Total")
```

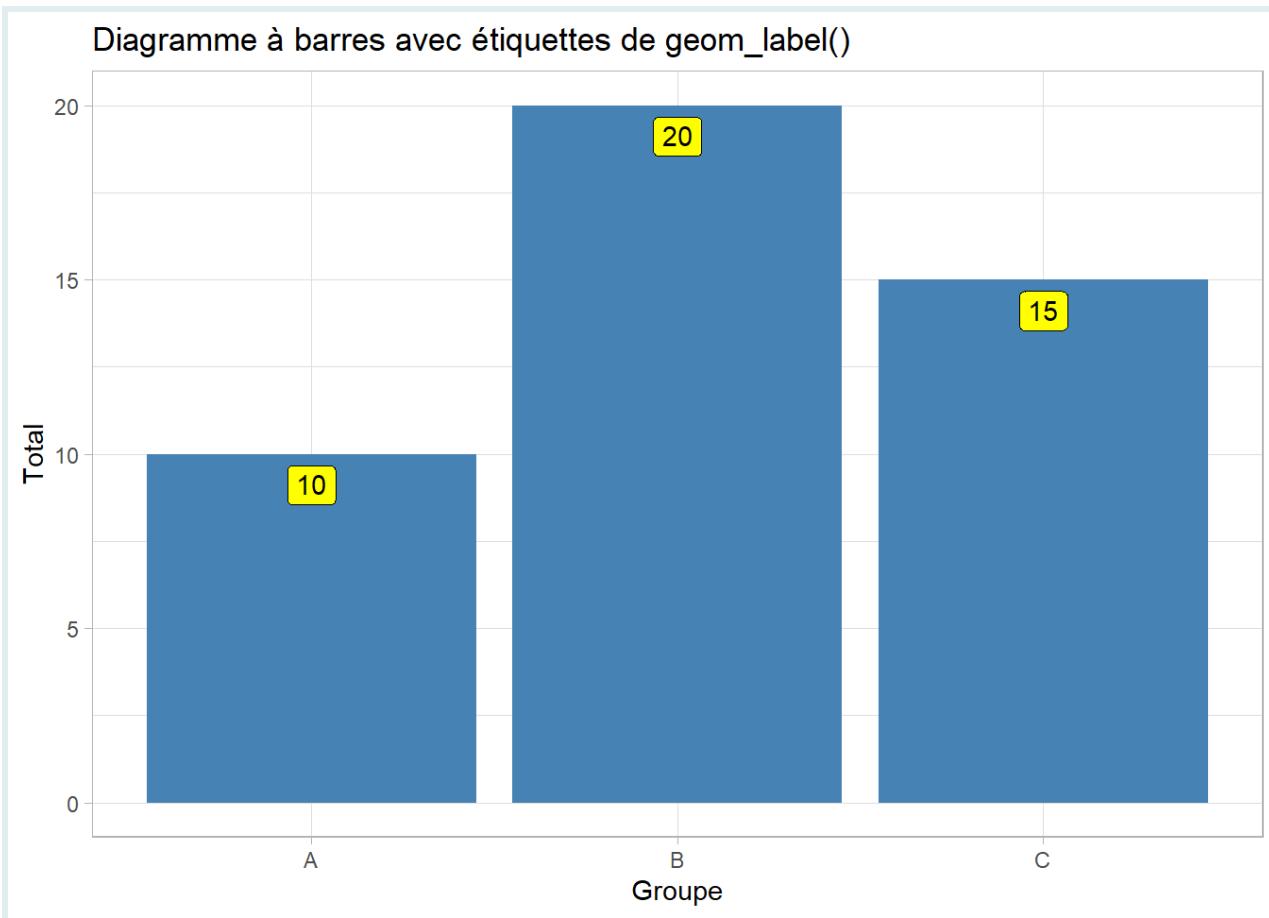
Diagramme à barres avec étiquettes de geom_text()



Dans ce code, le paramètre `vjust` est ajusté pour positionner le texte juste au-dessus des barres, et le paramètre `size` contrôle la taille de la police.

- `geom_label()` : cette fonction dessine un rectangle derrière le texte, améliorant le contraste et la lisibilité, en particulier utile dans les graphiques avec des arrière-plans complexes.

```
# Étiquettes avec geom_label()
ggplot(data, aes(x = categorie, y = total)) +
  geom_col(fill = "steelblue") +
  # GEOM_LABEL()
  geom_label(aes(label = total),
             vjust = 1.3, # Déplacer le texte vers le bas
             fill = "yellow") + # Couleur d'arrière-plan du rectangle
  theme_light() +
  labs(title = "Diagramme à barres avec étiquettes de geom_label()",  
       x = "Groupe", y = "Total")
```



Dans ce code, l'esthétique `fill` dans `geom_label()` peut être ajustée pour contrôler la couleur de remplissage d'arrière-plan des étiquettes, et cette fois le paramètre `vjust` est ajusté pour abaisser les étiquettes.



Deux façons d'étiquetter distinctes

- Les deux façons d'étiquetter peuvent être utilisées en combinaison avec d'autres fonction de `ggplot`, comme `geom_col()`, pour annoter la hauteur des barres
- `geom_text()` ajoute uniquement du texte au graphique
- `geom_label()` dessine un rectangle derrière le texte, le rendant plus facile à lire.





Définir un thème global pour {ggplot2} Jusqu'à présent, nous avons ajouté une fonction de thème à chacun de nos graphiques à barres. Nous pouvons utiliser la fonction `theme_set()` pour définir un thème global pour le reste de nos graphiques, de sorte que nous n'ayons pas à l'ajouter à chaque fois.

```
# Définir un thème light pour tous les ggplots de cette leçon  
theme_set(theme_light())
```

Désormais, `theme_light()` sera automatiquement appliqué à chaque graphique que vous créez.

Données : Résultats du traitement de la tuberculose au Bénin

Les exemples ci-dessus semblent assez simples, mais les données réelles sont souvent plus complexes, surtout lorsque vous avez plusieurs sous-groupes et niveaux d'agrégation. Vous rencontrerez des problèmes si vous ne préparez pas correctement les données. Plongeons-y.

Le jeu de données `resultats_tb` servira de base pour nos exemples, nous fournissant un ensemble riche de points de données à étiqueter.

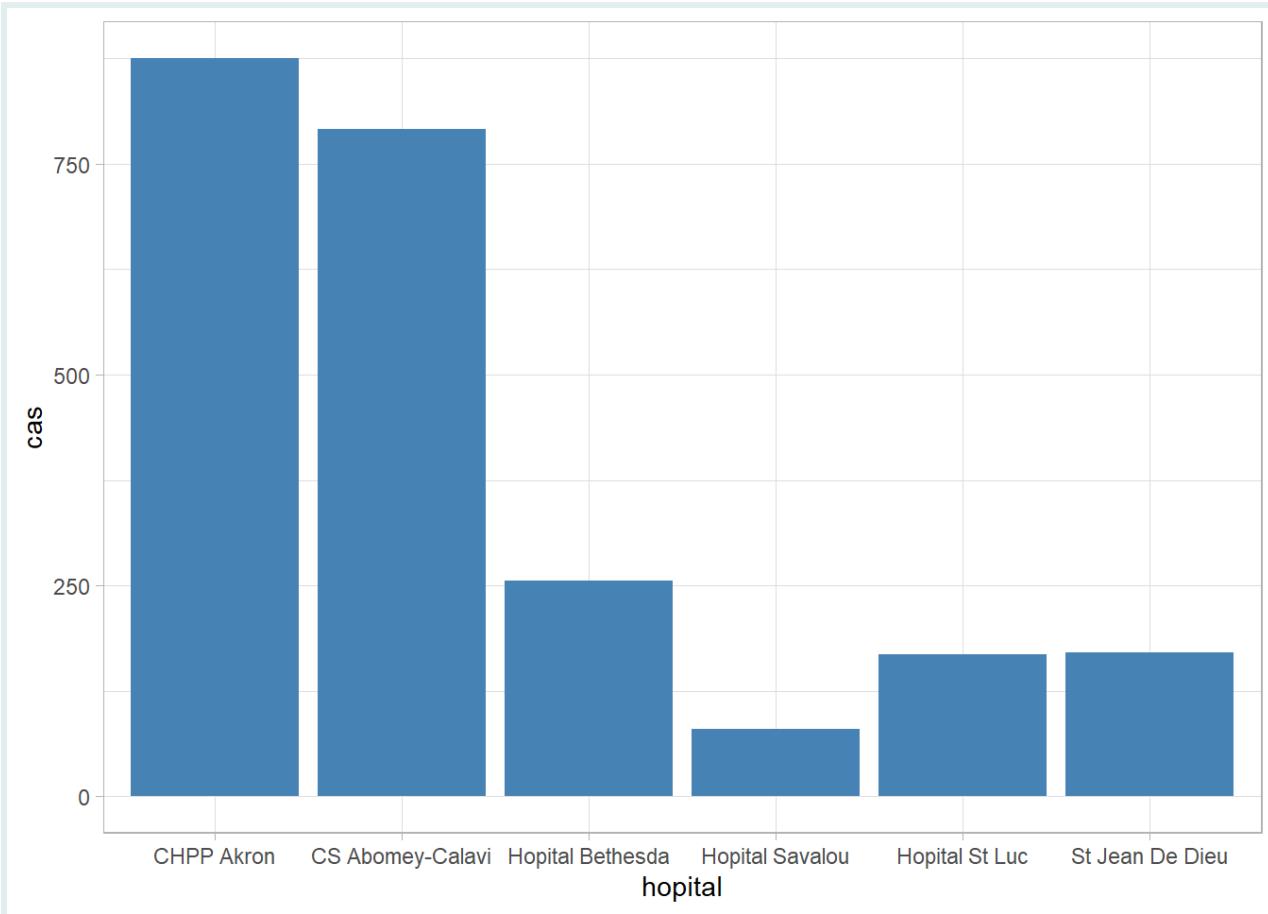
```
resultats_tb <- read_csv(here::here('data/benin_tb_fr.csv'))  
resultats_tb
```

Nous allons régénérer les graphiques de la leçon précédente pour les utiliser comme base pour cette leçon.

Étiquetage de diagrammes à barres simples

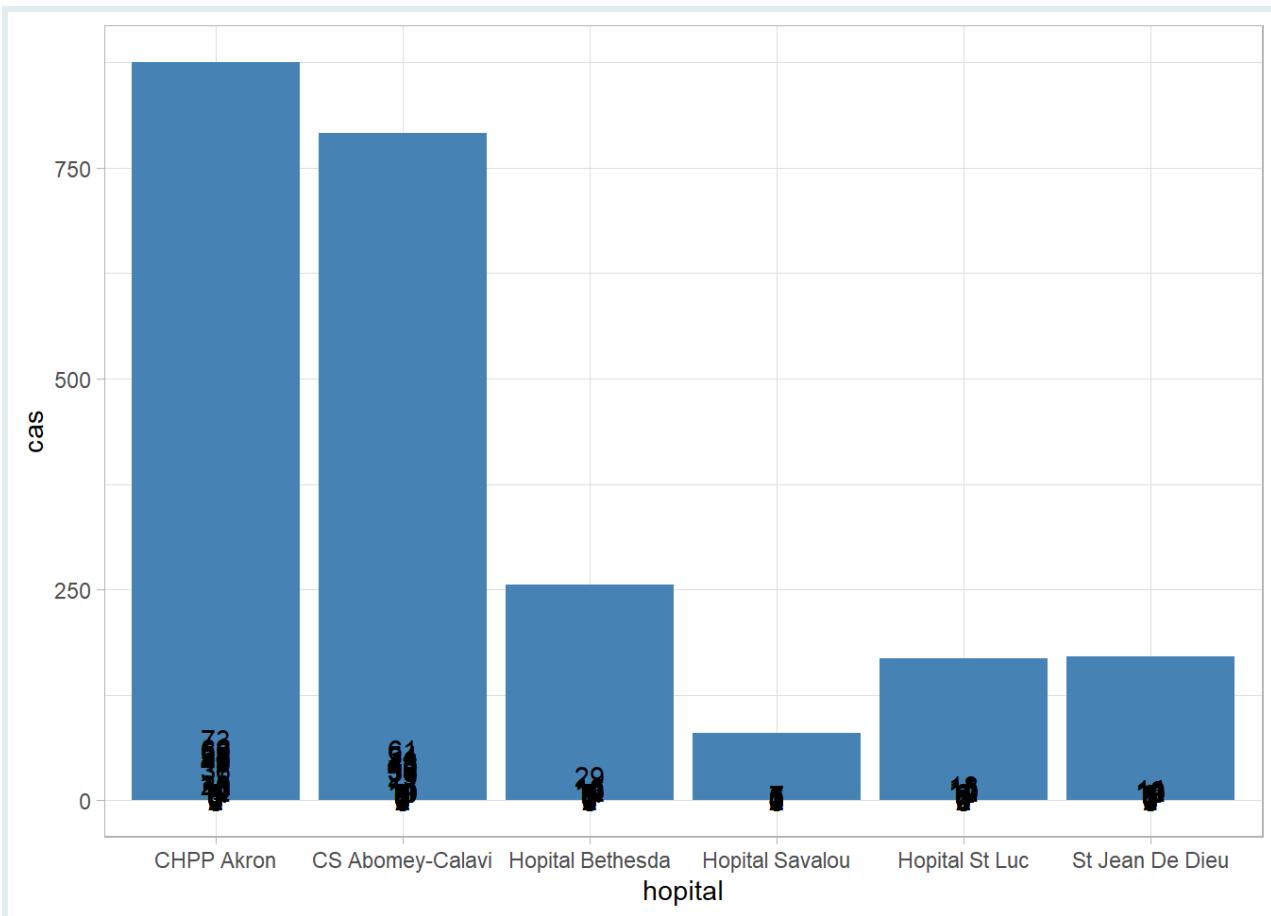
Commençons par créer un diagramme à barres simple pour visualiser le nombre de cas par pour chaque hôpital.

```
# Diagramme à barres normal - splendide! aucun précalcul requis  
resultats_tb %>%  
  ggplot(aes(x = hopital, y = cas)) +  
  geom_col(fill = "steelblue")
```

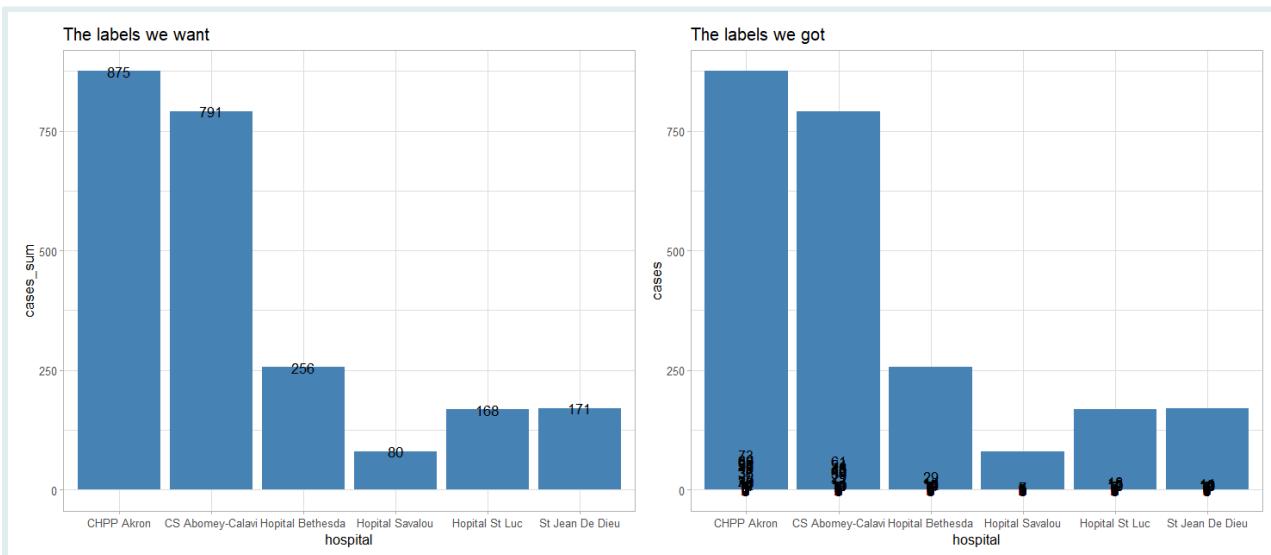


Super! Ajoutons `geom_text()` comme nous l'avons fait précédemment.

```
# mais si nous essayons d'ajouter geom_text, nous avons un problème...
resultats_tb %>%
  ggplot(aes(x = hopital, y = cas)) +
  geom_col(fill = "steelblue") +
  geom_text(aes(label = cas))
```



Oh non ! Plutôt qu'un seul nombre en haut de chaque barre, nous nous retrouvons avec beaucoup d'étiquettes entassées en bas de chaque barre. Examinons le problème.



Attentes vs. Réalité

- Regardez nos données `resultats_tb`, la colonne `cas`. La fonction `geom_text` prend des valeurs directement à partir de là et les place sur l'axe des y.

```
resultats_tb
```

Au lieu d'avoir un seul nombre en haut de chaque barre, on se retrouve avec une multitude d'étiquettes entassées en bas de chaque barre. Il y a 711 lignes, ce qui signifie que 711 étiquettes ont été dessinées par `geom_text()`.

Contrairement à notre ensemble de données fictif où on avait des nombres précalculés pour chaque barre, **avec les données brutes on a souvent des cas non agrégés**. Bien que la fonction de base `geom_col()` intègre le calcul des totaux de groupe, `geom_text()` et `geom_label()` ne le font pas automatiquement, ce qui veut dire que nous on doit créer manuelle des étiquettes.

Pour résoudre ce problème, on va créer un jeu de données temporaire à l'aide des fonctions de `{dplyr}`, ce qui nous permettra de résumer le nombre total de cas par hôpital.

Étape 1 : Résumer les données

Tout d'abord, nous allons commencer par agréger nos données puis créer un diagramme à barres de base comme point de départ

- **Commencez par résumer les données :** ce bloc de code régroupe d'abord notre ensemble de données `resultats_tb` par `hopital` et calcule la somme total des `cas` pour chaque groupe

```
hopital_cas_total <- resultats_tb %>%
  # Grouper les données par hôpital
  group_by(hopital) %>%
  # Obtenir le nombre total de cas par hôpital
  summarise(cas_total = sum(cas))

hopital_cas_total
```

Maintenant, nous avons 6 valeurs pour `cas_total`, qui serviront de 6 étiquettes de texte, une pour chaque barre.

Étape 2 : Créer le graphique de base

Utilisons maintenant `hopital_cas_total` pour visualiser le nombre total de cas pour chaque hôpital :

```

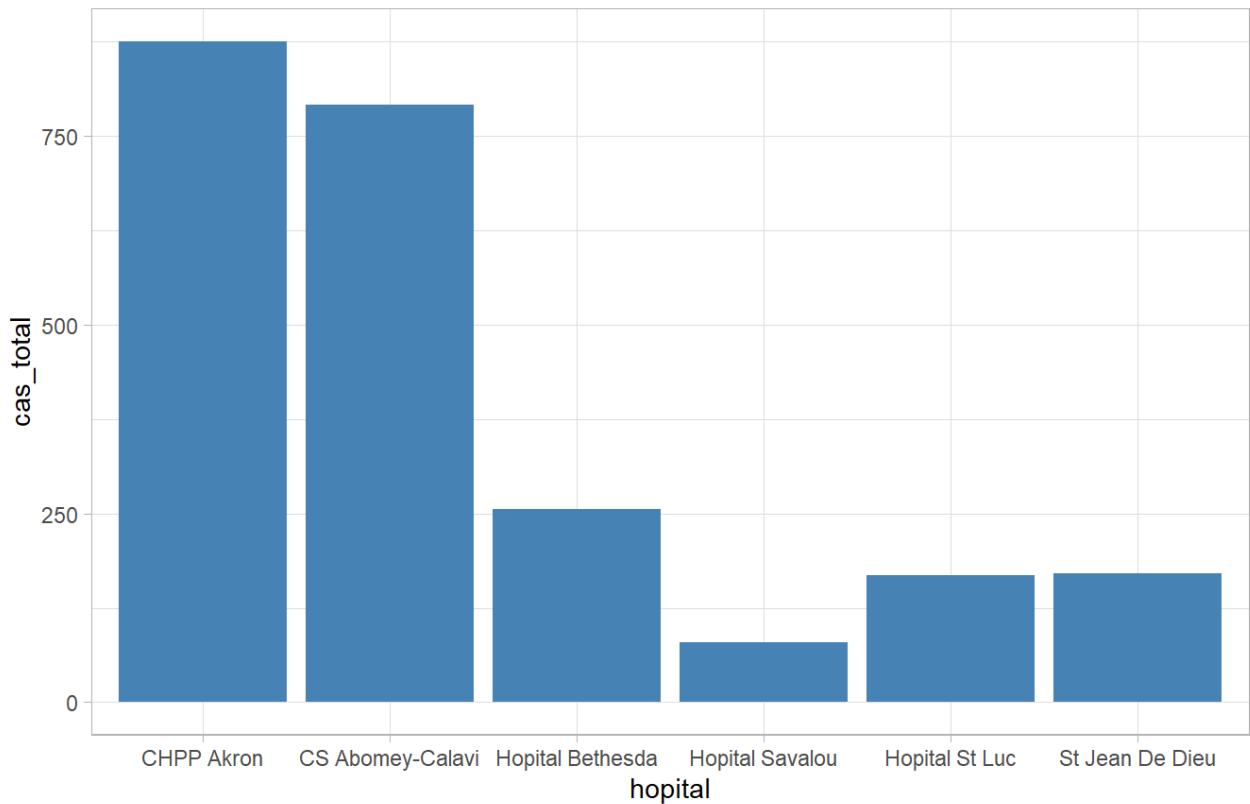
hopital_barre <- hopital_cas_total %>%
  # Transmettre les données résumées à ggplot pour la visualisation
  ggplot(
    # Dans aes(), spécifier que l'axe x représentera l' 'hopital',
    # la hauteur des barres (axe y) représentera le 'cas_total',
    aes(x = hopital, y = cas_total)) +
  # Utiliser geom_col() pour créer un diagramme à barres bleu
  geom_col(fill = "steelblue") +
  labs(title = "Nouveaux cas et rechutes de TB par hôpital",
       subtitle = "Données de six établissements de santé au Bénin, 2015-2017")

hopital_barre

```

Nouveaux cas et rechutes de TB par hôpital

Données de six établissements de santé au Bénin, 2015-2017



Ce graphique semble identique à celui créé avec `resultats_tb` mais la différence clé est que la hauteur des barres sur l'axe des y dans `hopital_cas_total` est précalculée, plutôt que faite avec `geom_col()`.

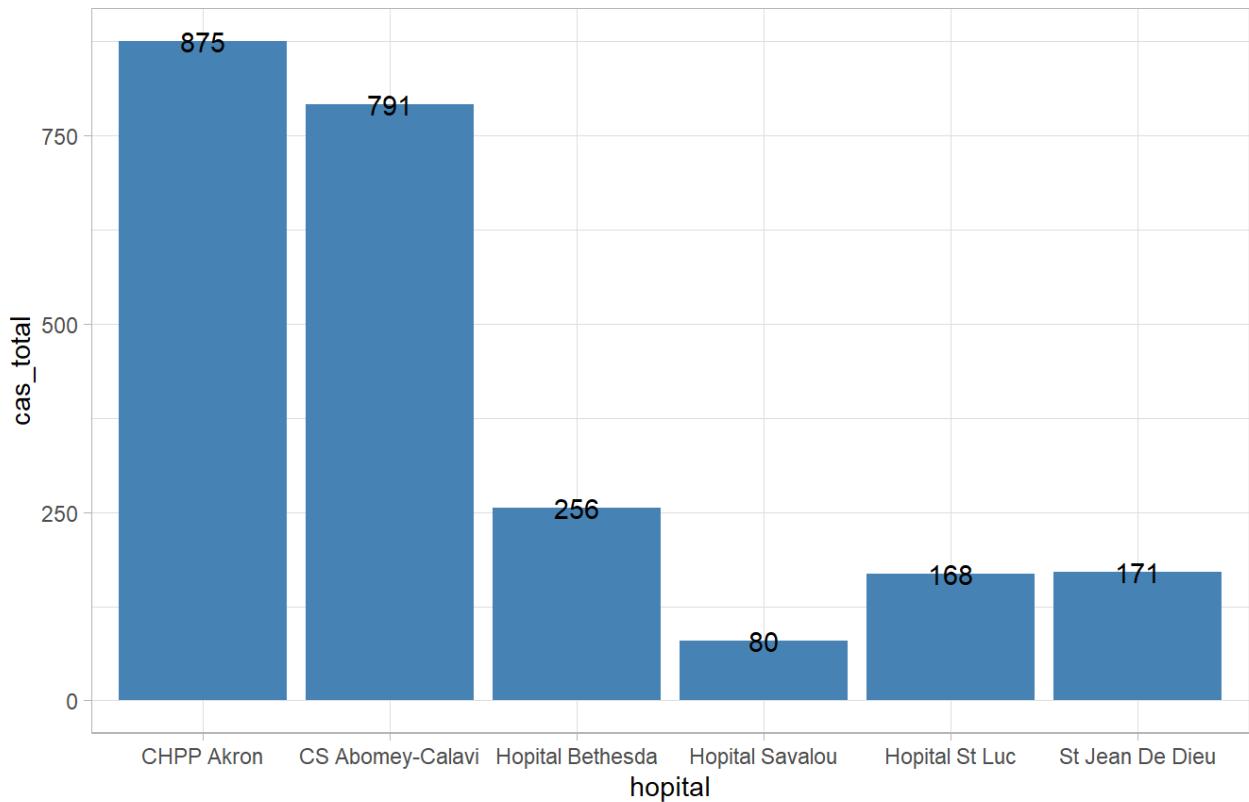
Étape 3 : Annoter le graphique avec `geom_text()` ou `geom_label()`

`geom_text()` fournit un moyen simple et efficace d'ajouter directement du texte à vos graphiques. Explorons comment nous pouvons l'utiliser pour mettre en évidence le nombre total de cas de tuberculose par hôpital dans notre diagramme à barres

`hopital_barre`

```
# Donner les étiquettes précalculé à geom_text()
hopital_barre +
  geom_text(aes(label = cas_total))
```

Nouveaux cas et rechutes de TB par hôpital
Données de six établissements de santé au Bénin, 2015-2017



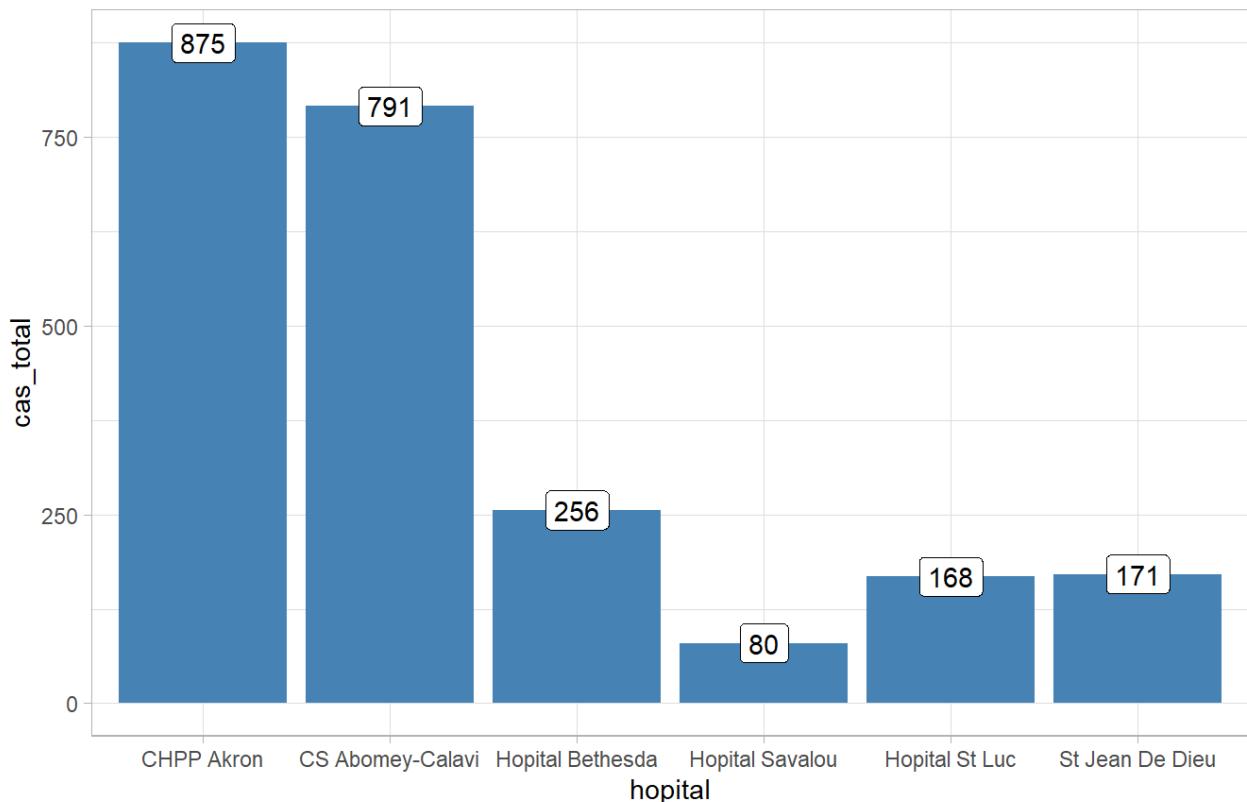
Voilà, c'est beaucoup mieux.

Pour les étiquettes qui doivent ressortir davantage, `geom_label()` dessine un rectangle derrière le texte, améliorant le contraste.

```
# Donner les étiquettes précalculé à geom_label()
hopital_barre +
  geom_label(aes(label = cas_total))
```

Nouveaux cas et rechutes de TB par hôpital

Données de six établissements de santé au Bénin, 2015-2017



Étape 4 : Ajuster le positionnement et le style du texte

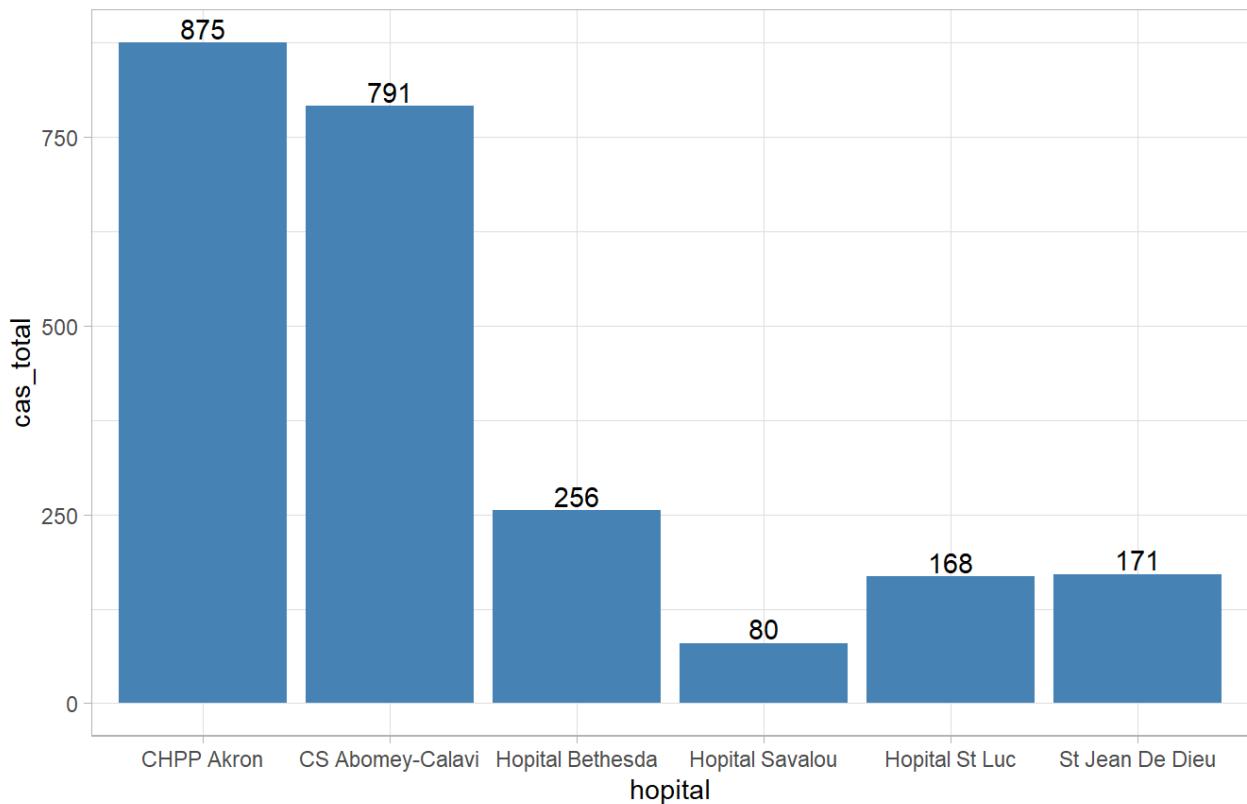
Un argument important à utiliser avec `geom_text()` et d'autres fonctions d'étiquetage dans `{ggplot2}` est `vjust`. Cet argument peut ajuster la position verticale du texte, c'est-à-dire le déplacer vers le haut ou vers le bas.

Réglez `vjust` sur un nombre négatif pour le déplacer vers le haut :

```
# Positionner le texte au-dessus des barres
hopital_barre +
  geom_text(aes(label = cas_total),
            # AJUSTER LA HAUTEUR VERTICALE VERS LE HAUT
            vjust=-0.2)
```

Nouveaux cas et rechutes de TB par hôpital

Données de six établissements de santé au Bénin, 2015-2017

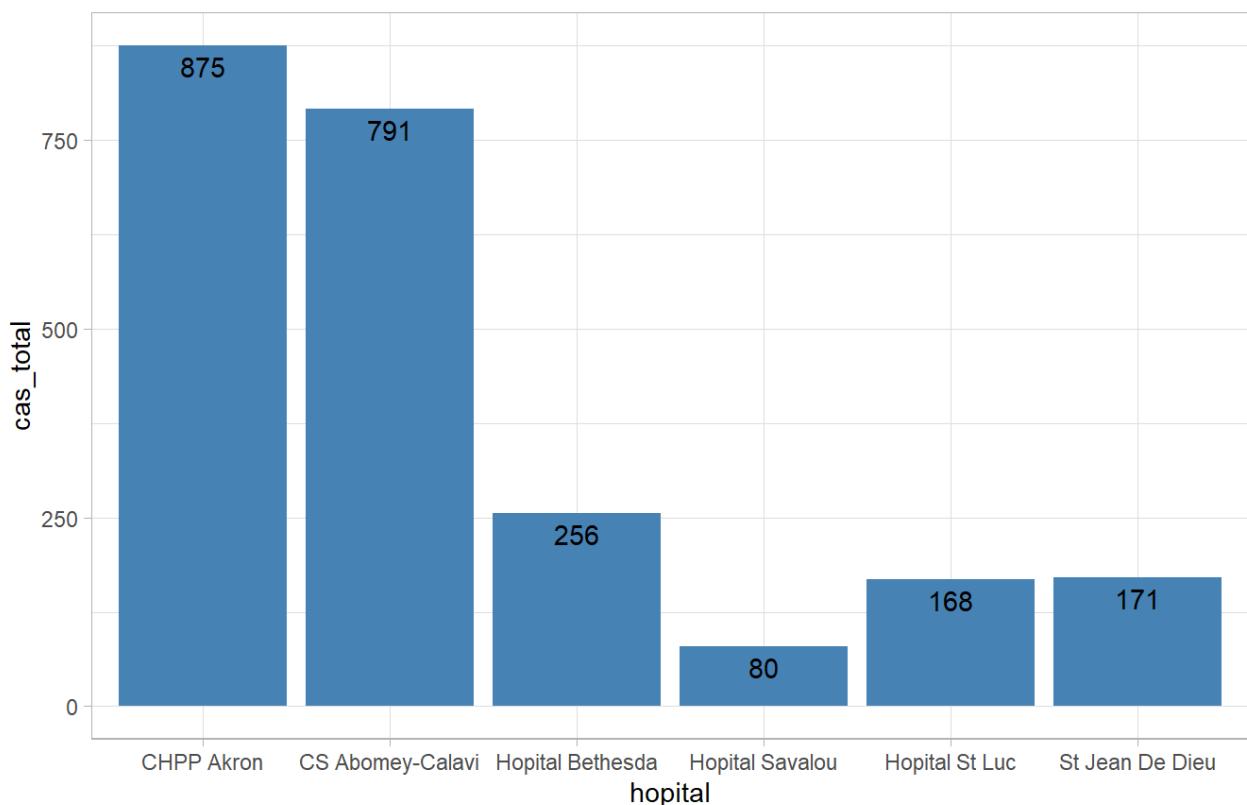


... ou un nombre positif pour déplacer le texte vers le bas :

```
# Positionner le texte à l'intérieur des barres
hopital_barre +
  geom_text(aes(label = cas_total),
            # AJUSTER LA HAUTEUR VERTICALE VERS LE BAS
            vjust=1.5)
```

Nouveaux cas et rechutes de TB par hôpital

Données de six établissements de santé au Bénin, 2015-2017

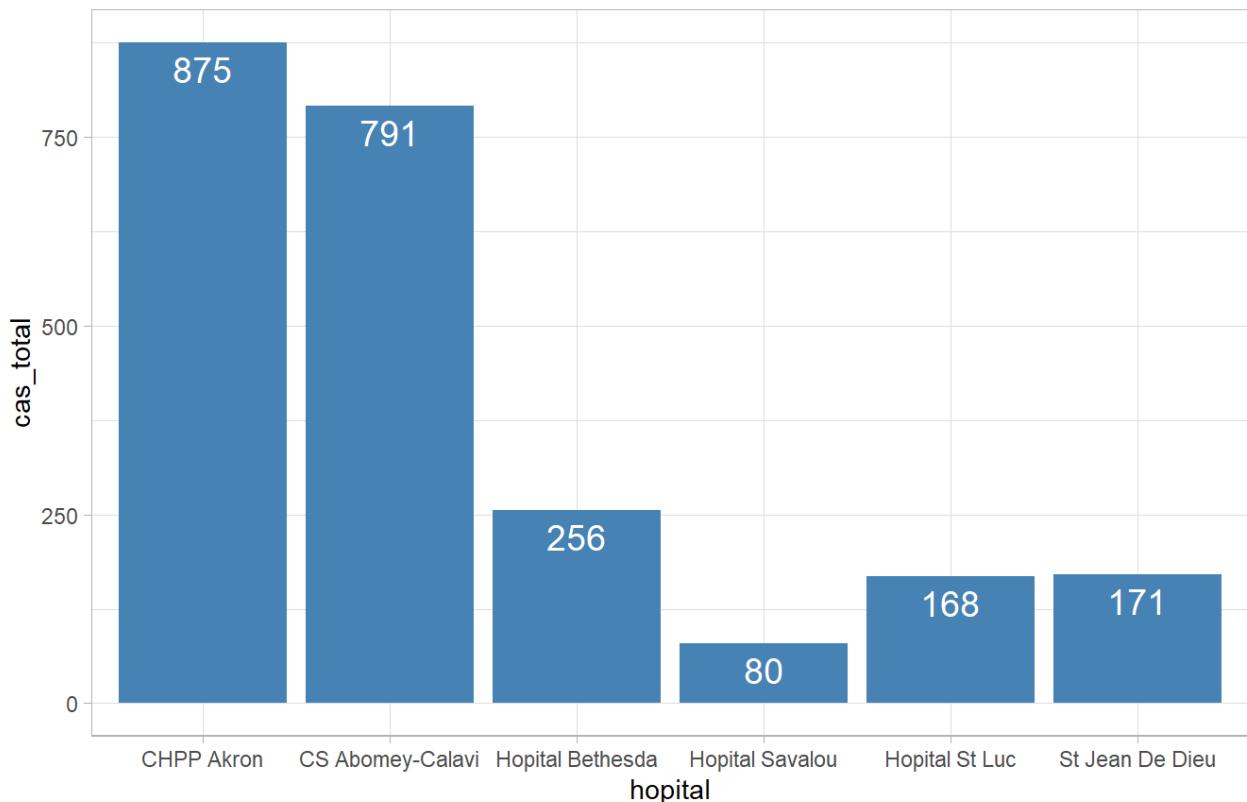


Nous pouvons utiliser des arguments supplémentaires dans `geom_text()` pour modifier la couleur et la taille du texte.

```
# Ajouter des ajustements (esthétiques fixes) à geom_text()
hopital_barre +
  geom_text(aes(label = cas_total),
            vjust=1.5,
            color="white",
            size=5)
```

Nouveaux cas et rechutes de TB par hôpital

Données de six établissements de santé au Bénin, 2015-2017

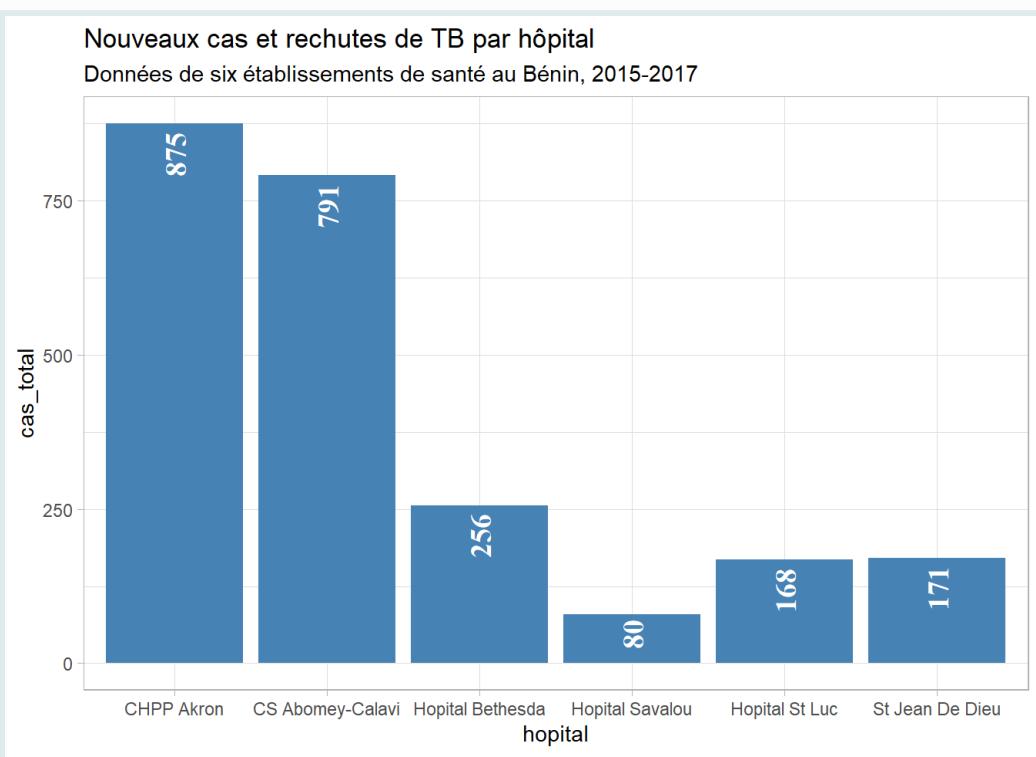


Modifications esthétiques Jusqu'à présent, nous n'avons utilisé que quelques-unes des esthétiques possibles pour `geom_text()` et `geom_label()`. Les trois esthétiques minimales sont `x`, `y` et `label`. Celles-ci doivent être mappées à une variable définie dans `aes()`. En plus des esthétiques requises, ces fonctions acceptent un certain nombre d'esthétiques optionnelles pour personnaliser le texte. Par exemple :

PRO TIP



```
# Ajustements supplémentaires (esthétiques fixes) dans
# geom_text()
hopital_barre + geom_text(aes(label = cas_total),
                           vjust = 1.5,
                           color = "white",
                           size = 5,
                           family = "serif",
                           fontface = "bold",
                           angle = 90,
                           # Utiliser hjust au lieu de vjust car le texte est
                           # pivoté de 90 degrés
                           hjust = 1.2)
```



Apprenez-en plus sur la définition de ces esthétiques dans `vignette("ggplot2-specs")`. Exécutez ce code dans votre console et faites défiler jusqu'à la section "Texte" de la vignette.

Bien que ce graphique à barres mettant en avant une seule variable catégorielle soit agréable, soyons réalistes - la plupart du temps, nous avons affaire à plusieurs variables catégorielles. Il se peut que nous ayons besoin de les empiler, de les aligner côté à côté, voire de les transformer en graphiques circulaires. C'est là que les choses deviennent plus complexes.

Étiquetage des graphiques à barres empilées

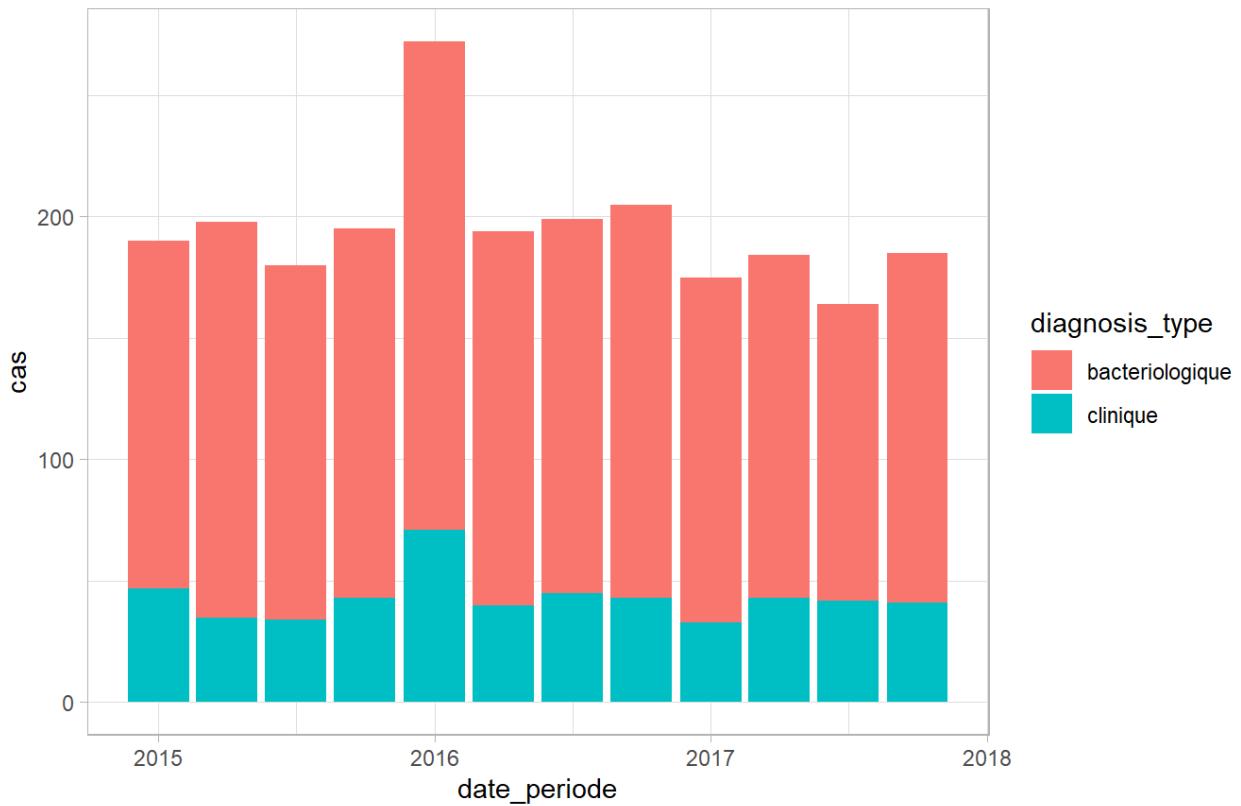
Construisons des graphiques avec deux variables catégorielles, chacune ayant plusieurs niveaux, et ajoutons des étiquettes à chaque sous-groupe, en commençant par des graphiques à barres empilées.

Tout d'abord, créons un graphique à barres empilées de base, sans étiquettes.

```
# Graphique à barres empilées de base
resultats_tb %>%
  ggplot(aes(x = date_periode, y = cas)) +
  # Cartographier la variable de remplissage dans geom_col() pour empiler les
  # barres
  geom_col(aes(fill = diagnosis_type)) +
  labs(title = "Nouveaux et rechutes de cas de tuberculose par trimestre",
       subtitle = "Données de six établissements de santé au Bénin, 2015-2017")
```

Nouveaux et rechutes de cas de tuberculose par trimestre

Données de six établissements de santé au Bénin, 2015-2017



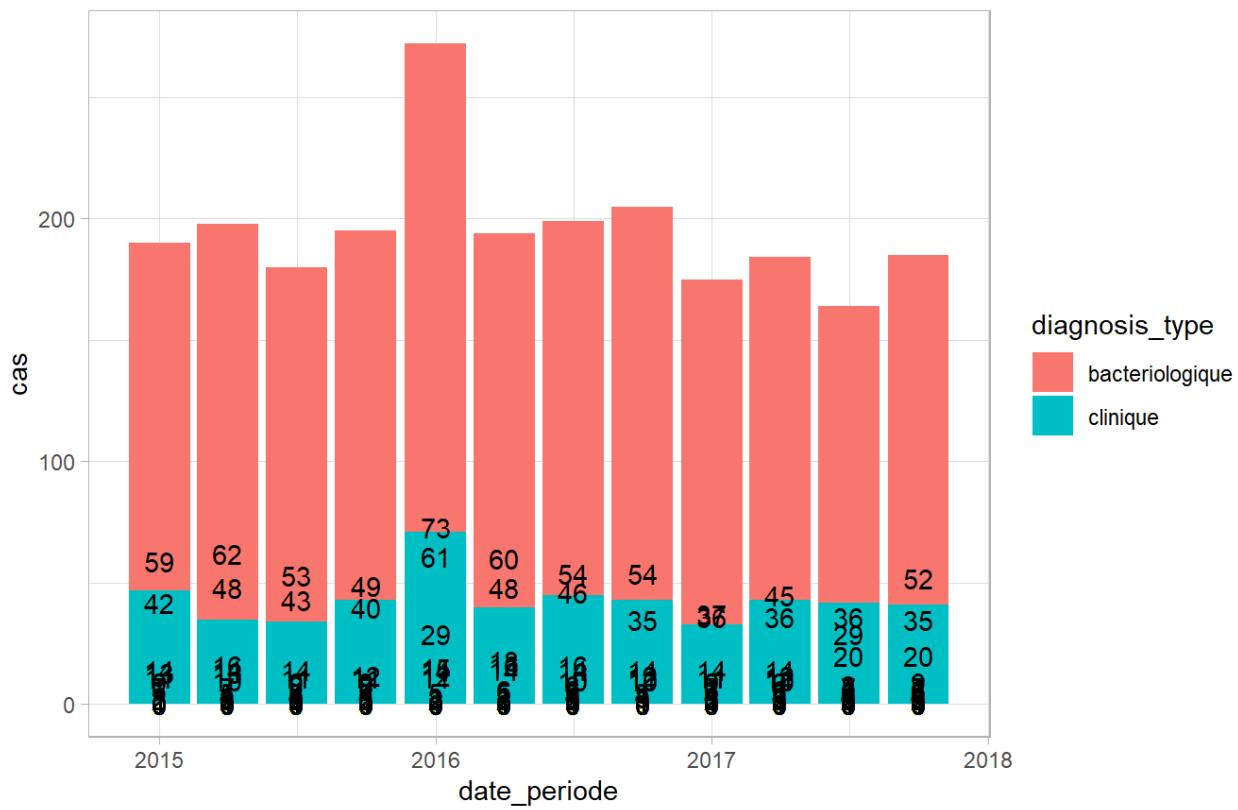
En général, le code reste assez similaire à notre graphique à barres avec une seule catégorie, mais au lieu de remplir les barres avec une couleur spécifique ("steelblue") comme nous l'avons fait auparavant, nous les remplissons maintenant en fonction d'une deuxième variable catégorielle.

Comme nous l'avons déjà constaté, simplement ajouter `geom_text()` à ce code n'aura pas l'effet désiré.

```
# Graphique à barres empilées avec geom_text() - code incorrect
resultats_tb %>%
  ggplot(aes(x = date_periode, y = cas)) +
  geom_col(aes(fill = diagnosis_type)) +
  labs(title = "Nouveaux et rechutes de cas de tuberculose par trimestre",
       subtitle = "Données de six établissements de santé au Bénin, 2015-2017") +
  geom_text(aes(label = cas))
```

Nouveaux et rechutes de cas de tuberculose par trimestre

Données de six établissements de santé au Bénin, 2015-2017



Rappelez-vous - nous devons résumer les données pour précalculer les étiquettes de texte. Le cadre de données de résumé doit avoir le même nombre de lignes que le nombre d'étiquettes souhaité. Dans ce cas, nous aurons besoin d'une étiquette pour chaque segment de barre - un total de 24 étiquettes.

Étape 1 : Résumer les données

Ce morceau de code résume d'abord notre jeu de données `resultats_tb` par `periode` et `diagnosis_type`, calculant la somme des cas (`cas`) pour chaque groupe.

```
# Résumer les données par période et type de diagnostic
tb_total <- resultats_tb %>%
  group_by(date_periode, diagnosis_type) %>%
  summarise(cas = sum(cas))

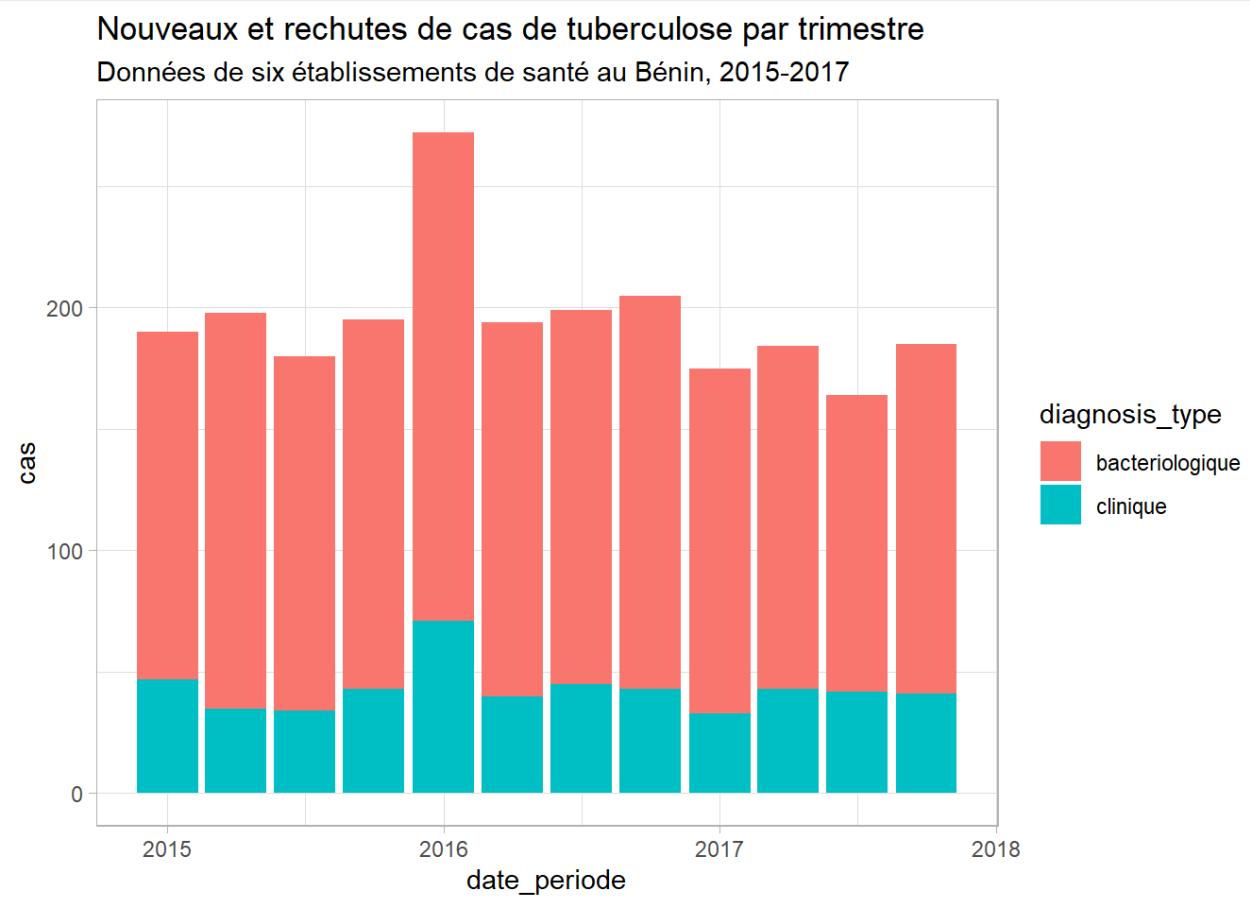
tb_total
```

Étape 2 : Créer le graphique de base

Ensuite, créons un simple graphique à barres empilées : Nous allons représenter les données résumées sous forme d'un graphique à barres (`geom_col()`), où la hauteur de chaque barre représente le nombre total de cas pour un diagnostic particulier dans chaque période.

```
# Crée un graphique à barres de base utilisant les données résumées
periode_dx_barres <- tb_total %>%
  ggplot(aes(x = date_periode, y = cas, fill = diagnosis_type)) +
  geom_col() +
  labs(title = "Nouveaux et rechutes de cas de tuberculose par trimestre",
       subtitle = "Données de six établissements de santé au Bénin, 2015-2017")

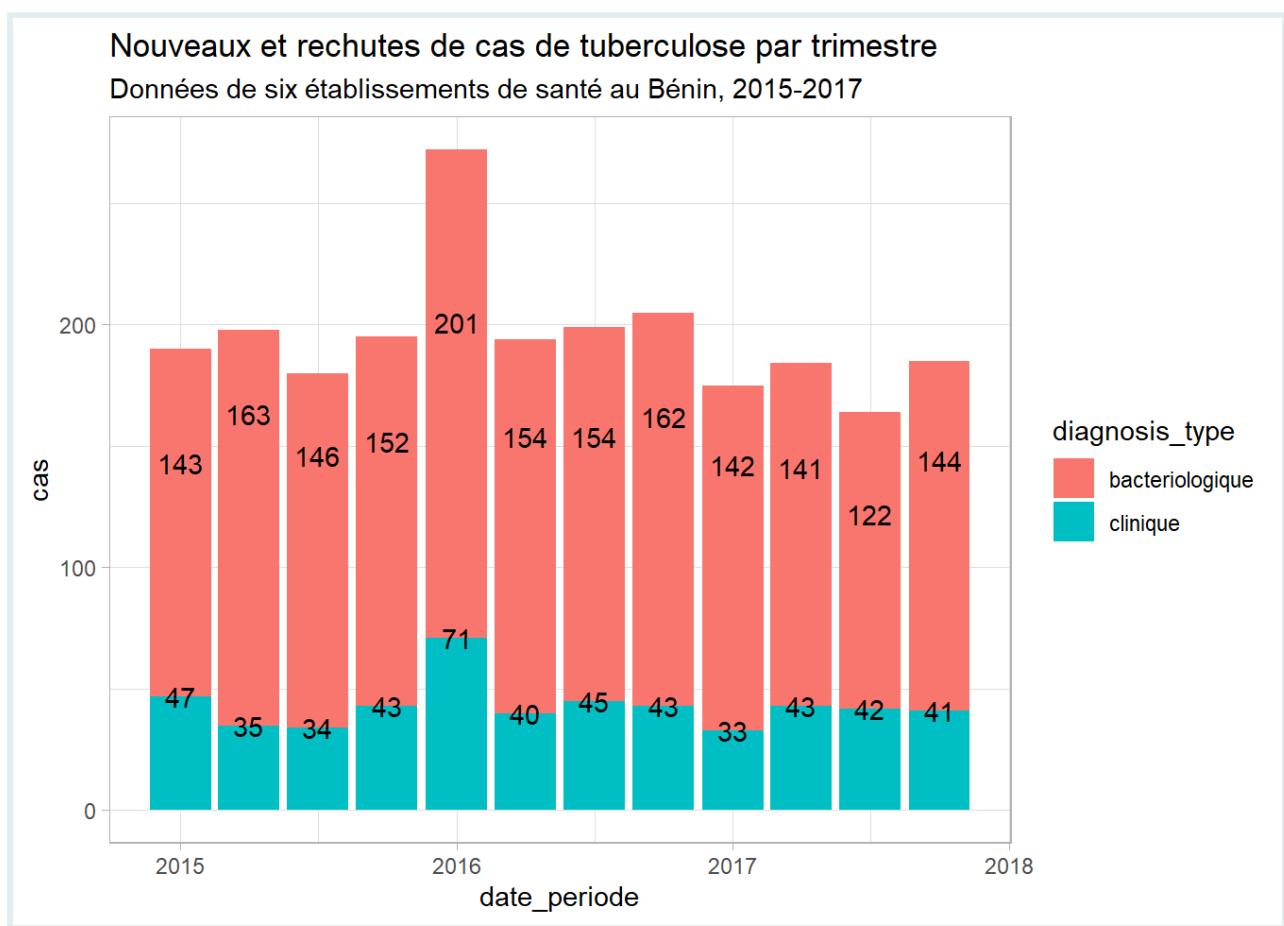
periode_dx_barres
```



Étape 3 : Annoter le graphique avec `geom_text()` ou `geom_label()`

- **Ajouter du texte au graphique à barres :** Ensuite, nous ajoutons des étiquettes à ce graphique. La colonne `cas`, qui représente le nombre total de cas, sera utilisée pour étiqueter chaque barre.

```
# Ajouter des étiquettes de texte au graphique à barres
periode_dx_barres +
  geom_text(aes(label = cas))
```

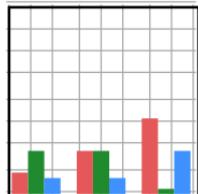


Ici, `geom_text()` est utilisé pour placer des étiquettes sur le dessus de chaque barre. Les étiquettes montrent le nombre total de cas (`cas`) pour chaque type de diagnostic durant chaque période.

Mais les étiquettes ne sont pas au bon endroit; elles ne correspondent pas à la hauteur des barres dans le graphique à barres empilées. Cela arrive parce que nous devons indiquer au graphique comment positionner chaque élément, et actuellement, nous l'avons fait uniquement pour les barres (`geom_col()`).

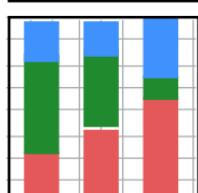
Même si cela peut ne pas être évident dans le code, lorsque nous utilisons `fill = variable` dans `aes()` pour `geom_col()`, cela a pour effet par défaut de les empiler (en anglais: "stack"). Pour résoudre cela, il est essentiel d'aussi spécifier

explicitement l'ajustement de la position dans `geom_text()`, assurant ainsi une cohérence dans leur configuration.



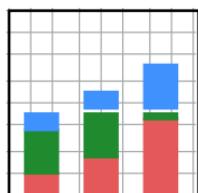
s + geom_bar(position = "dodge")

Arrange elements side by side



s + geom_bar(position = "fill")

Stack elements on top of one another, normalize height



s + geom_bar(position = "stack")

Stack elements on top of one another

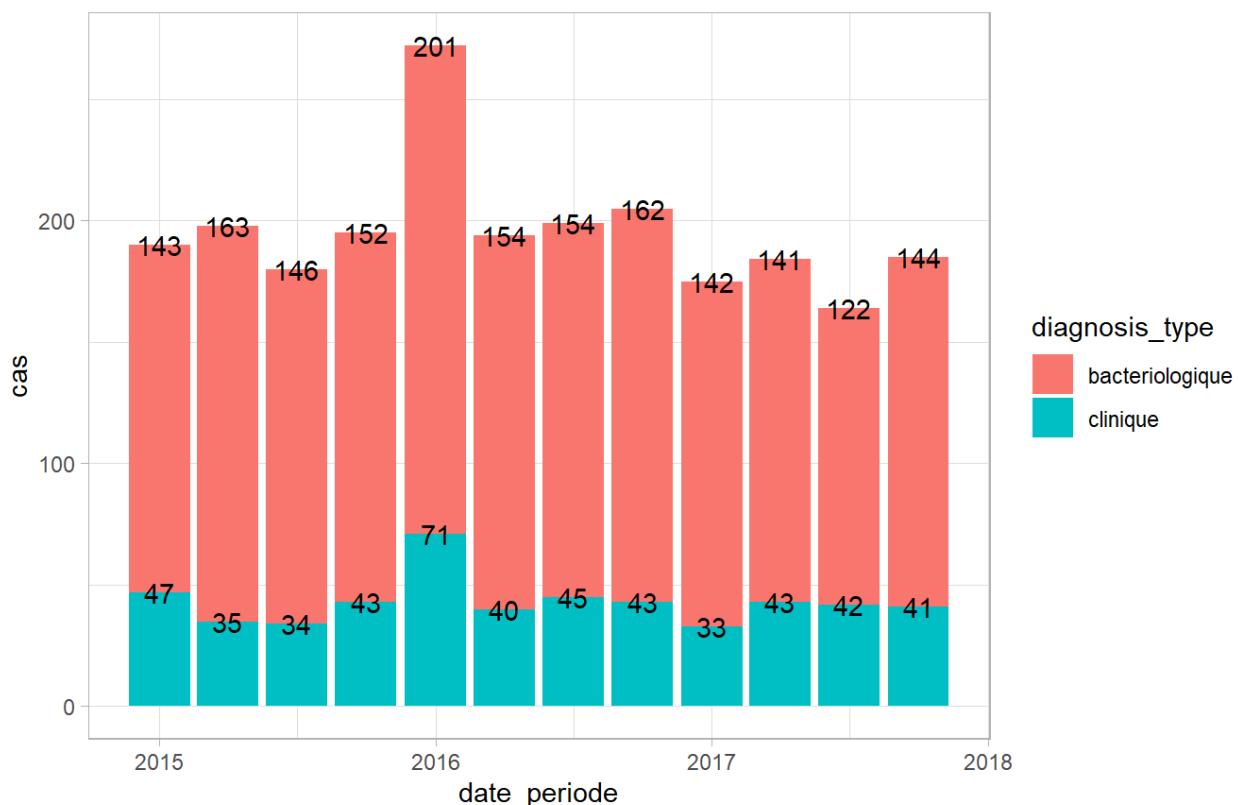
Étape 4 : Ajuster la position du texte pour l'aligner avec les barres

Si nous ajoutons un argument `position` à `geom_text()` et lui demandons d'empiler les étiquettes, nous obtiendrons des étiquettes en haut de chaque segment de barre.

```
# Placer le texte en haut de chaque segment de barre
periode_dx_barres +
  geom_text(
    aes(label = cas),
    position = position_stack()) # Définir la position en tant que stack
```

Nouveaux et rechutes de cas de tuberculose par trimestre

Données de six établissements de santé au Bénin, 2015-2017



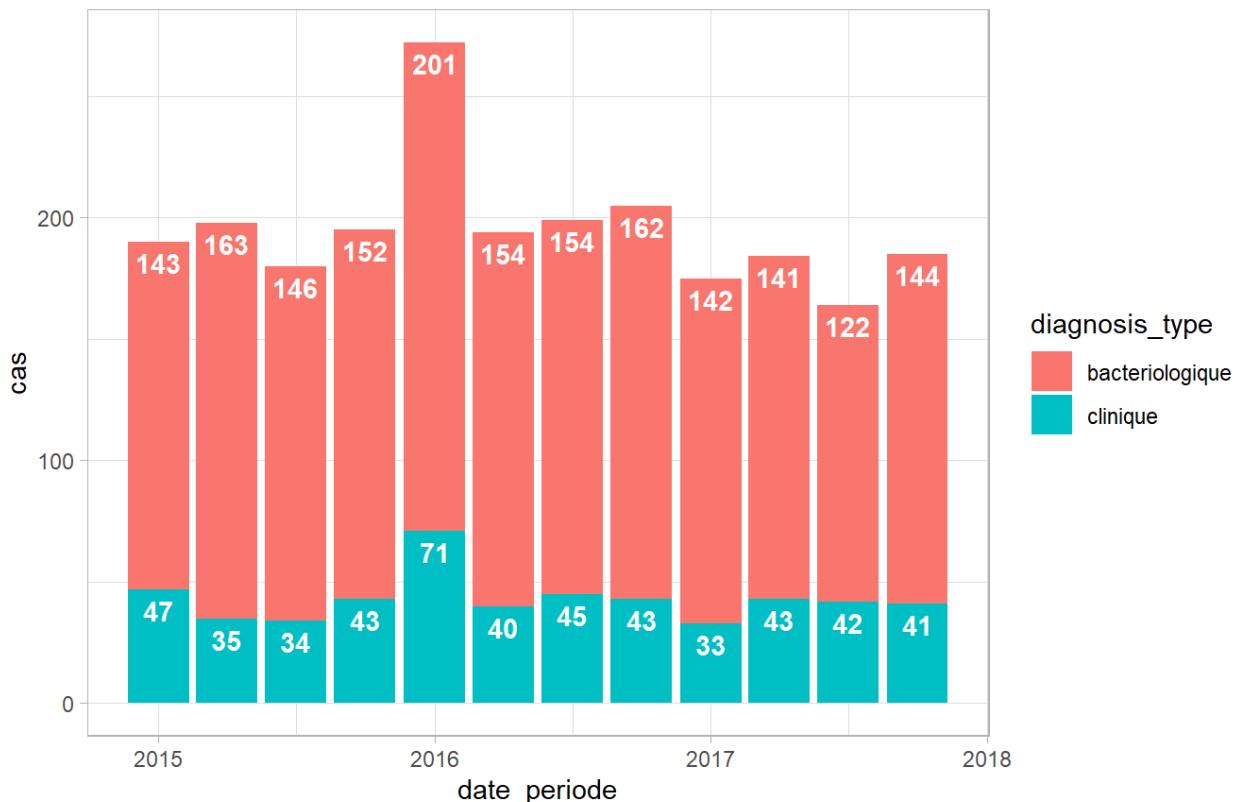
C'est corrigé !

Pour aligner verticalement le texte à l'intérieur des barres, nous pouvons ajouter `vjust` à `geom_text()` comme d'habitude.

```
# Repositionner les étiquettes à l'intérieur des empilements pour plus de
# clarté et changer le style de police
periode_dx_barres +
  geom_text(aes(label = cas),
            position = position_stack(),
            vjust = 1.5,
            color = "white",
            fontface = "bold")
```

Nouveaux et rechutes de cas de tuberculose par trimestre

Données de six établissements de santé au Bénin, 2015-2017

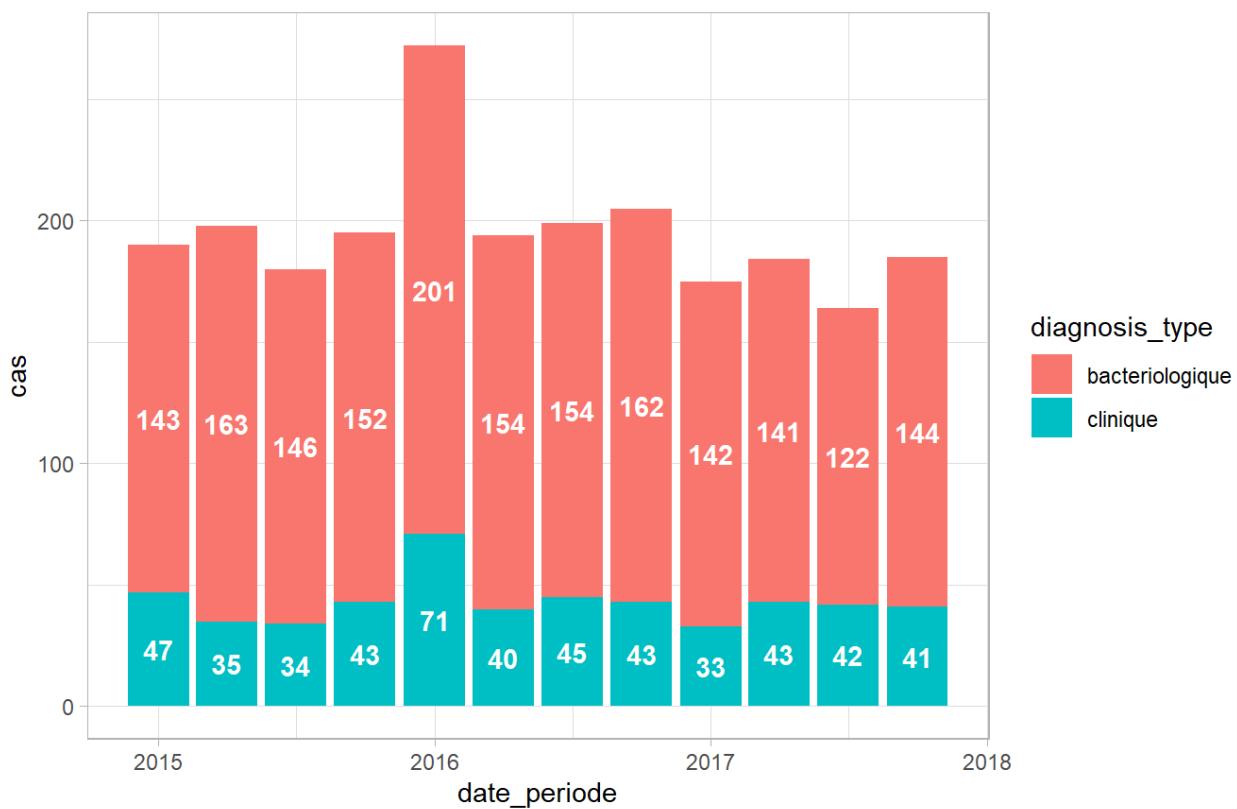


Plutôt que de déplacer les étiquettes vers le haut ou vers le bas d'une quantité fixe, nous pouvons également spécifier la hauteur de l'étiquette par rapport au segment de barre correspondant. Nous pouvons ajuster l'emplacement de l'étiquette au sein de chaque empilement comme suit :

```
# Pour placer le texte au milieu de chaque segment de barre dans un graphique
# à barres empilées, vous devez définir le paramètre vjust de
# position_stack()
periode_dx_barres +
  geom_text(
    aes(label = cas),
    # AJOUTER LA FONCTION DE POSITION APPROPRIÉE
    position = position_stack(vjust = 0.5),
    color = "white",
    fontface = "bold")
```

Nouveaux et rechutes de cas de tuberculose par trimestre

Données de six établissements de santé au Bénin, 2015-2017



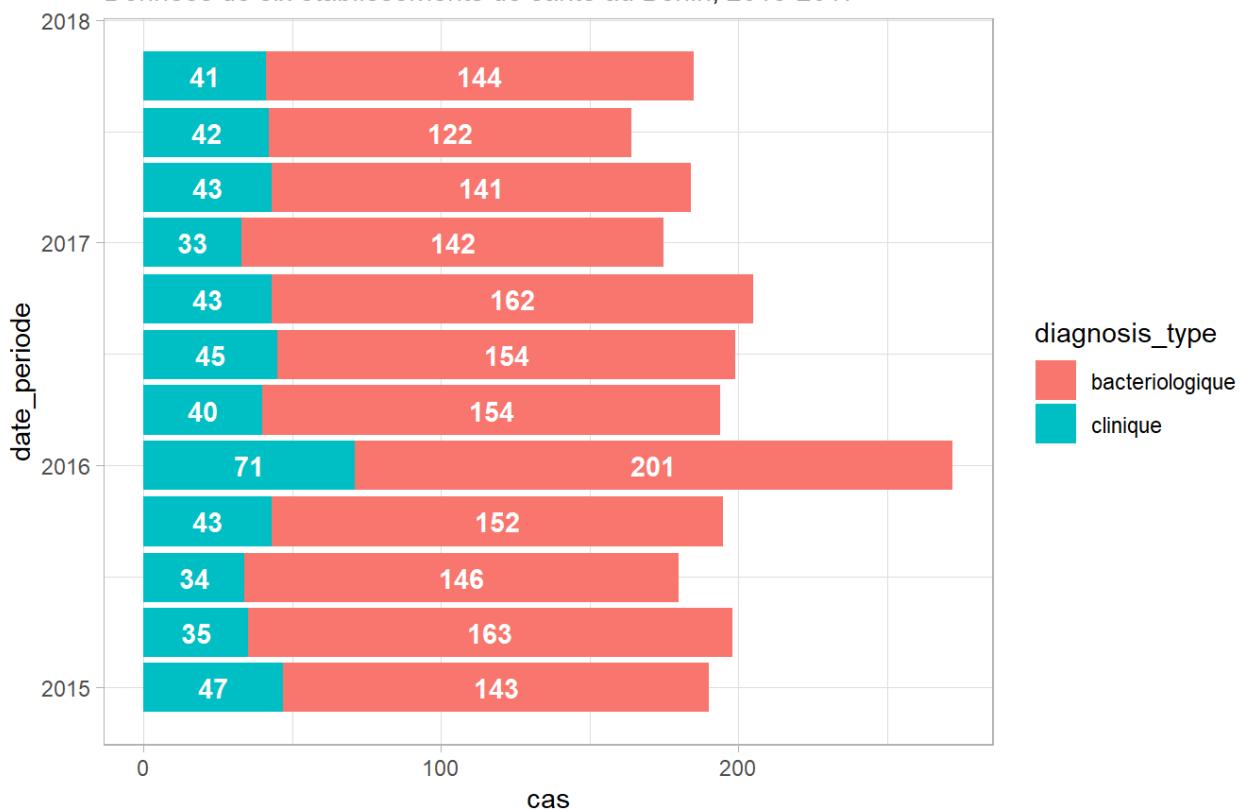
Ce code modifie le graphique précédent en utilisant `position_stack(vjust = 0.5)` dans `geom_text()`. Cet ajustement centre verticalement les étiquettes dans chaque segment des barres empilées, facilitant leur association avec le segment de barre pertinent.

Cette disposition d'étiquette est particulièrement agréable pour les graphiques à barres horizontaux.

```
# Convertir en graphique horizontal avec coord_flip()
periode_dx_barres +
  geom_text(
    aes(label = cas),
    position = position_stack(vjust = 0.5),
    color = "white",
    fontface = "bold") +
  coord_flip() # Inverser l'orientation des barres
```

Nouveaux et rechutes de cas de tuberculose par trimestre

Données de six établissements de santé au Bénin, 2015-2017



Étiquetage des graphiques à barres groupées

Les graphiques à barres groupées présentent plusieurs catégories juxtaposées, regardons comment regrouper les données et placer correctement les étiquettes pour une interprétation claire.

Étape 1 : Résumer les données

Nous allons d'abord regrouper notre jeu de données `resultats_tb` par `hopital` et `diagnosis_type` et calculer la somme des cas (`cas`) pour chaque groupe.

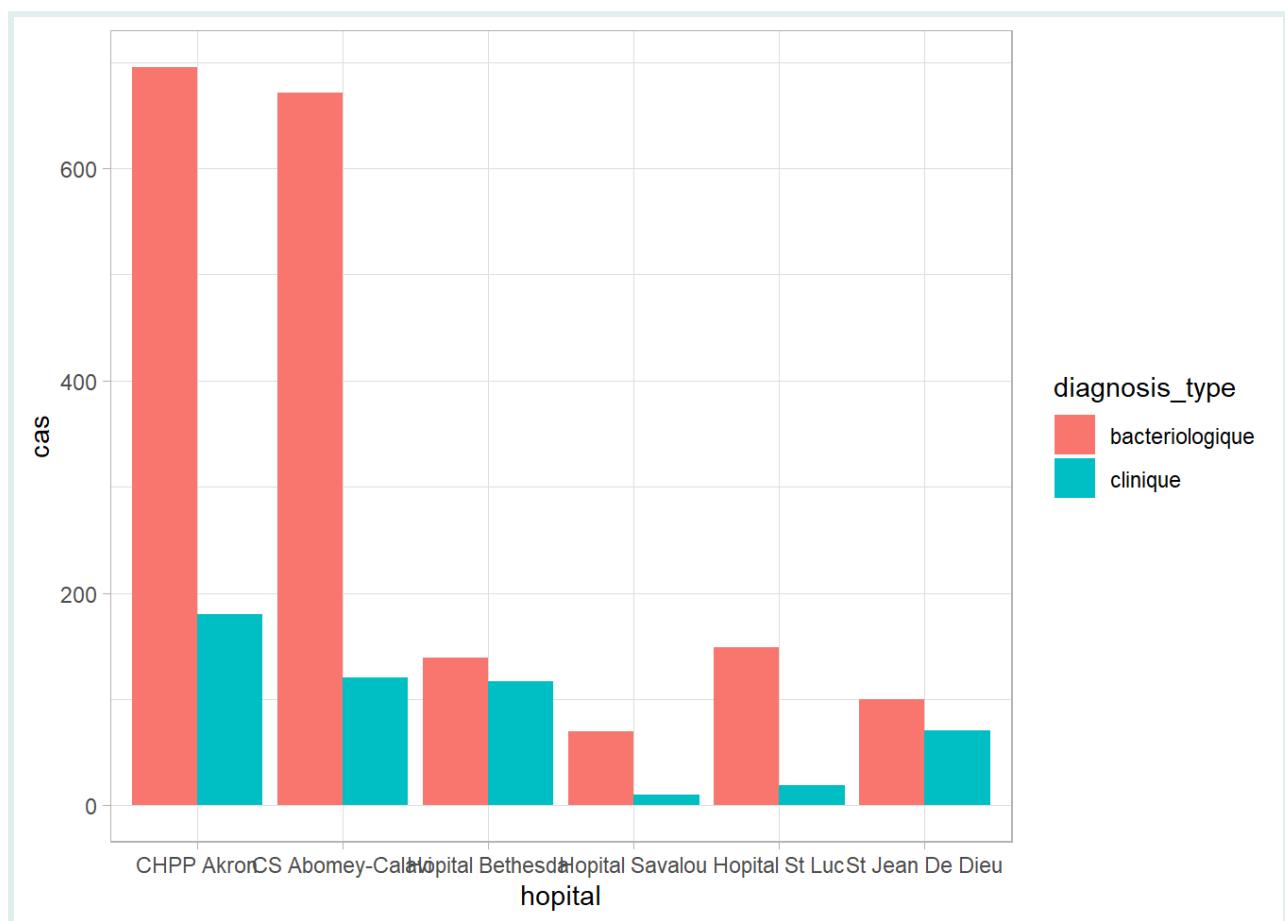
```
hopital_dx_cas <- resultats_tb %>%
  group_by(hopital, diagnosis_type) %>%
  summarise(cas = sum(cas))

hopital_dx_cas
```

Étape 2 : Créer le graphique de base

Ensuite, créons un simple graphique à barres groupées : Nous allons représenter les données résumées sous forme d'un graphique à barres regroupées (`geom_col()`), où la hauteur de chaque barre représente le nombre total de cas pour un diagnostic spécifique pour chaque hôpital. Le paramètre par défaut de `geom_col` est `stack`. Pour créer un graphique à barres groupées, nous allons spécifier `position = position_dodge()`.

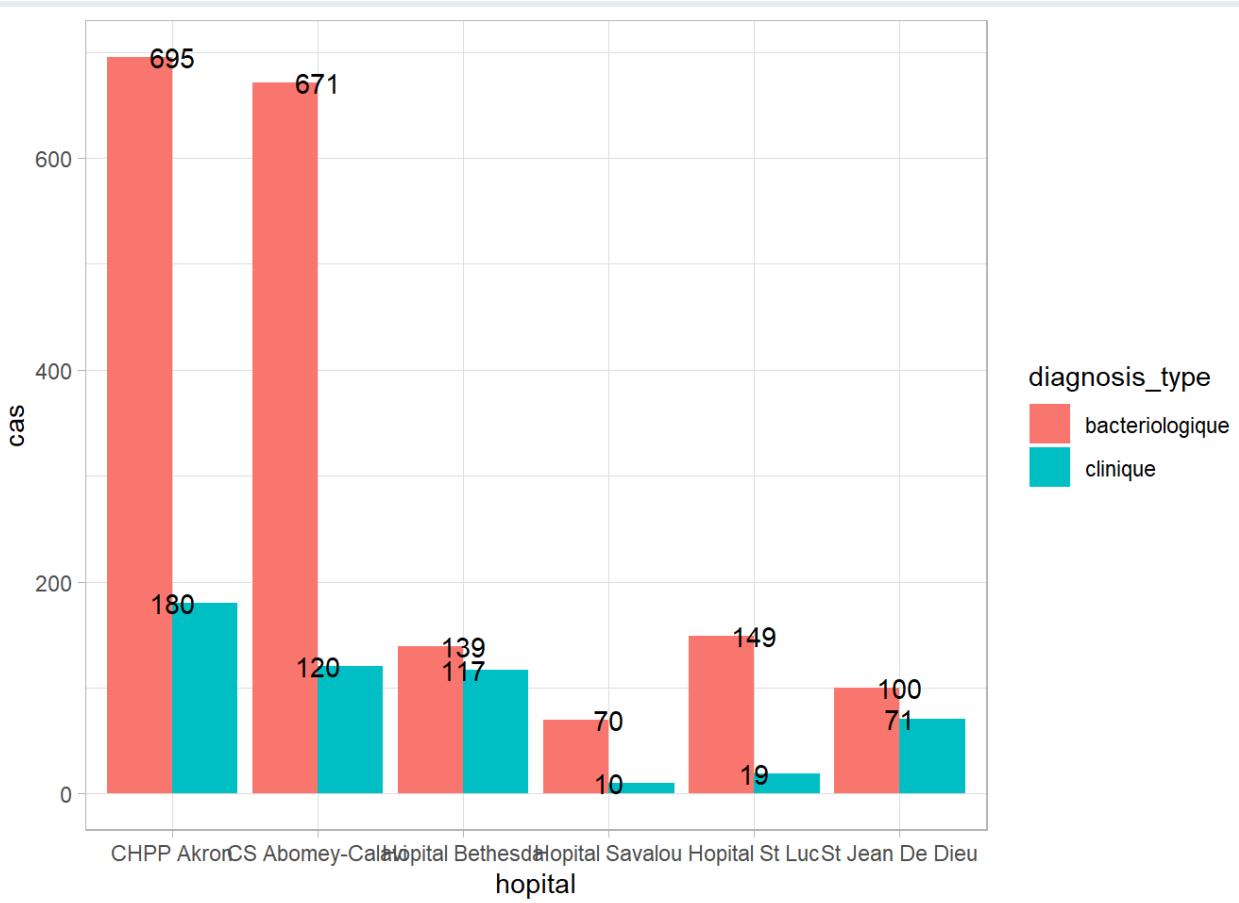
```
# utiliser "dodge" au lieu de la valeur par défaut "stack"  
hopital_dx_barre <- hopital_dx_cas %>%  
  ggplot(aes(x = hopital, y = cas, fill = diagnosis_type)) +  
  geom_col(position = position_dodge())  
  
hopital_dx_barre
```



Étape 3 : Annoter le graphique avec `geom_text()`

Maintenant, nous pouvons annoter le graphique avec `geom_text()` pour afficher des étiquettes, tout comme nous l'avons fait précédemment.

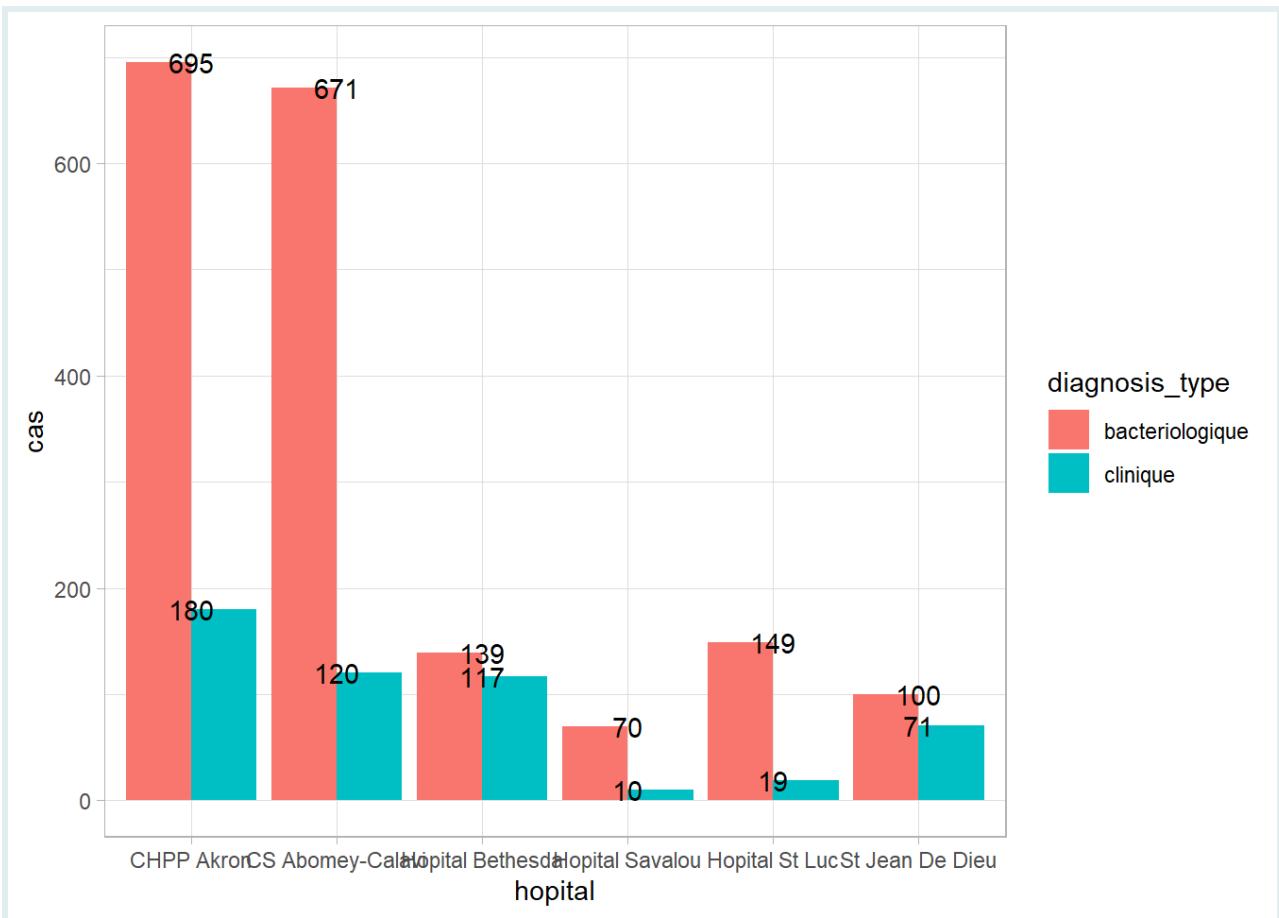
```
# Ajouter geom_text()
hopital_dx_barre +
  geom_text(aes(label = cas))
```



Étape 4: Ajuster la position du texte pour l'aligner avec les barres

Les étiquettes sont centrées verticalement sur une ligne droite, non alignées avec les barres juxtaposées. Encore une fois, comme pour les barres empilées dans la section précédente, nous devons ajouter l'ajustement de position à `geom_text()`.

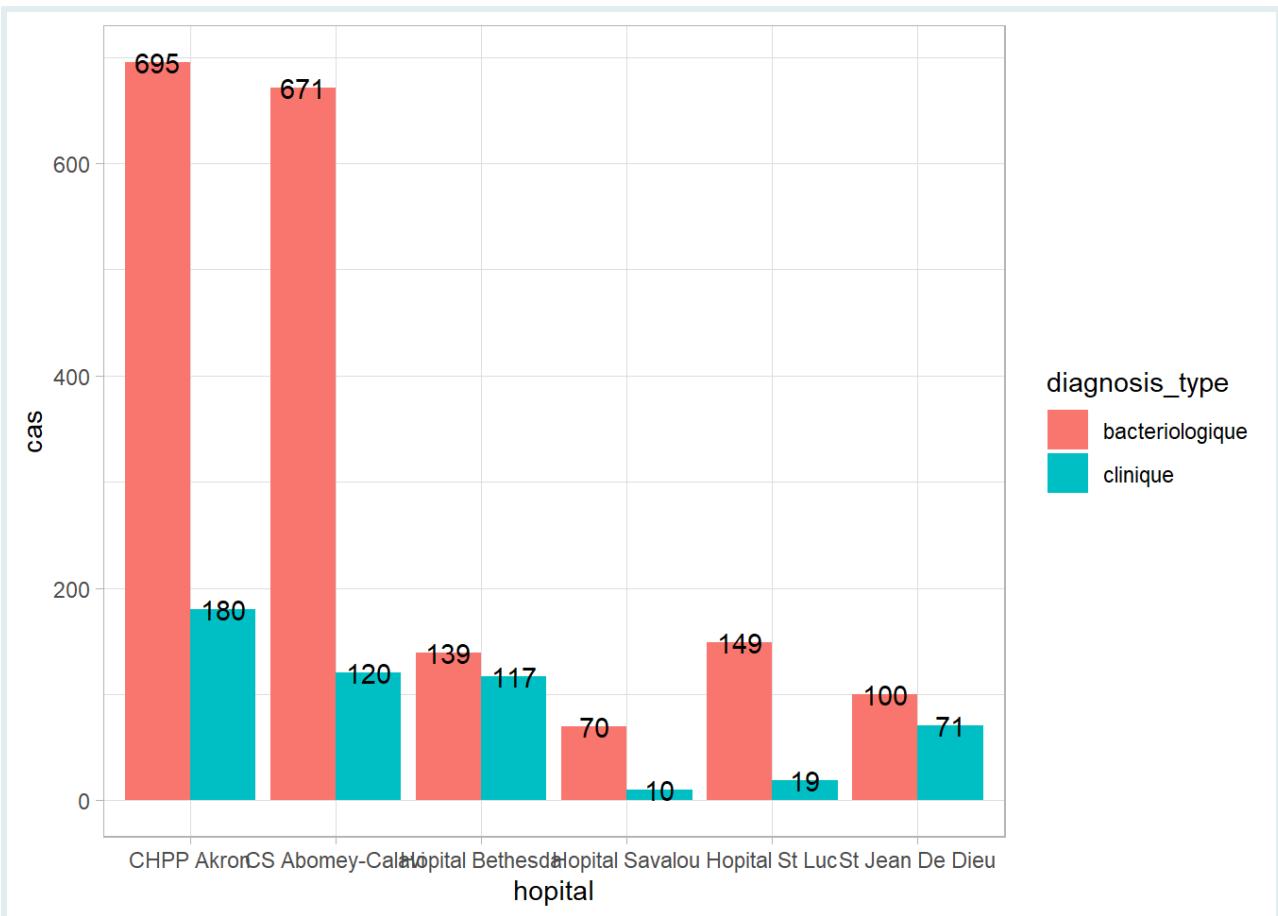
```
# Spécifier la position dodge à `geom_text()`
hopital_dx_barre +
  geom_text(aes(label = cas),
            position = position_dodge())
```



Nous obtenons le même graphique qu'avant! Lorsque vous utilisez `geom_text()` avec un positionnement `dodge` dans `ggplot2`, spécifier le paramètre `width` devient nécessaire car par défaut, `ggplot2` ne connaît pas la largeur appropriée à utiliser pour décaler les étiquettes.

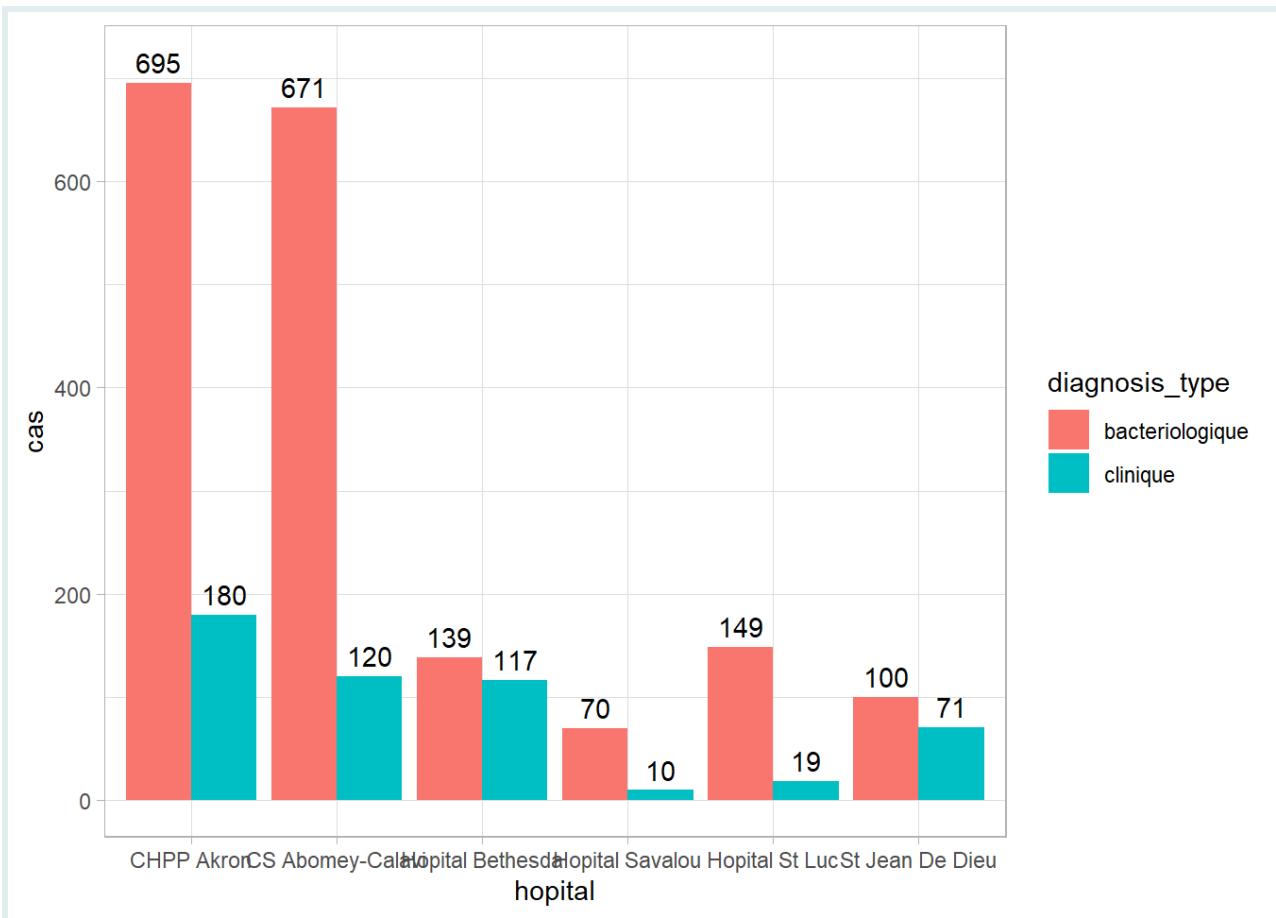
Pour `geom_col()`, la valeur par défaut de `width` est `0.9`. Étant donné que nous n'avons pas spécifier une autre valeur en créant notre graphique, nous allons aussi utiliser `0.9` pour `geom_text()` pour que les barres et les étiquettes s'allignent.

```
# Spécifier à ggplot2 d'aligner le texte en ajoutant la largeur du décalage
hopital_dx_barre +
  geom_text(aes(label = cas),
            position = position_dodge(width = 0.9))
```



Vous ne pouvez pas déplacer le texte aussi facilement avec `vjust`, alors ajustez plutôt la position `y` en ajoutant manuellement une petite quantité.

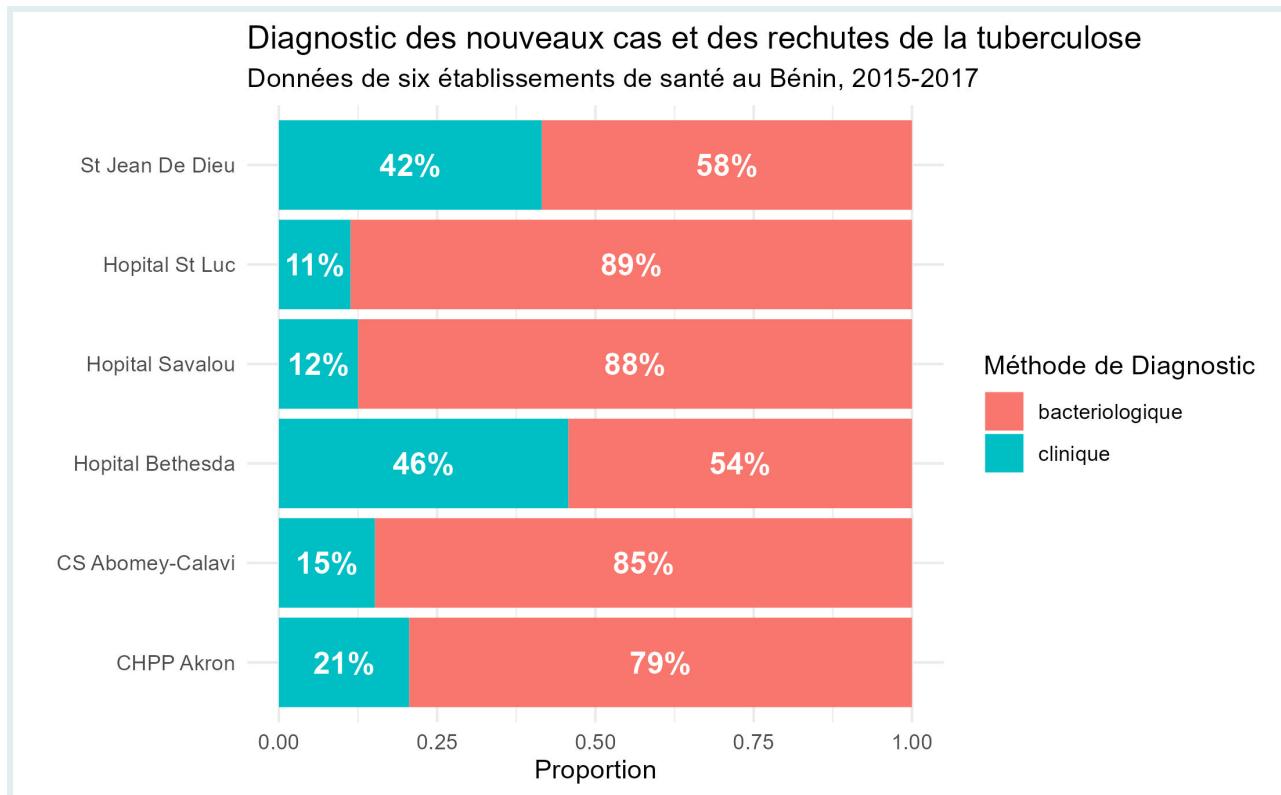
```
# Ajustez les étiquettes verticalement
hopital_dx_barre +
  geom_text(aes(label = cas,
                y = cas + 20), # ajouter 20 à l'axe des y
            position = position_dodge(width = 0.9))
```



Étiquetage des graphiques à barres empilées normalisées

L'étiquetage des graphiques à barres empilées normalisées est un cas particulier car l'axe des y varie de 0 à 1 pour représenter des proportions ou des pourcentages plutôt que des valeurs brutes. Chaque segment à l'intérieur de la barre représente un pourcentage du total, et l'ensemble des segments totalise 100%.

Lors de l'étiquetage des graphiques à barres empilées normalisées, les étiquettes doivent refléter les pourcentages de chaque catégorie. Cela signifie que nous devons formater les étiquettes en pourcentages pour nous assurer qu'elles correspondent aux segments sur le graphique.



Étape 1 : Résumer les données

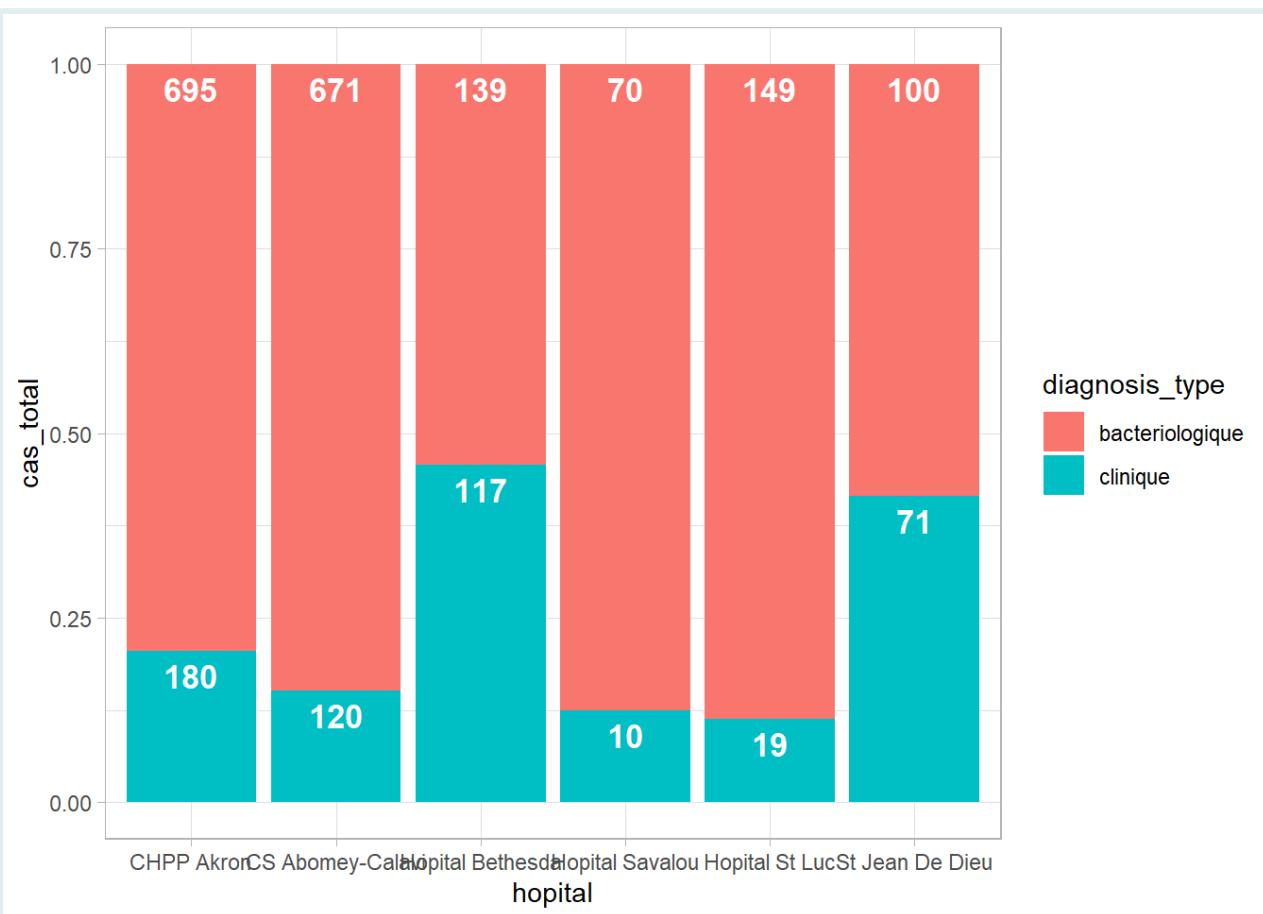
Nous voulons visualiser la proportion de cas dans chaque hôpital appartenant à chaque type de diagnostic. Il faut donc que nous traitons d'abord nos données pour calculer le nombre total de cas pour chaque établissement de santé (`hopital`) puis calculer le pourcentage de chaque type de diagnostic pour chacun d'eux.

```
# Préparation des données pour l'étiquetage de diagrammes à barres empilées à
# 100%
hopital_dx_total <- resultats_tb %>%
  ## regrouper par hôpital et type de diagnostic
  group_by(hopital, diagnosis_type) %>%
  ### résumer nos données par la somme des cas
  summarise(cas_total = sum(cas))

hopital_dx_total
```

Nous pourrions utiliser ceci pour créer un diagramme à barres empilées normalisées comme cela :

```
# Créer un diagramme à barres normalisé avec les données résumées
hopital_dx_total %>%
  ggplot(aes(x = hopital, y = cas_total, fill = diagnosis_type)) +
  geom_col(position = position_fill()) +
  geom_text(aes(label = cas_total),
    ## Définir position = position_fill pour normaliser l'axe y
    position = position_fill(),
    vjust = 1.5,
    color = "white", fontface = "bold", size = 4.5)
```



C'est un bon début, cependant nous voulons des pourcentages, pas des valeurs brutes.

Voici comment nous préparons nos données :

```

# Préparation des données pour l'étiquetage de diagrammes à barres empilées à
# 100% avec des étiquettes en POURCENTAGE
hopital_dx_prop <- resultats_tb %>%
  ## regrouper par hôpital et type de diagnostic
  group_by(hopital, diagnosis_type) %>%
  ## d'abord, nous résumons nos données par la somme des cas, comme d'habitude
  summarise(cas_total = sum(cas)) %>%
  ## calculer les proportions :
  ### ajouter une nouvelle colonne avec les proportions
  mutate(prop = cas_total / sum(cas_total))

hopital_dx_prop

```

```

## # A tibble: 12 × 4
## # Groups:   hopital [6]
##       hopital      diagnosis_type  cas_total    prop
##       <chr>        <chr>           <dbl>     <dbl>
## 1 CHPP Akron bacteriologique     695  0.794
## 2 CHPP Akron clinique          180  0.206
## 3 CS Abomey-Calavi bacteriologique 671  0.848
## 4 CS Abomey-Calavi clinique      120  0.152
## 5 Hopital Bethesda bacteriologique 139  0.543
## 6 Hopital Bethesda clinique      117  0.457
## 7 Hopital Savalou bacteriologique 70  0.875
## 8 Hopital Savalou clinique      10  0.125
## 9 Hopital St Luc bacteriologique 149  0.887
## 10 Hopital St Luc clinique      19  0.113
## 11 St Jean De Dieu bacteriologique 100  0.585
## 12 St Jean De Dieu clinique      71  0.415

```

Dans ce bloc de code, nous commençons par regrouper notre ensemble de données `resultats_tb` par hôpital et type de diagnostic. Nous calculons ensuite la somme des cas pour chaque combinaison, puis calculons la proportion de cas confirmés bactériologiquement et cliniquement diagnostiqués.

Étape 2 : Créer le graphique de base

Ensuite, nous allons créer un graphique à barres en utilisant notre nouveau jeu de données `hopital_dx_prop` :

```

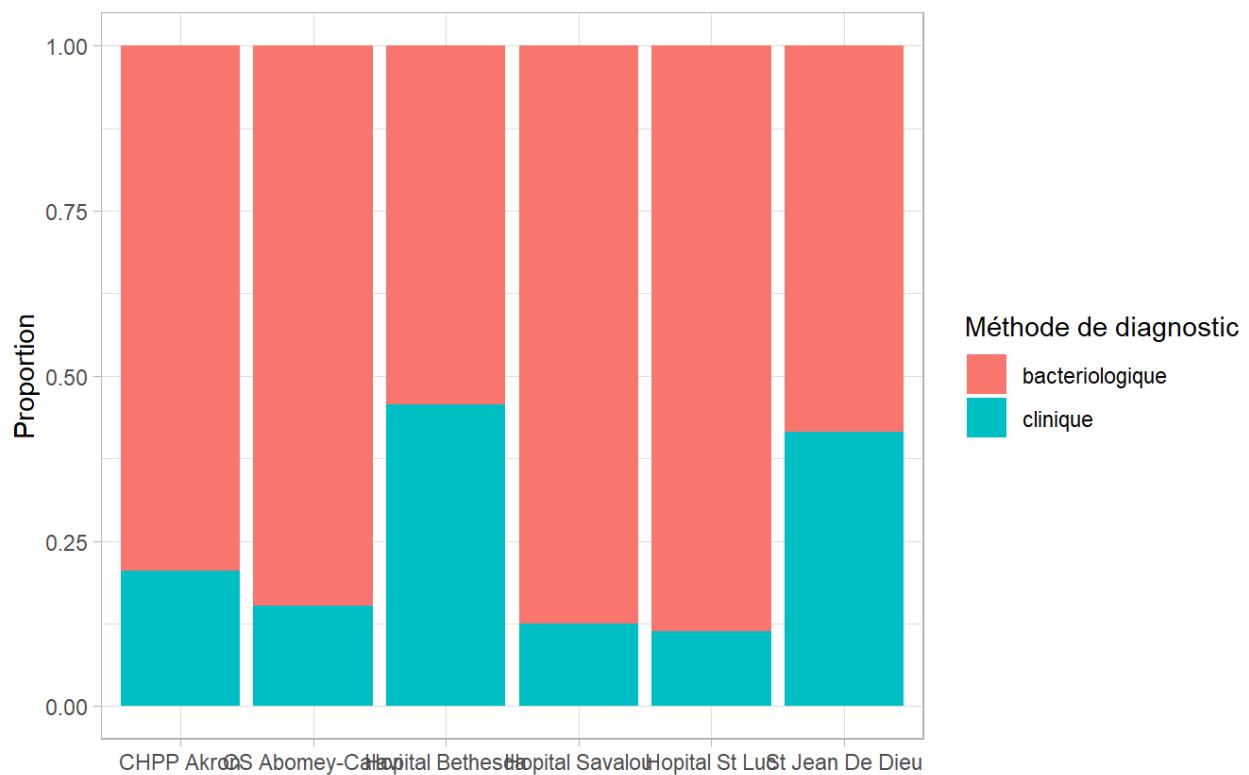
# Crédit à un diagramme à barres empilées normalisées avec les données
# résumées
hosp_dx_fill <- hopital_dx_prop %>%
  ggplot(aes(x = hopital, y = prop, fill = diagnosis_type)) +
  geom_col(position = position_fill()) +
  labs(title = "Diagnostic des nouveaux cas et des rechutes de tuberculose",
       subtitle = "Données de six établissements de santé au Bénin, 2015-
2017",
       x="", y = "Proportion", fill = "Méthode de diagnostic")

hosp_dx_fill

```

Diagnostic des nouveaux cas et des rechutes de tuberculose

Données de six établissements de santé au Bénin, 2015-2017



Dans ce graphique, `geom_col()` est utilisé pour créer un graphique à barres basé sur les variables `hopital` et `prop`.

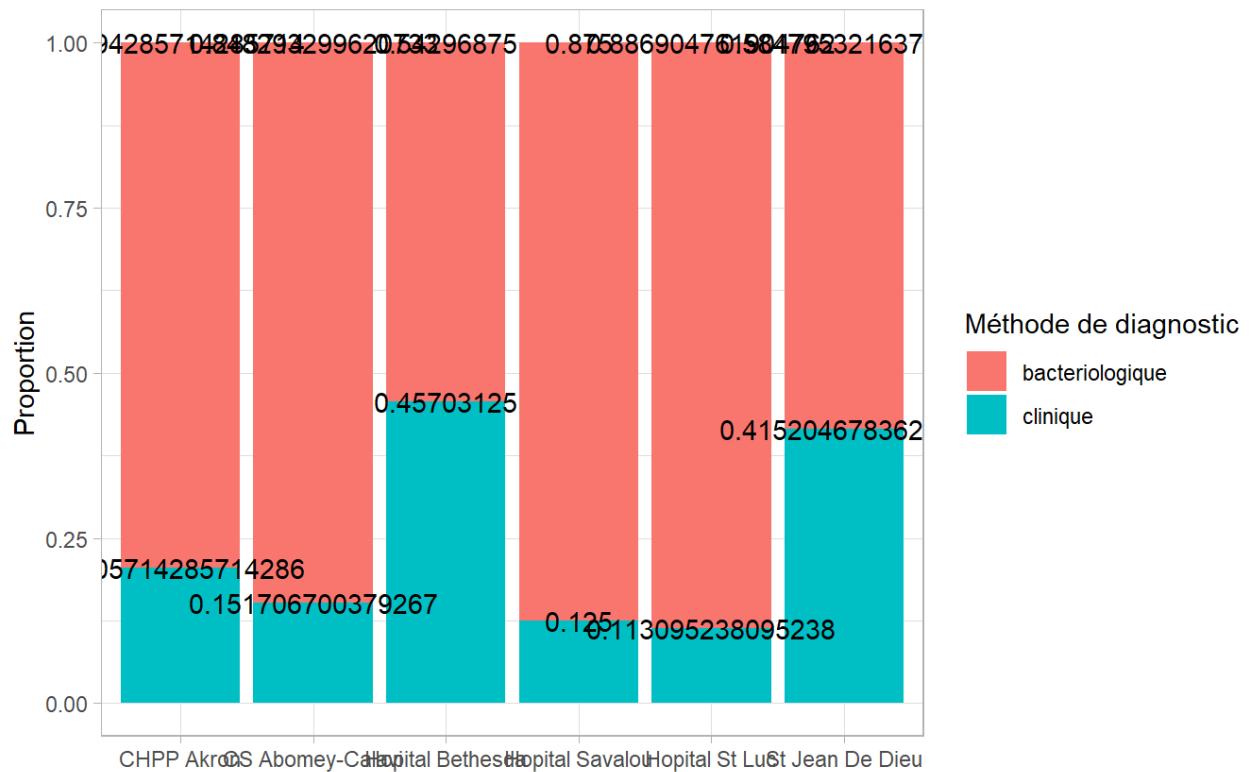
Étape 3 : Annoter le graphique avec `geom_text()` ou `geom_label()`

Maintenant, nous pouvons utiliser `geom_text()` en spécifiant la position de pour ajouter des étiquettes :

```
# Ajouter des étiquettes de texte au diagramme à barres empilées en pourcentage
hosp_dx_fill +
  geom_text(aes(label = prop),
            position=position_fill())
```

Diagnostic des nouveaux cas et des rechutes de tuberculose

Données de six établissements de santé au Bénin, 2015-2017



C'est un bon début mais évidemment nous avons encore un peu de travail à faire pour le rendre plus joli!

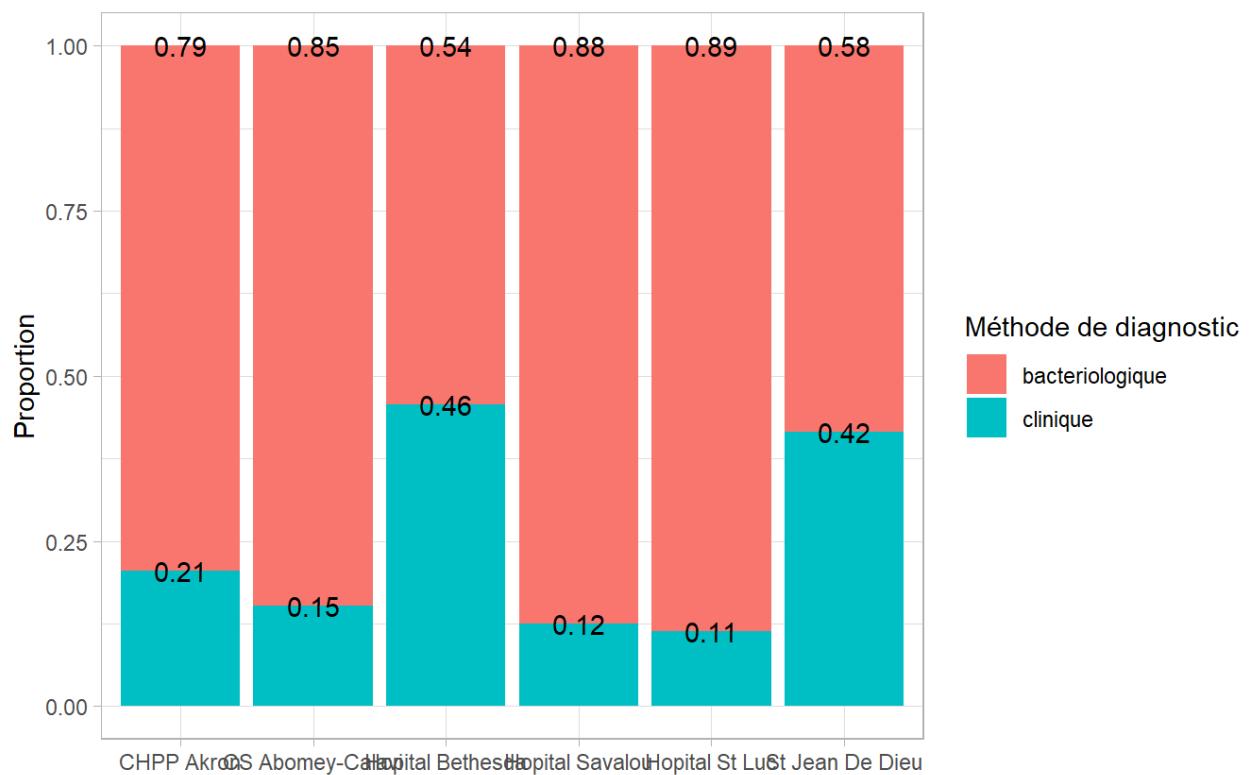
Étape 4 : Ajuster la position du texte pour l'aligner avec les barres

Avant d'ajuster nos étiquettes, nous allons nous occuper de ces décimales. Nous pourrions réduire le nombre de décimales comme ceci :

```
hosp_dx_fill +  
  geom_text(aes(label = round(prop, 2)), # arrondir le texte de l'étiquette à 2  
            chifffres sig  
            position = position_fill())
```

Diagnostic des nouveaux cas et des rechutes de tuberculose

Données de six établissements de santé au Bénin, 2015-2017

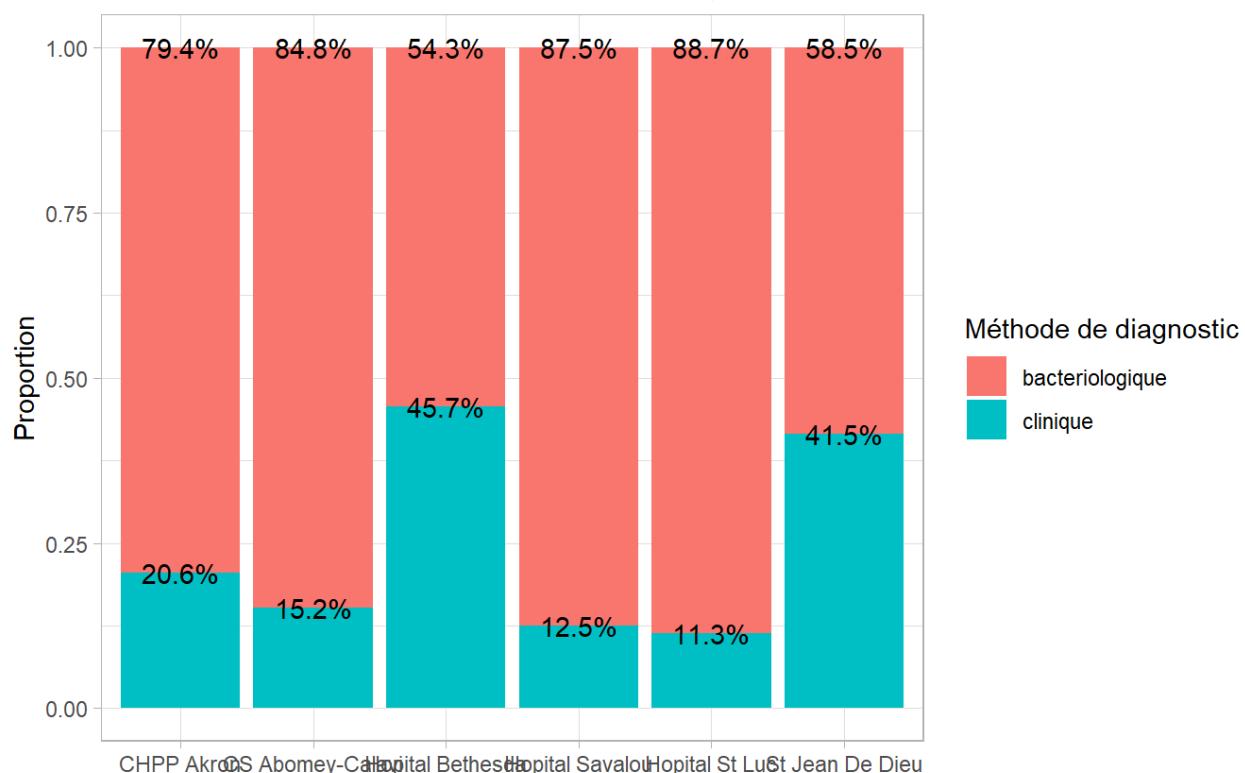


Cependant, la meilleure méthode est :

```
hosp_dx_fill +  
  geom_text(aes(label = scales::percent(prop)),  
            position = position_fill())
```

Diagnostic des nouveaux cas et des rechutes de tuberculose

Données de six établissements de santé au Bénin, 2015-2017



Le package `{scales}` est couramment utilisé avec `{ggplot2}` pour personnaliser les esthétiques, transformer les échelles d'axes, formater les étiquettes, définir des palettes de couleurs, etc.

SIDE NOTE



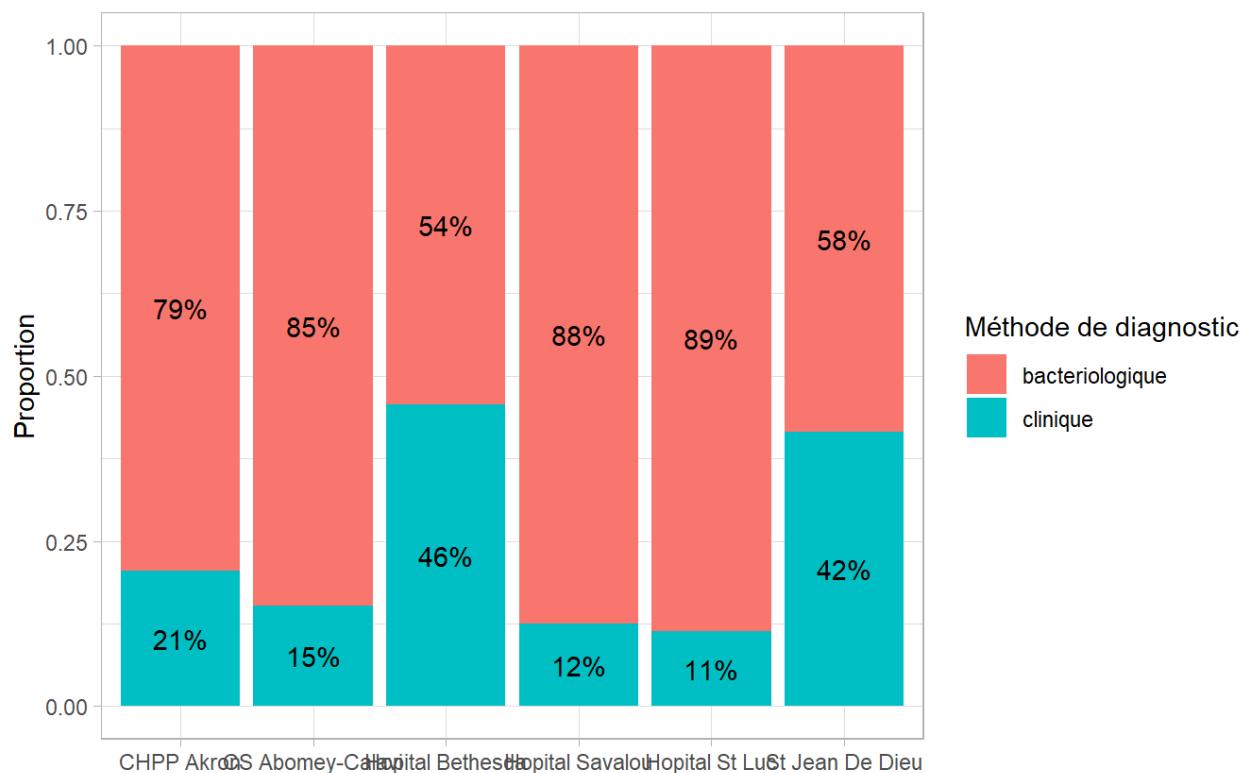
La fonction `scales::percent(prop)` que nous avons utilisée dans le code ci-dessus avec `geom_text()` convertit les proportions (valeurs de notre variable `prop`) en un format de pourcentage et ajoute des signes de pourcentage. Nous pouvons également contrôler le nombre de chiffres affichés à l'aide de l'argument `accuracy` (voir ci-dessous).

Ensuite, nous pouvons centrer les étiquettes dans chaque segment de barre en utilisant `vjust` dans la fonction `position_fill()`

```
# Déplacer le texte des étiquettes au milieu de chaque segment de barre
hosp_dx_fill +
  geom_text(aes(label = scales::percent(prop, accuracy = 1)),
            position = position_fill(vjust = 0.5)) # centrer les étiquettes
```

Diagnostic des nouveaux cas et des rechutes de tuberculose

Données de six établissements de santé au Bénin, 2015-2017

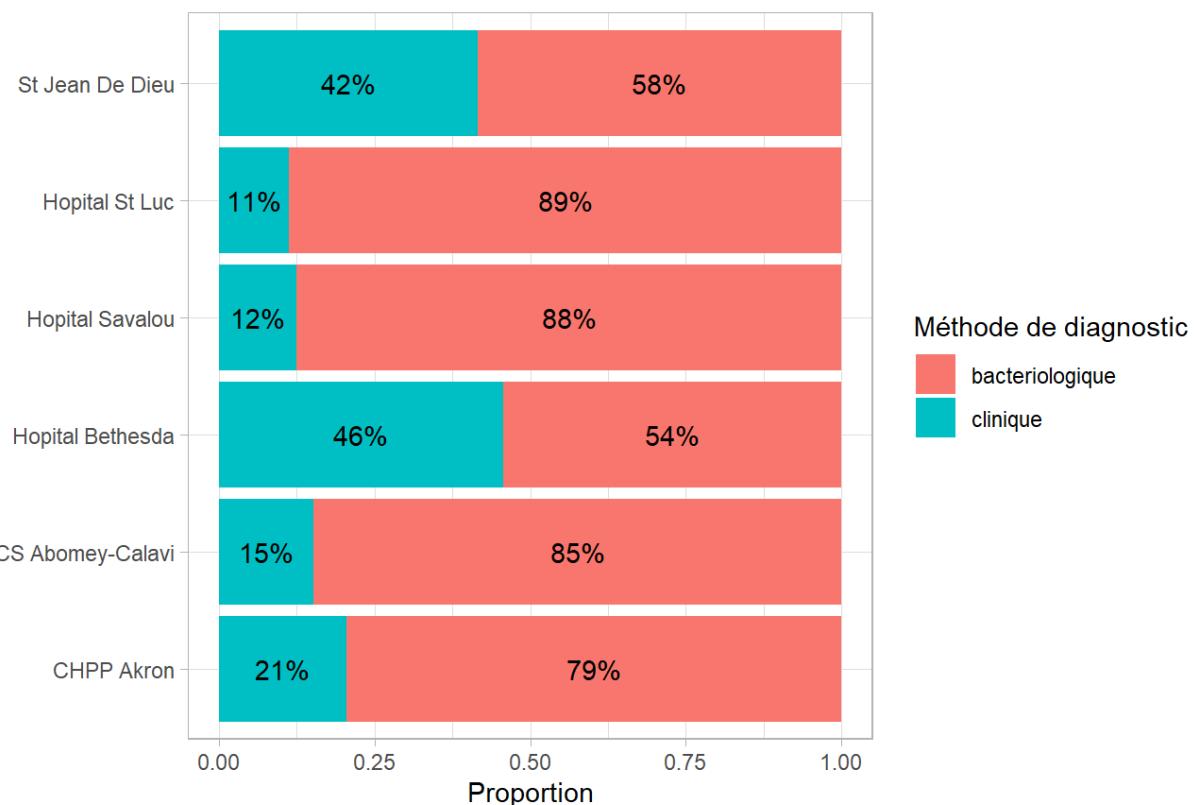


L'utilisation de coordonnées inversées dans les graphiques à barres peut grandement améliorer la lisibilité, en particulier lorsqu'on a affaire à de longs noms de catégories ou à de nombreuses catégories. Pour faire cela, nous allons utiliser `coord_flip` avec `geom_text()` comme nous l'avons fait pour les graphiques à barres empilées :

```
# Inverser les coordonnées pour une meilleure visualisation
hosp_dx_fill +
  geom_text(aes(label = scales::percent(prop, accuracy = 1)),
            position = position_fill(vjust = 0.5)) +
  coord_flip()
```

Diagnostic des nouveaux cas et des rechutes de tuberculose

Données de six établissements de santé au Bénin, 2015-2017

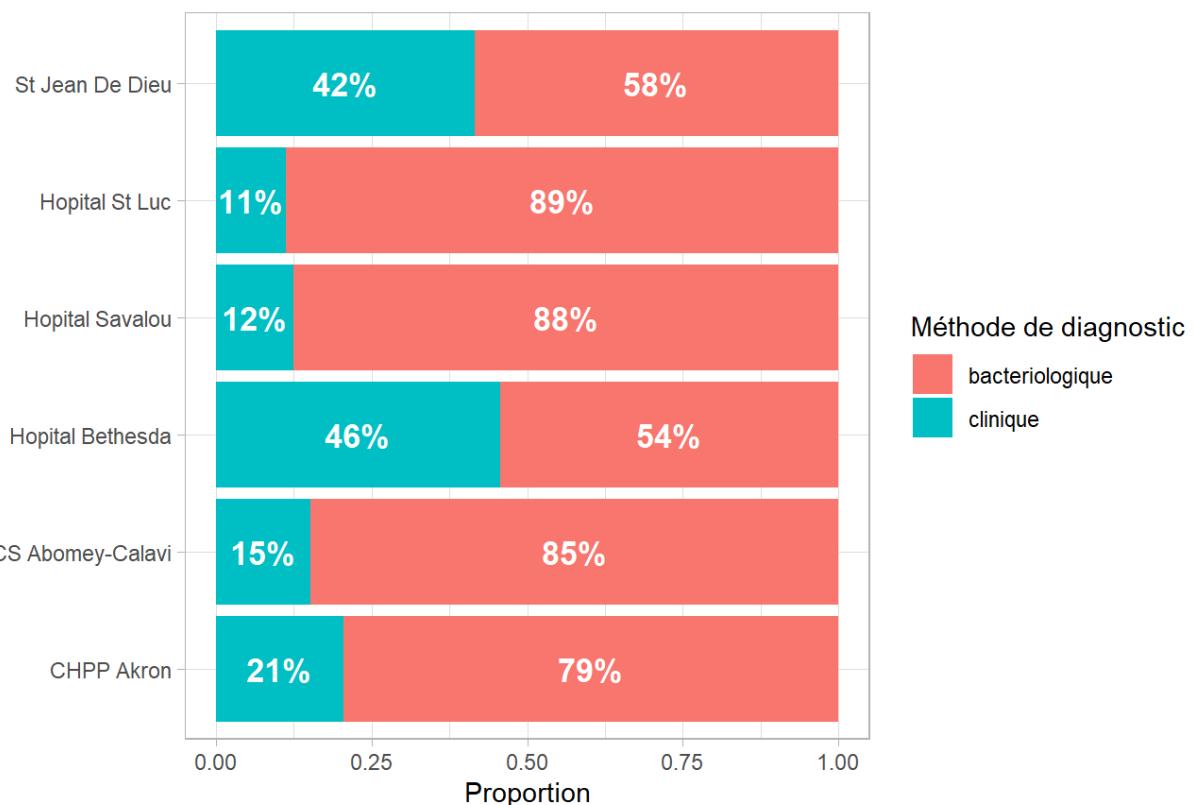


Super, maintenant nous pouvons ajouter quelques modifications esthétiques supplémentaires et nous obtiendrons le même graphique que nous avons vu au début de la section!

```
# Ajouter des modifications supplémentaires
hosp_dx_fill +
  geom_text(aes(label = scales::percent(prop, accuracy = 1)),
            position = position_fill(vjust = 0.5),
            color = "white", # Changer la couleur du texte
            fontface = "bold", # Mettre en gras
            size = 4.5) + # Changer la taille de la police
  coord_flip()
```

Diagnostic des nouveaux cas et des rechutes de tuberculose

Données de six établissements de santé au Bénin, 2015-2017



Étiquetage de graphiques circulaires

Étape 1 : Résumer les données

Commençons par résumer les données. Nous allons calculer le nombre total de cas pour chaque hôpital en regroupant les données par la variable `hopital`, puis en calculant la somme des cas dans chaque groupe.

```
# Nouveau tableau récapitulatif - les diagrammes circulaires ne peuvent
# visualiser qu'une seule variable catégorielle, donc un seul
# regroupement cette fois
resultats_totals <- resultats_tb %>%
  group_by(hopital) %>%
  summarise(
    cas_total = sum(cas))

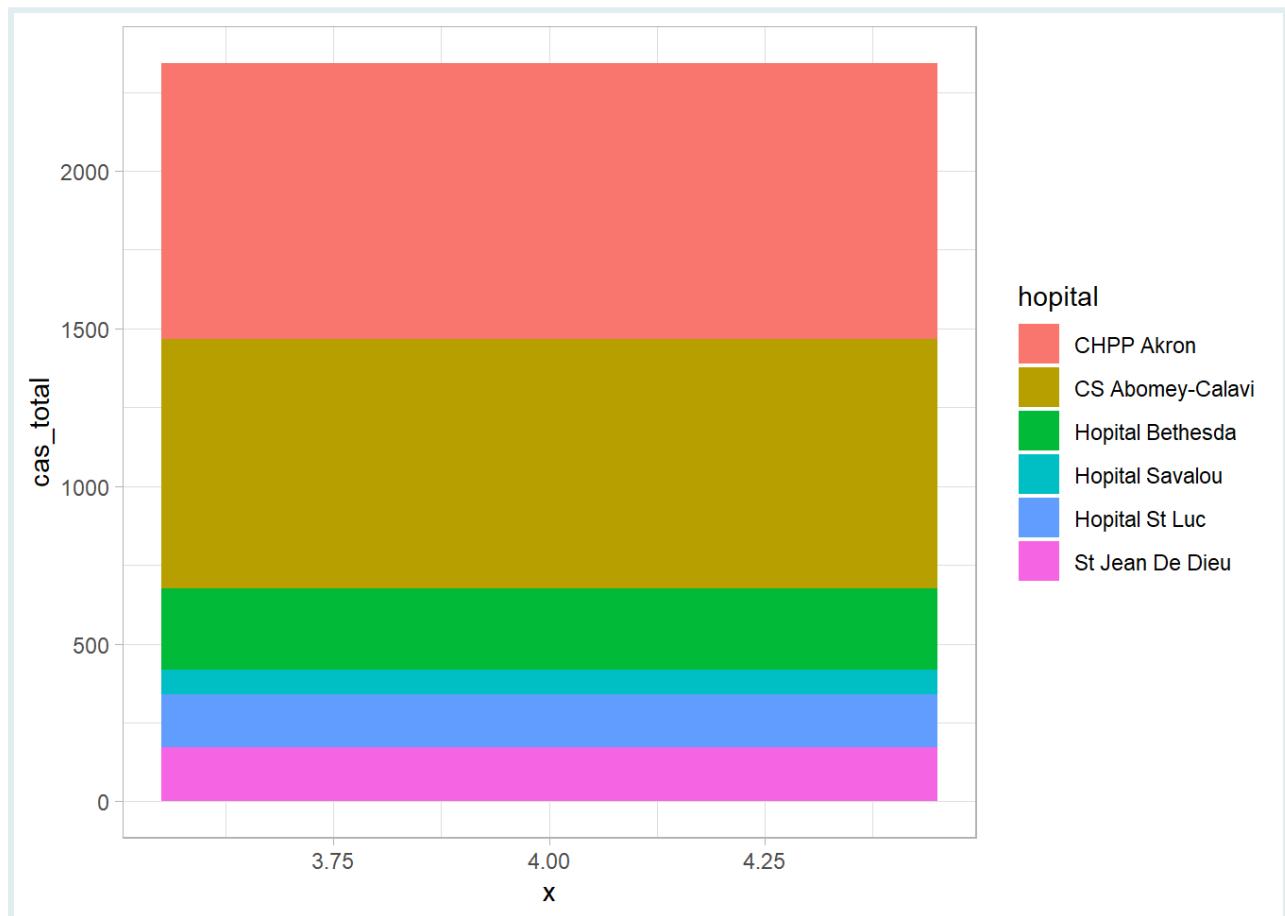
resultats_totals
```

Étape 2 : Créer les graphiques de base

Maintenant que nous avons notre nouvel ensemble de données, nous pouvons commencer par créer un diagramme à barres simple. Vous vous souvenez peut-être de la leçon précédente qu'un diagramme circulaire est fondamentalement une version ronde d'un diagramme à barres empilées.

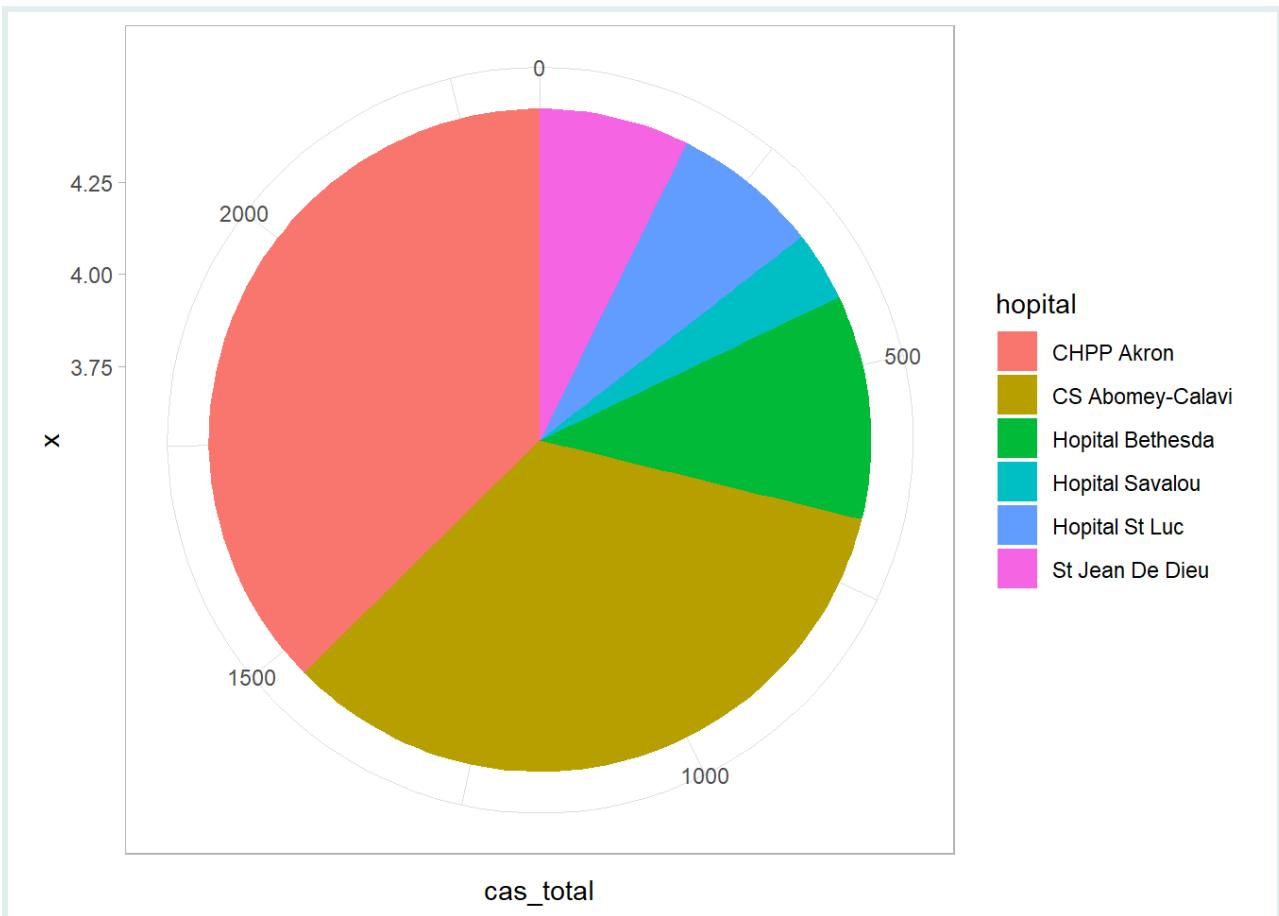
```
# Diagramme à barres simple (précurseur du diagramme circulaire)
resultats_stack <- ggplot(resultats_totals,
  aes(x=4, # Définir une valeur x arbitraire
      y=cas_total,
      fill=hopital)) +
  geom_col()

resultats_stack
```



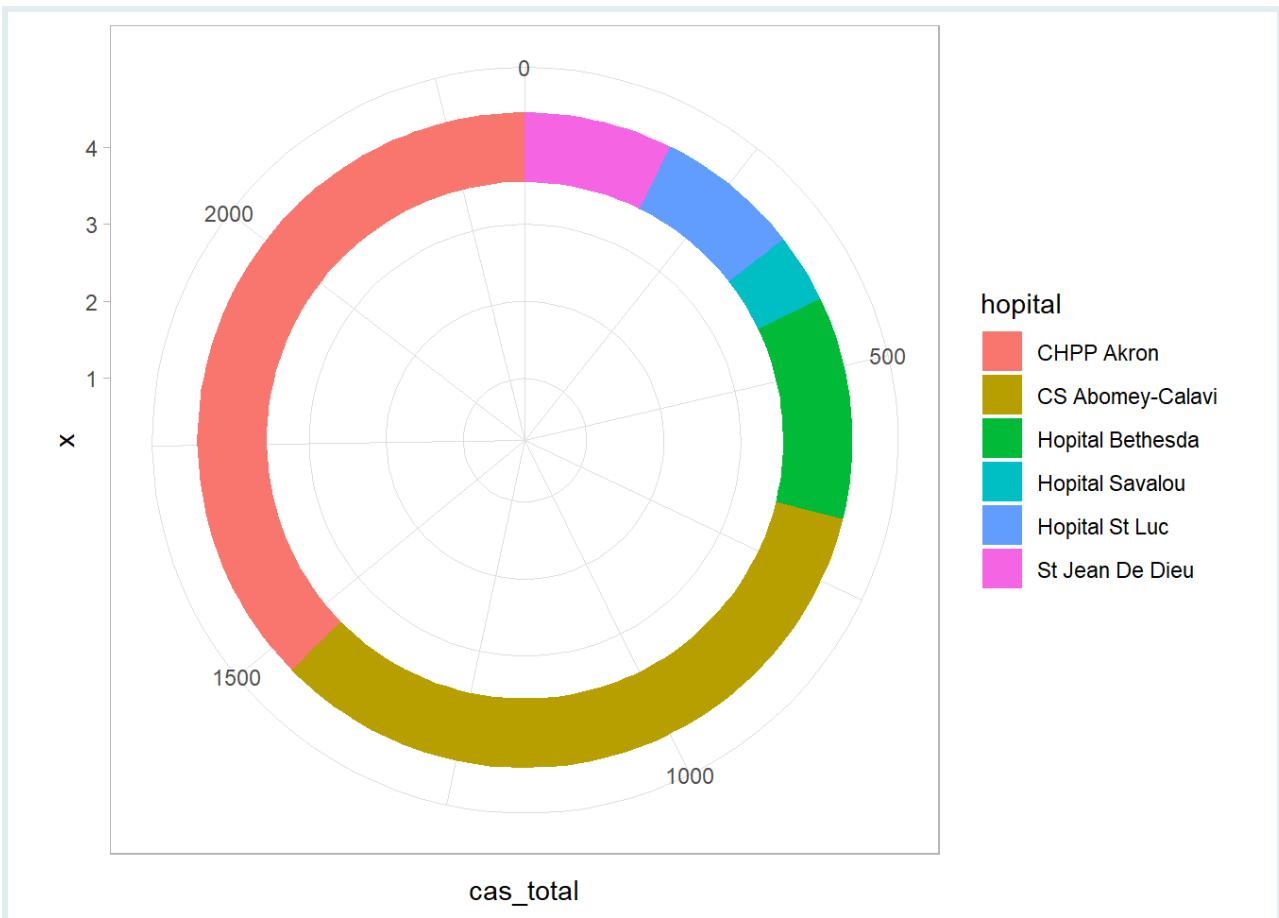
Maintenant, nous pouvons créer notre diagramme circulaire de base. Comme nous l'avons appris dans la dernière leçon, pour transformer des coordonnées linéaires en coordonnées polaires, nous utilisons la fonction `coord_polar()`. Le paramètre `theta` définit quelle variable esthétique doit être mappée sur la coordonnée angulaire dans le système de coordonnées polaires. En spécifiant "`y`", nous utilisons la hauteur des barres pour déterminer l'angle de chaque tranche de notre diagramme circulaire.

```
resultats_circulaire <- resultats_stack +  
  coord_polar(theta = "y")  
  
resultats_circulaire
```



Ensuite, nous allons créer un diagramme circulaire à anneau vide de base en utilisant `xlim()`.

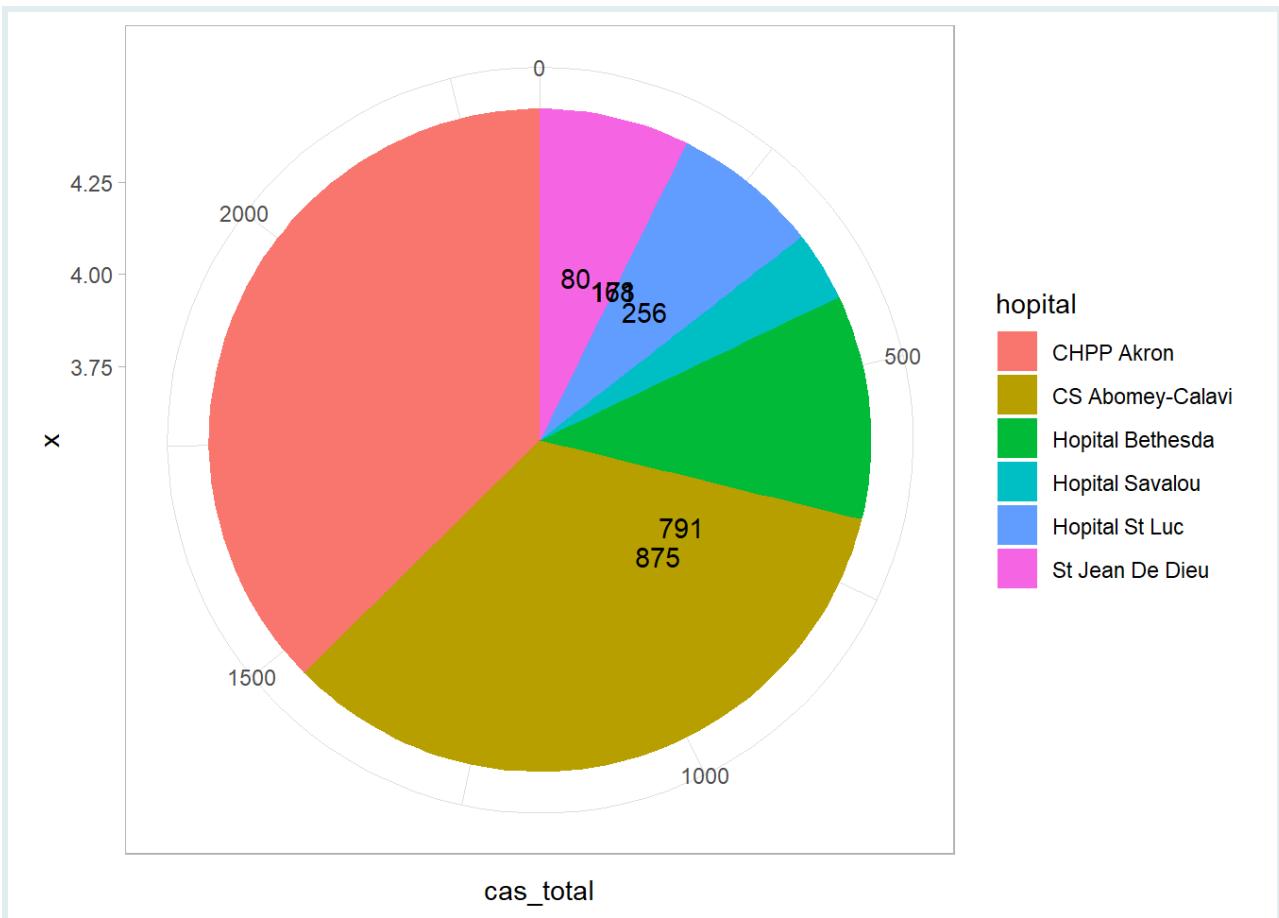
```
resultats_anneau <- resultats_circulaire +  
  xlim(c(0.2, 4.5))  
  
resultats_anneau
```



Étape 3 : Annoter le graphique avec `geom_text()` ou `geom_label()`

Maintenant, ajoutons des étiquettes à notre graphique circulaire avec `geom_text()`.

```
# Ajoutez geom_text comme vous le feriez habituellement pour un diagramme à
# barres empilées
resultats_circulaire +
  geom_text(aes(label = cas_total))
```

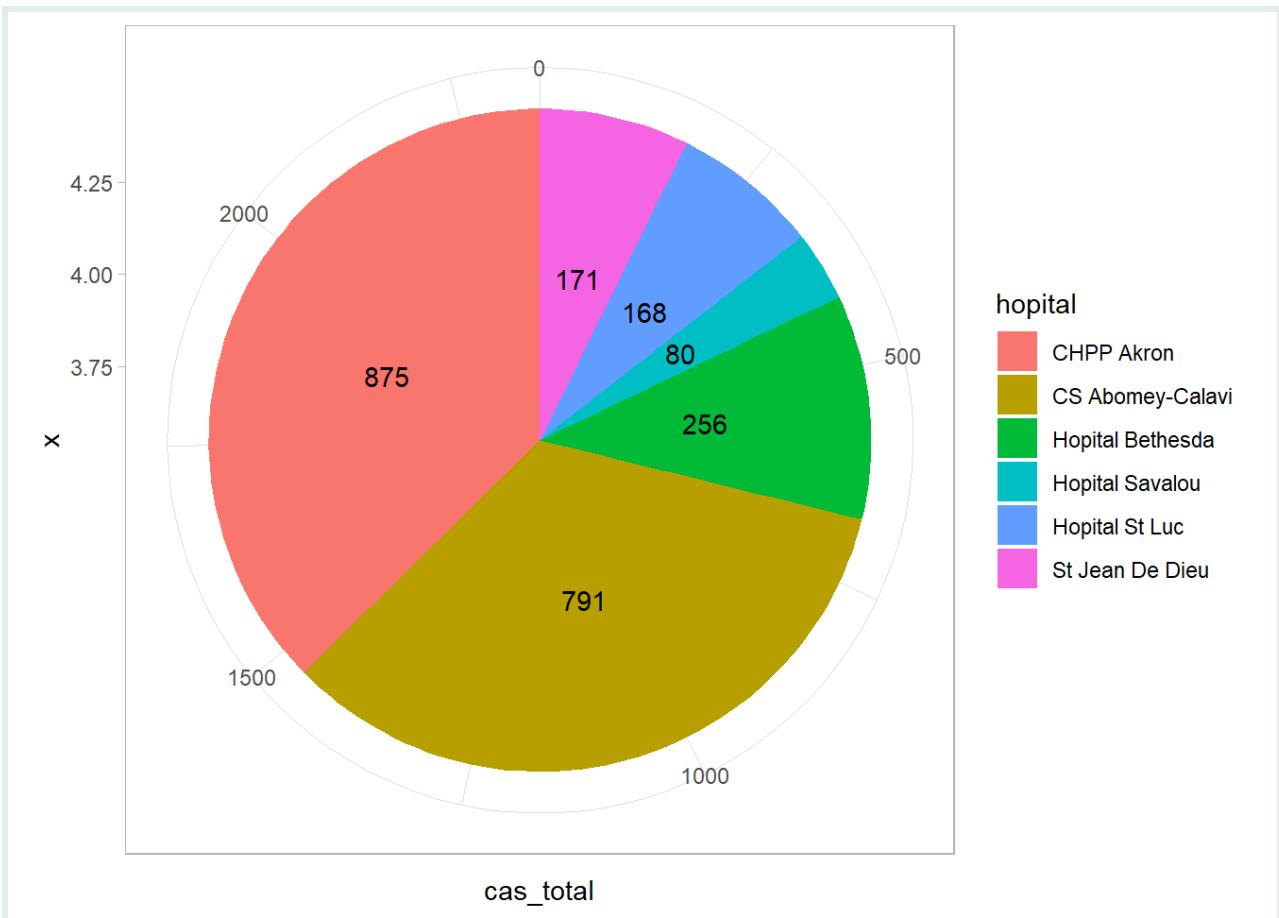


Vous remarquerez que nos étiquettes restent au milieu des tranches parce que `coord_polar()` est appliqué à la fois à `geom_col()` et `geom_text()`. Les nombres apparaissent dans les mauvais segments car nous n'avons pas encore ajouté de réglage `position` à la géométrie d'étiquetage.

Étape 4 : Ajuster la position du texte pour l'aligner avec les tranches du cercle et sections de l'anneau

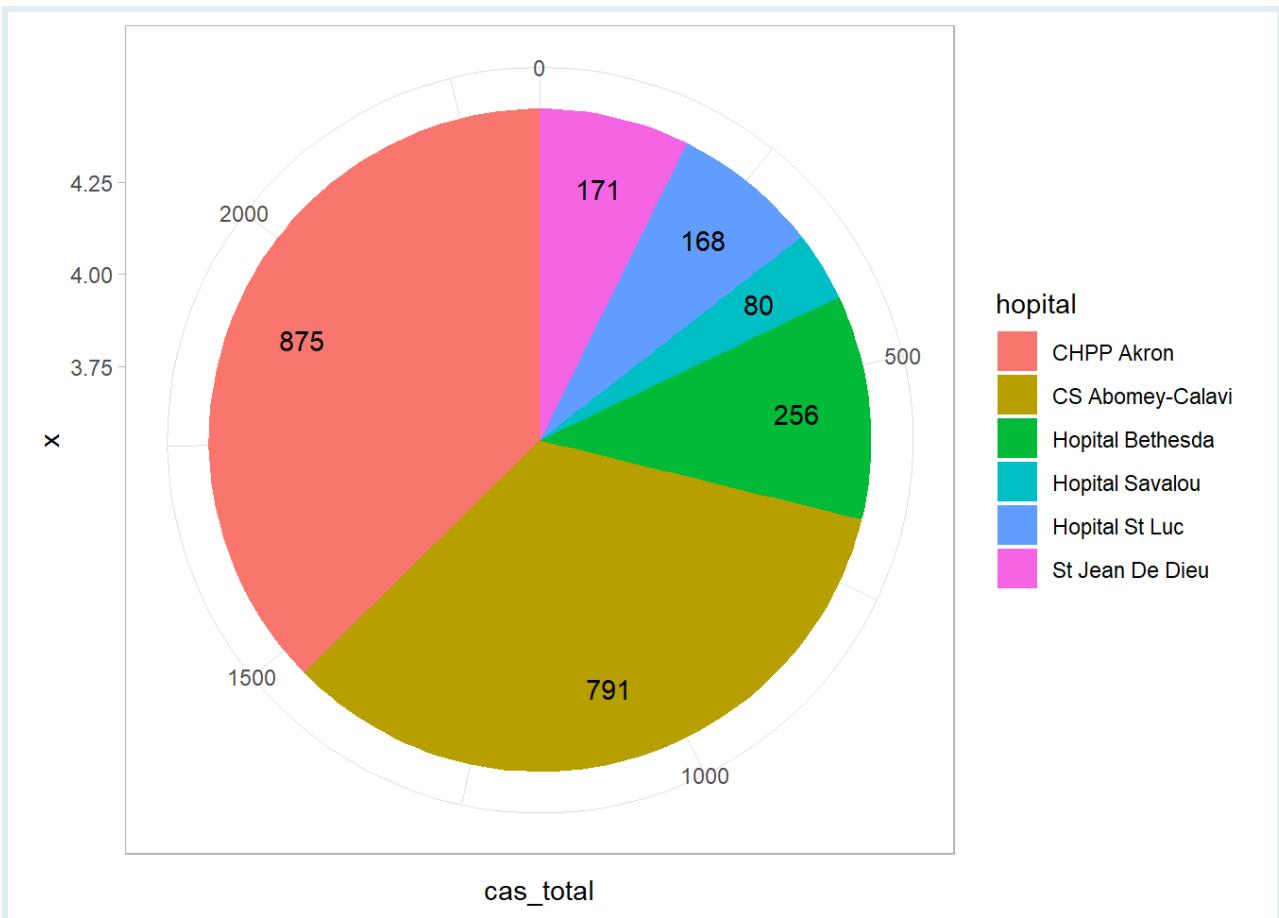
Maintenant, tout comme nous l'avons fait précédemment, nous utiliserons l'argument `position_stack()` pour centrer les étiquettes.

```
resultats_circulaire +
  geom_text(aes(label = cas_total),
            position = position_stack(vjust = 0.5)) # Centrer les étiquettes
```



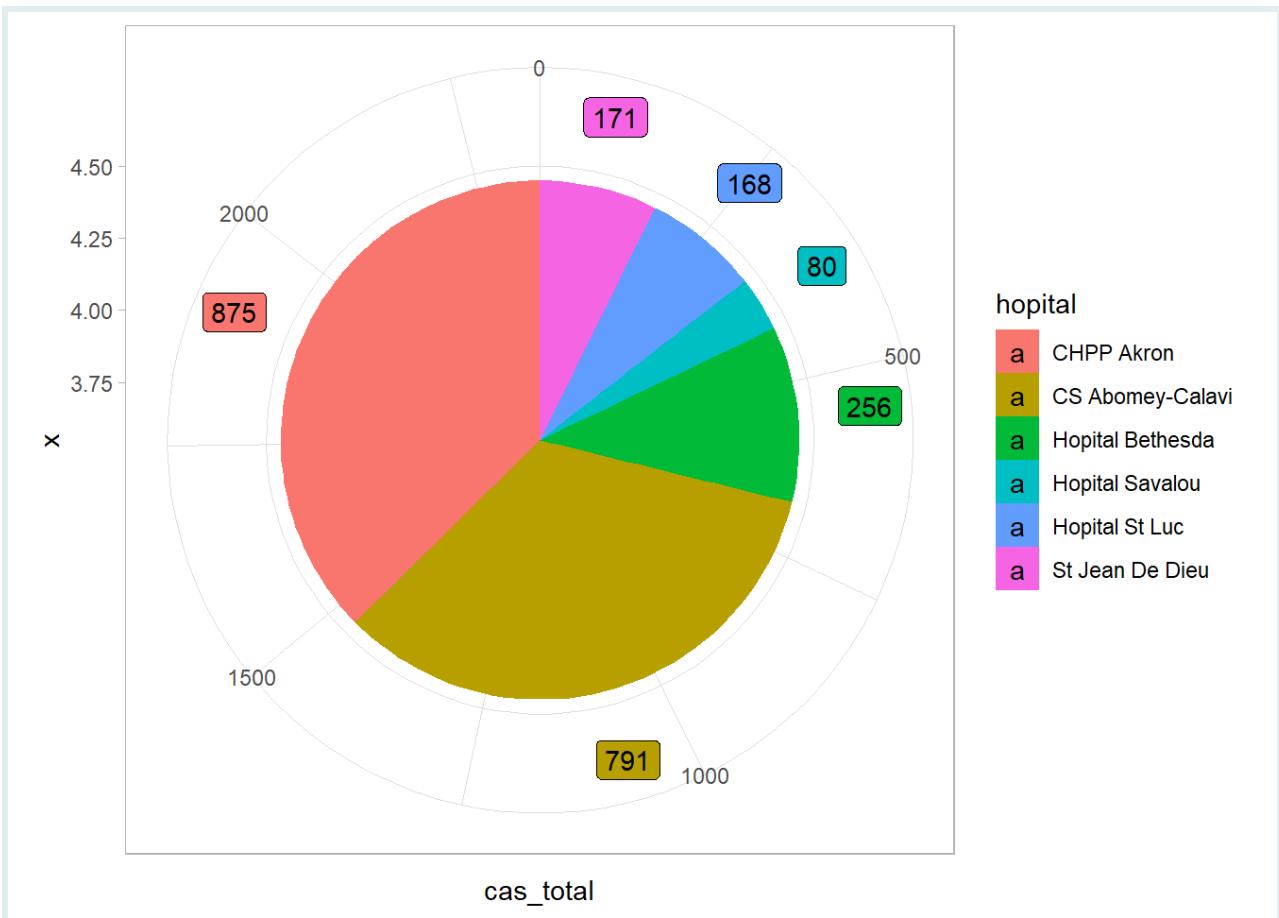
Pour déplacer les étiquettes le long de l'axe x de notre diagramme circulaire (vers le haut et vers le bas du rayon), nous pouvons spécifier une valeur fixe à l'esthétique x dans `geom_text()`

```
resultats_circulaire +
  geom_text(aes(label = cas_total,
                 x = 4.25), # déplacer le texte loin du centre
            position = position_stack(vjust = 0.5))
```



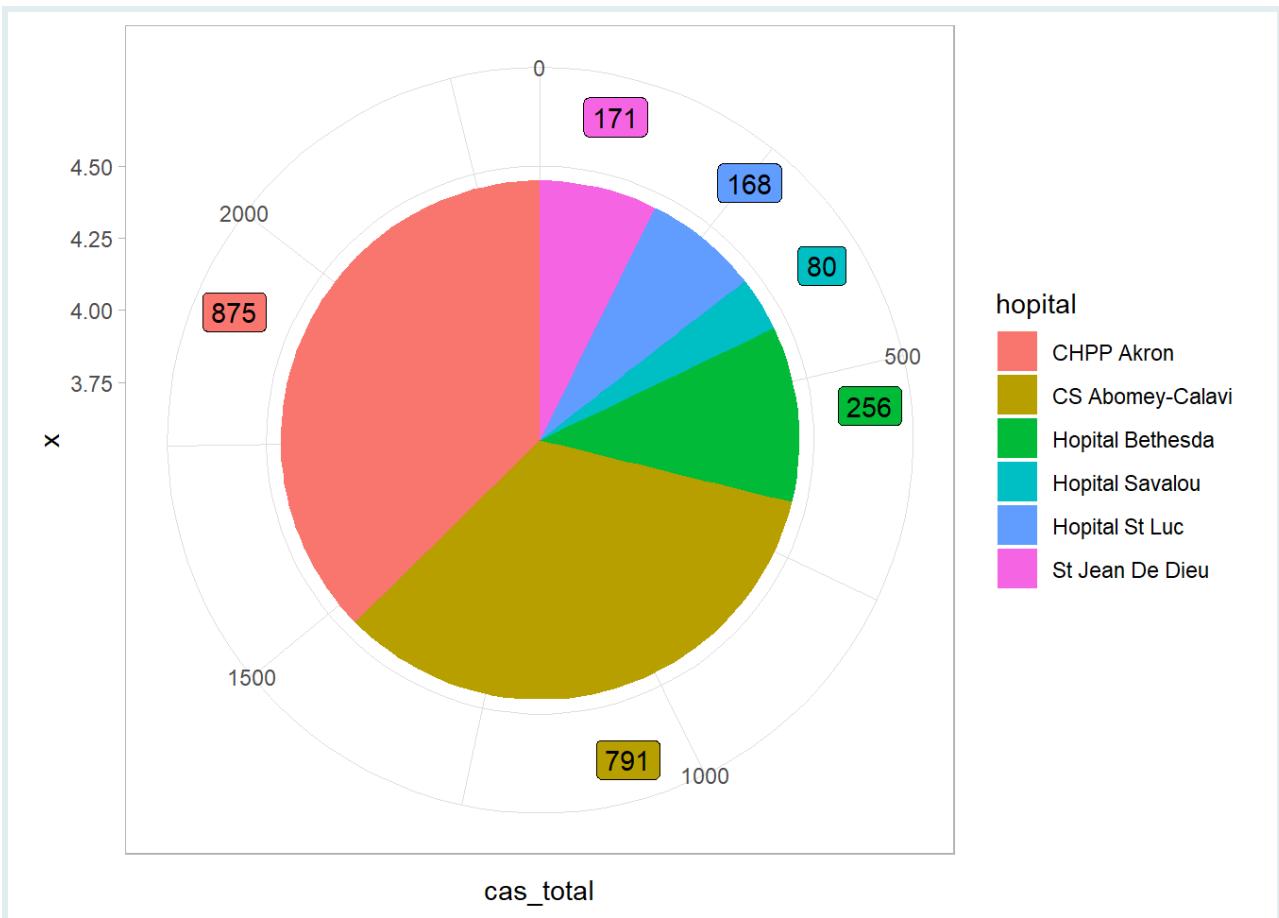
Nous pouvons faire la même chose avec `geom_label()`.

```
# Ajustement similaire avec geom_label()
resultats_circulaire +
  geom_label(aes(label = cas_total,
                 x = 4.7), # déplacer le texte loin du centre
             position = position_stack(vjust = 0.5))
```



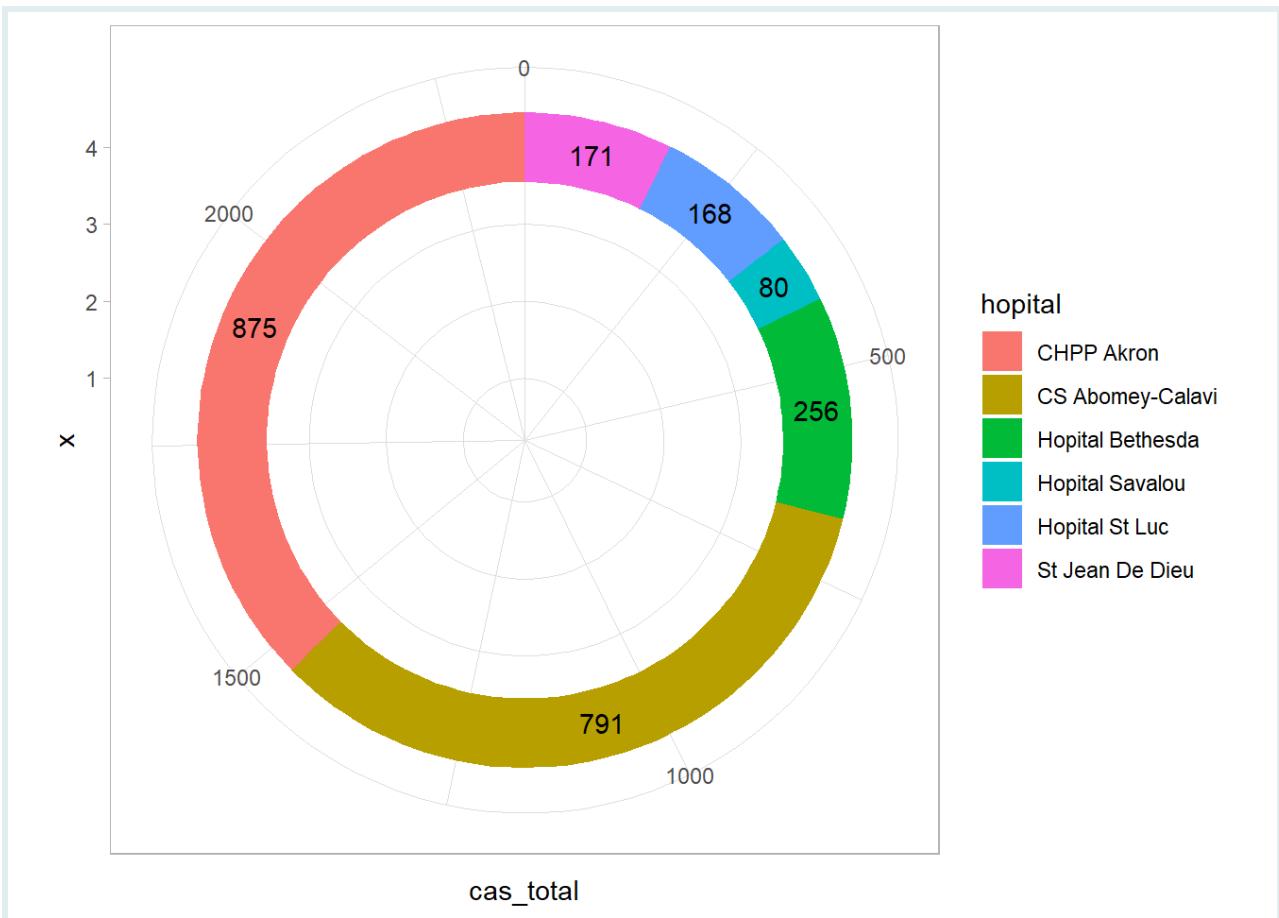
Remarquez qu'une fois que nous avons utilisé `geom_label()`, la lettre `a` est apparue sur les légendes. C'est gênant et ruinerait tout le travail ardu que nous avons fait pour rendre ces graphiques présentables. Pour atténuer ce problème, nous ajoutons l'argument `show.legend = FALSE` à la fonction `geom_label()` comme ceci :

```
# Ajustement similaire avec geom_label()
resultats_circulaire +
  geom_label(aes(label = cas_total,
                 x = 4.7), # déplacer le texte loin du centre
             position = position_stack(vjust = 0.5),
             show.legend=FALSE)
```



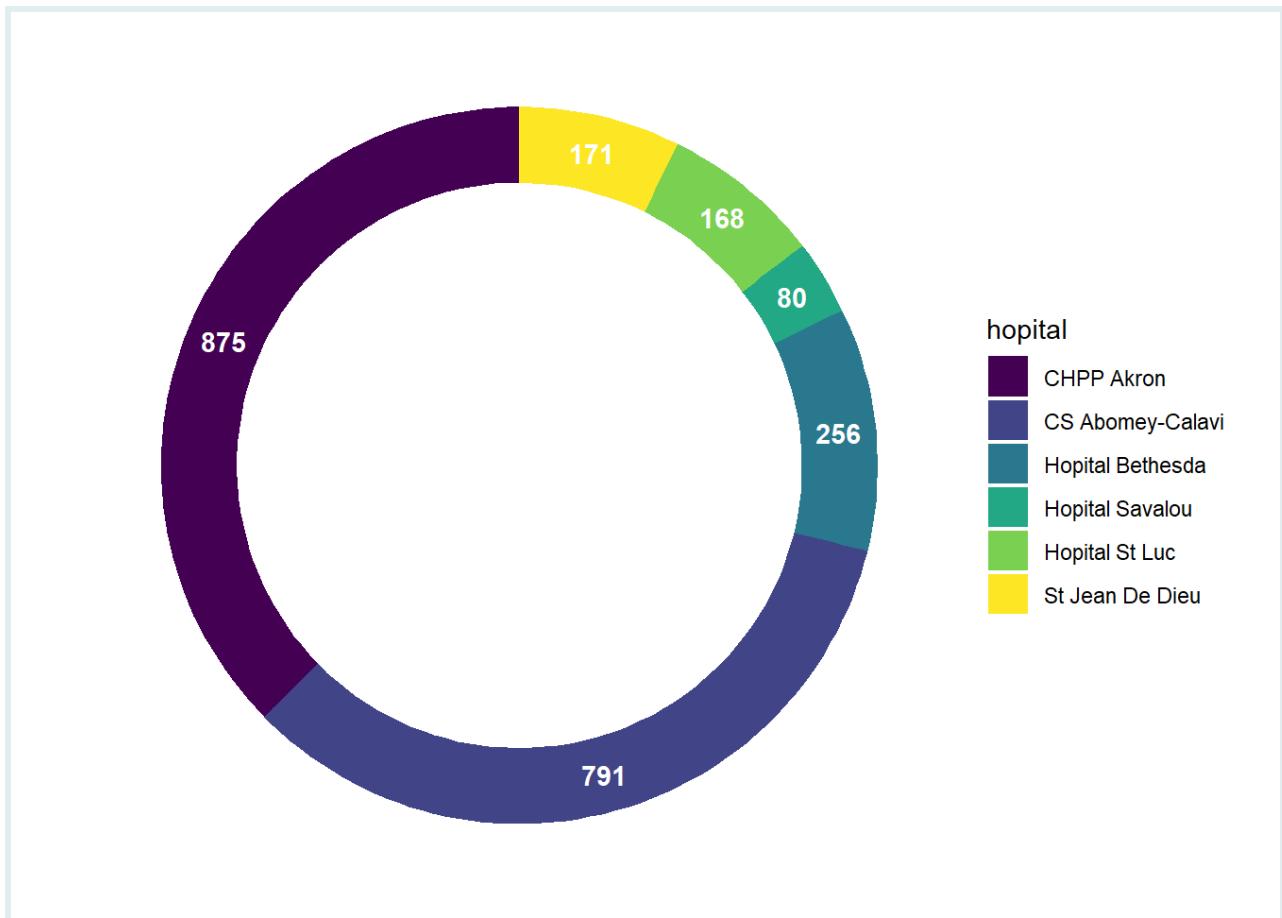
Ensuite, passons à notre diagramme à anneau de base. Nous allons l'étiquetter en utilisant `geom_text()` et en spécifiant directement la position et centrant nos étiquettes au milieu de chaque section du graphique, tout comme nous l'avons fait pour notre diagramme circulaire.

```
# ajouter du texte - lims appliqué aux colonnes et au texte
resultats_anneau +
  geom_text(aes(label = cas_total),
            position = position_stack(vjust = 0.5))
```



Pour finir, nous pouvons apporter quelques modifications supplémentaires. Ici, nous améliorons l'esthétique du graphique en appliquant `theme_void()` pour supprimer les éléments d'arrière-plan encombrants, en introduisant une nouvelle palette de couleurs avec `scale_fill_viridis_d()`, et en ajustant les étiquettes de texte à l'aide de `geom_text()` avec du texte blanc et en gras pour une meilleure visibilité et un meilleur contraste.

```
# Modifications esthétiques supplémentaires
resultats_anneau +
  geom_text(aes(label = cas_total),
            position = position_stack(vjust = 0.5),
            color = "white",
            fontface = "bold") +
  theme_void() +
  scale_fill_viridis_d()
```



Conclusion !

Dans cette leçon, nous nous sommes plongés dans l'amélioration des graphiques avec des étiquettes, en nous concentrant sur `geom_label()` et `geom_text()`. En utilisant des packages comme `{tidyverse}`, `{ggtext}` et `{glue}`, nous explorons une gamme de techniques d'étiquetage allant du texte simple au formatage avancé.

Nous avons commencé avec `geom_text()`, démontrant comment placer du texte lisible directement sur les graphiques en utilisant l'ensemble de données `resultats_tb`. Ensuite, `geom_label()` est introduit pour créer des étiquettes plus proéminentes avec des encadrés de fond, idéales pour les arrière-plans de graphiques complexes.

Ceci a été suivi par une discussion sur l'utilisation de coordonnées inversées dans les graphiques à barres pour une meilleure lisibilité et visibilité des étiquettes, ainsi que des conseils sur l'ajustement des limites d'axe avec `expand_limits()`.

Cette leçon est un guide complet sur l'utilisation efficace de l'étiquetage dans `{ggplot2}`, améliorant la clarté et l'attrait visuel des visualisations de données.

Références

Certains éléments de cette leçon ont été adaptés à partir des sources suivantes :

- Horst, Allison. “Allisonhorst/Dplyr-Learnr: A Colorful Introduction to Some Common Functions in Dplyr, Part of the Tidyverse.” GitHub. Accessed April 6, 2022. <https://github.com/allisonhorst/dplyr-learnr>.

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.



Création de cartes choroplèthes avec {ggplot2}

Création de cartes choroplèthes avec {ggplot2}
Introduction
Charger les packages nécessaires
Préparation des données
Création d'une Carte Choroplète avec {ggplot2}
Mise à l'échelle des Couleurs
Facet Wrap vs. Grille
En Résumé !
Solutions
Références

Création de cartes choroplèthes avec {ggplot2}

Introduction

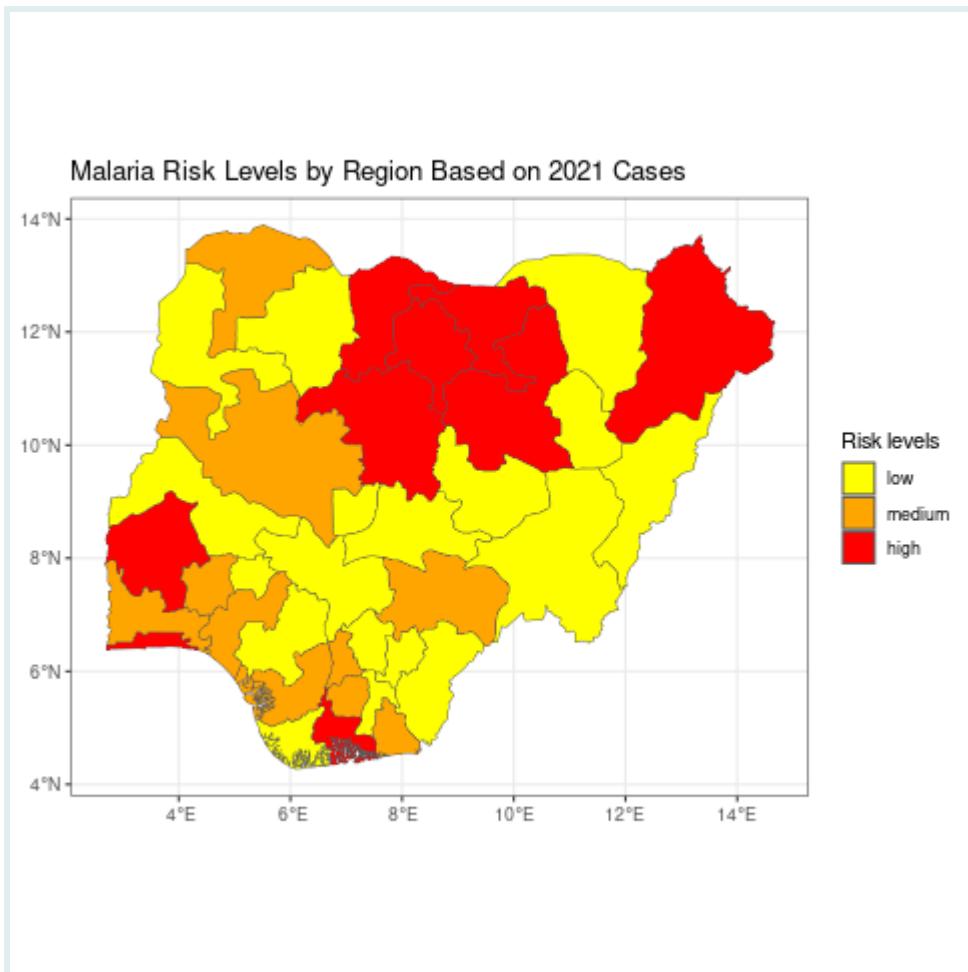
Une carte choroplète est une carte thématique dans laquelle les régions géographiques sont ombrées ou modélisées proportionnellement à la valeur d'une variable représentée. Cette variable peut être un indicateur épidémiologique tel que la prévalence ou le taux de mortalité d'une maladie. Les cartes choroplèthes sont particulièrement utiles pour visualiser les motifs spatiaux et les variations entre différentes régions.

Les composants essentiels d'une carte choroplète incluent :

Régions géographiques : Ce sont les zones qui seront représentées sur la carte, telles que les pays, les états, les districts ou d'autres divisions géographiques.

Valeurs des données : Ce sont les valeurs associées à chaque région géographique qui seront représentées sur la carte, y compris la densité de population, la prévalence, l'incidence, le taux de mortalité, etc.

Échelle de couleurs : C'est la gamme de couleurs utilisée pour représenter les différentes valeurs des données. Typiquement, un gradient de couleurs est utilisé, avec des couleurs plus claires représentant des valeurs inférieures et des couleurs plus foncées représentant des valeurs supérieures.



Les cartes choroplèthes offrent :

- Des visuels clairs mettant en évidence les tendances des données spatiales.
- Des conceptions intuitives compréhensibles sans expertise.

SIDE NOTE



Cependant, elles présentent des défis :

- La classification des données et les choix de couleurs peuvent biaiser les apparences.
- Les régions plus grandes pourraient dominer, causant un biais visuel.

Dans la section suivante, vous apprendrez à créer une carte choroplète en utilisant le package `{ggplot2}` dans R.

Objectifs d'apprentissage

1. Cartes choroplèthes avec {ggplot2} :

- Maîtriser les fonctions `ggplot()` et `geom_sf()` pour la visualisation des cartes.

2. Appariement des données avec des polygones :

- Obtenir des limites et des données liées aux maladies.
- Combiner les données en fonction des niveaux administratifs.

3. Techniques d'échelle de couleurs :

- Implémenter des échelles de couleurs pour les types de données continues et discrètes.

4. Facettes pour la visualisation des cartes :

- Utiliser `facet_wrap()` et `facet_grid()` pour créer des petites cartes multiples.

Charger les packages nécessaires

Ce morceau de code ci-dessous montre comment charger les paquets nécessaires en utilisant `p_load()` du package `{pacman}`. S'ils ne sont pas installés, il installe le paquet avant de le charger.

```
# Charger les paquets nécessaires

# Utiliser pacman pour charger plusieurs paquets ; il les installera s'ils ne
# sont pas déjà installés
pacman::p_load(tidyverse,    # pour la manipulation et la visualisation des
               données
               here,          # pour les chemins de fichiers relatifs au projet
               sf)            # pour les données spatiales

# Désactiver la notation scientifique pour des affichages numériques plus
# clairs
options(scipen=10000)
```

Préparation des données

Avant de créer une carte choroplète, il est essentiel de préparer les données. Les données doivent contenir les régions géographiques et les valeurs que vous souhaitez visualiser.

Dans cette section, vous passerez en revue le processus de préparation des données, qui comprend les étapes suivantes :

- 1. Importer les données de polygones** : Il s'agit des données géographiques contenant les limites de chaque région que vous souhaitez inclure dans votre carte.
- 2. Importer les données attributaires** : Il s'agit des données contenant les valeurs que vous souhaitez visualiser sur la carte, telles que la prévalence de la maladie, la densité de population, etc.
- 3. Joindre les données de polygones et les données attributaires** : Cette étape implique de fusionner les données de polygones avec les données attributaires en fonction d'un identifiant commun, tel que le niveau administratif ou le nom de la région. Cela créera un ensemble de données unique contenant à la fois les limites géographiques et les valeurs de données correspondantes.

Maintenant, passons en détail à chaque étape !

Étape 1 : Importer les données de polygones

Les polygones sont des formes fermées avec trois côtés ou plus. Dans les données spatiales, les polygones sont utilisés pour représenter des zones telles que la limite d'une ville, d'un lac ou de tout type d'utilisation des terres. Ils sont essentiels dans les systèmes d'information géographique (SIG) pour des tâches telles que la cartographie, l'analyse spatiale et la classification de l'occupation des terres.

SIDE NOTE



Les fichiers de formes (shapefiles) sont un format courant pour stocker des données spatiales. Ils se composent d'au moins trois fichiers avec des extensions .shp (forme), .shx (index) et .dbf (données attributaires).

Dans R, vous pouvez charger des fichiers de formes en utilisant le package `sf`. Dans notre leçon d'aujourd'hui, nous travaillerons avec des données sur le paludisme provenant de la revue Epi du Nigeria (2022).

```
# Lecture du fichier de formes
nga_adm1 <-
  sf::st_read(here::here("data/raw/NGA_adm_shapefile/NGA_adm1.shp"))
```

```
## Reading layer `NGA_adm1` from data source
##
`C:\GitHub\graph_courses\epi_reports_staging\EPIREP_FR_choropleth_maps\data\raw\
```

```
NGA_adm_shapefile\NGA_adm1.shp'  
##   using driver `ESRI Shapefile'  
## Simple feature collection with 38 features and 9 fields  
## Geometry type: MULTIPOLYGON  
## Dimension:      XY  
## Bounding box:  xmin: 2.668431 ymin: 4.270418 xmax: 14.67642 ymax: 13.89201  
## Geodetic CRS:  WGS 84
```

REMINDER



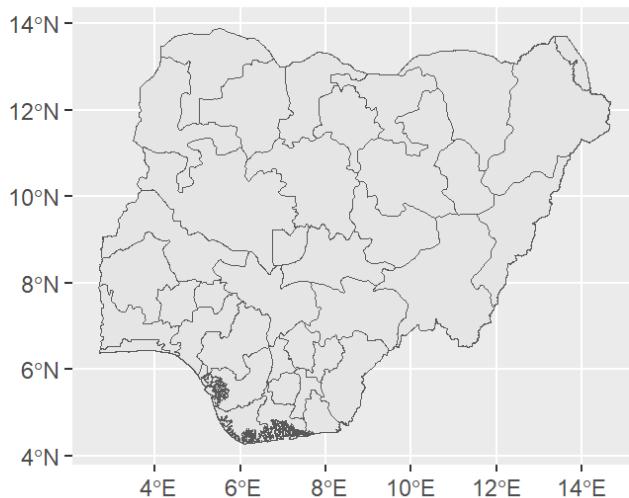
Les éléments importants de toute couche ggplot sont les mappings esthétiques `aes(x, y, ...)` qui indiquent à ggplot où placer les objets graphiques.

Nous pouvons imaginer une carte comme un graphique avec toutes les fonctionnalités mappées sur un axe x et y. Tous les types de géométrie (`geom_`) dans ggplot comportent une sorte de mapping esthétique, et ceux-ci peuvent être déclarés au niveau du graphique, par exemple, `ggplot(data, aes(x = variable1, y = variable2))`, ou au niveau de chaque couche individuelle, par exemple, `geom_point(aes(color = variable3))`.

Ci-dessous, vous pouvez voir que `geom_sf()` est utilisé pour tracer les limites des différents états du Nigeria. De même, différentes couches peuvent être ajoutées par-dessus.

```
ggplot() +  
  geom_sf(data = nga_adm1) +  
  labs(title = "Aperçu du fichier de formes du Nigeria")
```

Aperçu du fichier de formes du Nigeria



Ici, nous lisons d'abord le fichier de formes du Nigeria en utilisant `sf::st_read()` puis nous le traçons en utilisant `ggplot`. La fonction `geom_sf()` est utilisée pour rendre les données spatiales, et `labs()` est utilisé pour ajouter un titre au graphique.



Comme le fichier de formes a été chargé en utilisant `sf::st_read()`, nous n'avons pas besoin de spécifier les noms des axes. En fait, les coordonnées sont stockées comme un objet multipolygone dans la variable géométrie, et `geom_sf()` les reconnaît automatiquement.

SIDE NOTE



- Le {rgeoboundaries} (<https://github.com/wmgeolab/rgeoboundaries>) récupère des données de Geoboundaries, une base de données ouverte de limites administratives politiques.
- Le {rnatural-earth} (<https://github.com/ropensci/rnaturalearth>) récupère des données de Natural Earth, un ensemble de données cartographiques du domaine public.

Étape 2 : Importer les Données d'Attribut



Dans le contexte des cartes choroplèthes, les “données d’attribut” font référence aux informations quantitatives ou qualitatives qui seront utilisées pour colorer ou nuancer les différentes zones géographiques sur la carte. Par exemple, si vous avez une carte des régions d’un pays et souhaitez colorer chaque région en fonction de sa population, les données sur la population seraient considérées comme des données d’attribut.

Comme souligné ci-dessus, nous utiliserons le nombre de cas de paludisme signalés au Nigeria pour les années 2000, 2006, 2010, 2015 et 2021.

```
# Lecture des données d'attribut
malaria_cases <- read_csv(here::here("data/malaria.csv"))
malaria_cases
```

Étape 3 : Vérification des Données Jointes

Il est essentiel de vérifier et de valider les données jointes pour s'assurer que la fusion sera réussie et que les données seront exactes.

```
# Valider les données jointes  
all.equal(unique(nga_adm1$NAME_1), unique(malaria_cases$state_name))
```

```
## [1] "Lengths (38, 37) differ (string compare on first 37)"  
## [2] "2 string mismatches"
```

```
# Identifier les discordances  
setdiff(unique(nga_adm1$NAME_1), unique(malaria_cases$state_name))
```

```
## [1] "Water body"
```

Dans l'extrait de code fourni, nous comparons les noms de régions uniques entre les ensembles de données `nga_adm1` et `malaria_cases` en utilisant la fonction `all.equal()`. Ceci afin de s'assurer que toutes les régions du fichier de formes correspondent à leurs homologues dans les données d'attribut.

Si les noms de régions sont identiques, `all.equal()` renverra `TRUE`. Cependant, s'il y a des divergences, elle détaillera les différences entre les deux ensembles de noms de régions.

Il est noté que le “Plan d'eau” est présent dans le fichier de formes mais pas dans le `malaria_cases`. Comme “Plan d'eau” n'est pas une région propre, il devrait être retiré avant de joindre les ensembles de données. Nous pouvons le faire avec `filter()`.

```
nga_adm1 <- filter(nga_adm1, NAME_1 != "Plan d'eau")
```

Étape 4 : Joindre les Données par Niveaux Administratifs

Nous allons maintenant obtenir les données que nous voulons tracer sur la carte. L'important est que les noms des régions soient les mêmes dans le fichier de formes que dans vos données que vous souhaitez tracer, car cela sera nécessaire pour qu'ils fusionnent correctement.

Avant de joindre les deux ensembles de données, nous devons définir la clé de jointure (`by =`).

N'oubliez pas de revoir la leçon sur la jointure des tables pour plus de détails sur la façon de fusionner les data.frames dans R si vous n'êtes pas familier avec la jointure.

```

# Ajouter les données de population et calculer les cas pour 10K de population
malaria <- malaria_cases %>%
  left_join(nga_adm1,
            by = c("state_name" = "NAME_1")) %>%
  st_as_sf() # convertir en fichier de formes

# Sélectionner les colonnes les plus importantes
malaria2 <- malaria%>%
  select(state_name, cases_2000, cases_2006, cases_2010, cases_2015,
         cases_2021, geometry)

```

Dans cette étape, nous fusionnons les données du fichier de formes `nga_adm1` avec les données `malaria_cases` en utilisant `left_join()` du package `{dplyr}`. L'argument `by` est utilisé pour spécifier la colonne commune sur laquelle fusionner les ensembles de données.

Enfin, nous convertissons les données fusionnées en un fichier de formes en utilisant `st_as_sf()`.

Nous pouvons conserver uniquement les variables importantes qui seront utilisées dans la construction des tracés en utilisant `select()`.



Dans cette partie de la leçon, nous avons appris l'importance des niveaux administratifs dans les données spatiales et comment joindre les données spatiales et d'attribut par niveaux administratifs en utilisant le package `{dplyr}` dans R et comment vérifier et valider les données jointes.

Création d'une Carte Choroplète avec {ggplot2}

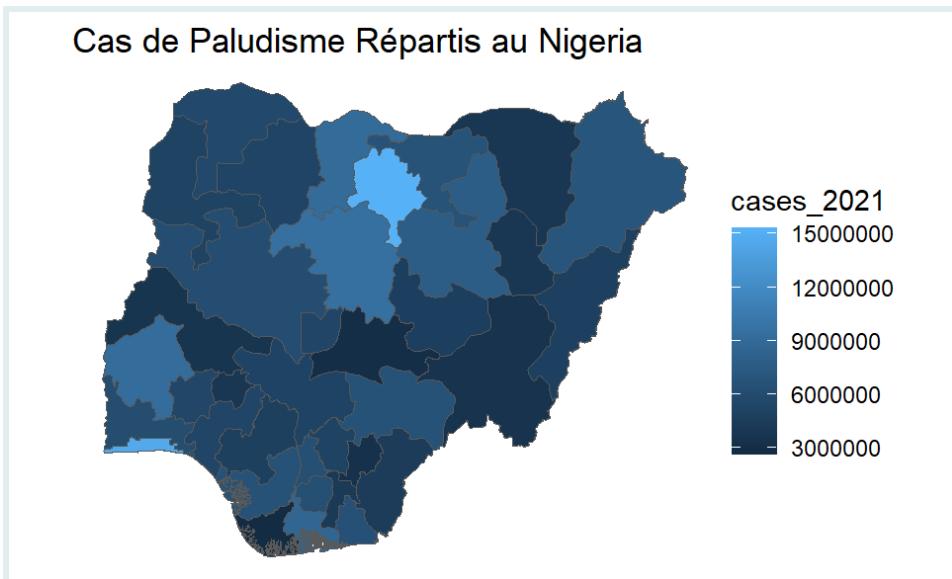
Utilisation de la Variable de Remplissage (c.-à-d. Variable d'Attribut)

Pour afficher le nombre de cas, par exemple pour 2021, nous devons remplir les polygones tracés avec `geom_sf()` en utilisant la variable `fill`. Cela est très simple car cela suit la même logique que la syntaxe du package `{ggplot2}`.

```

ggplot(data=malaria2) +
  geom_sf(aes(fill=cases_2021)) + # régler le remplissage pour varier selon
  # la variable de nombre de cas
  labs(title = "Cas de Paludisme Répartis au Nigeria") +
  theme_void()

```

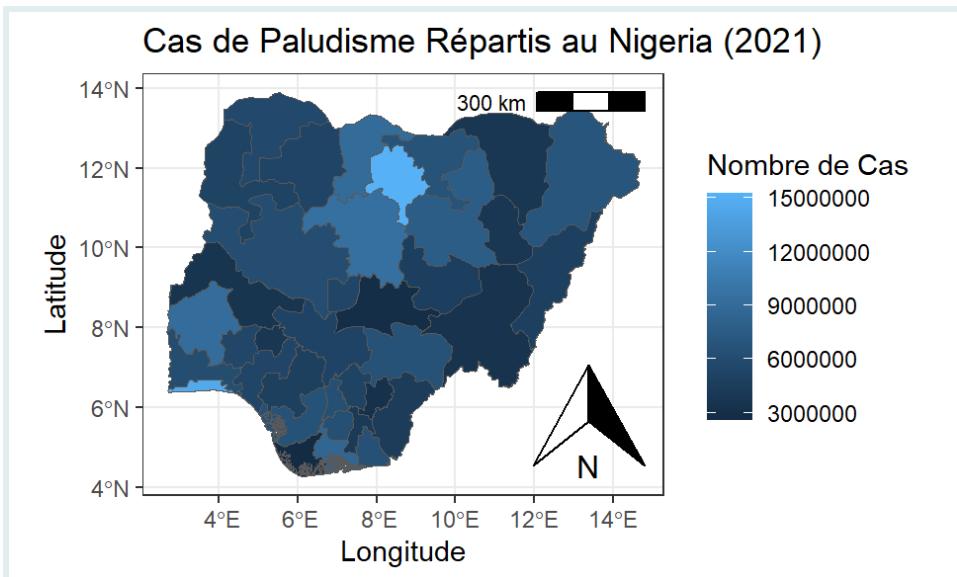


Nous créons une carte choroplèthe de base en utilisant {ggplot2}. L'esthétique `fill` est réglée pour varier selon la variable `cases`, ce qui colore les régions en fonction du nombre de cas de paludisme.

Personnalisation de la Carte

Nous pouvons personnaliser la carte en ajoutant des titres aux axes et à la légende, en ajoutant une flèche du nord et une barre d'échelle avec `ggspatial::annotation_north_arrow()` et `ggspatial::annotation_scale()`, et en changeant le thème pour `theme_bw()`.

```
ggplot(data=malaria2) +
  geom_sf(aes(fill=cases_2021)) + # régler le remplissage pour varier selon
  # la variable de nombre de cas
  labs(title = "Cas de Paludisme Répartis au Nigeria (2021)",
       fill = "Nombre de Cas") +
  xlab("Longitude") +
  ylab("Latitude") +
  ggspatial::annotation_north_arrow(location = "br") +
  ggspatial::annotation_scale(location = "tr") +
  theme_bw()
```



Dans cette section, nous avons d'abord créé une carte choroplèthe de base en utilisant `{ggplot2}` pour visualiser les cas régionaux de paludisme au Nigeria pour 2021. Ensuite, nous avons amélioré la carte en ajoutant des **titres**, des **étiquettes d'axes**, une **flèche du nord**, une **barre d'échelle** et en appliquant un **thème monochrome**.

Q : Construisez votre magnifique carte choroplèthe

Construisez une carte choroplèthe pour afficher la distribution des cas de paludisme en 2015, en utilisant la colonne `cases_2015` du jeu de données `malaria2`. Vous pouvez améliorer la conception et la clarté de votre carte en intégrant des titres, des étiquettes d'axes et toute autre étiquette pertinente.

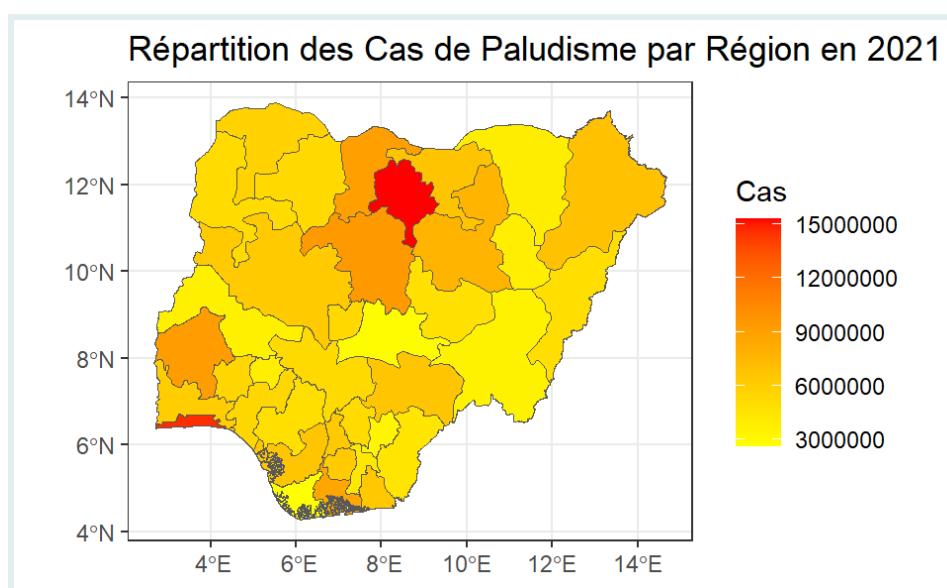
Mise à l'échelle des Couleurs

Mise à l'échelle des Couleurs Continue

La fonction `scale_fill_continuous()` du package `{ggplot2}` dans R est utilisée pour appliquer une échelle de couleurs continue à une carte choroplèthe.

Nous pouvons personnaliser la palette de couleurs utilisée pour l'échelle de couleurs continue en spécifiant les paramètres `low` et `high` dans la fonction `scale_fill_continuous()`.

```
# Créer un objet ggplot
ggplot(data = malaria2) +
  geom_sf(aes(fill = cases_2021)) +
  scale_fill_continuous(low = "yellow", high = "red", name = "Cas") + # appliquer une échelle de couleurs continue
  ggtitle("Répartition des Cas de Paludisme par Région en 2021") + # Ajouter le titre corrigé ici
  theme_bw()
```



RECAP



Dans cette section, nous avons appris comment appliquer une échelle de couleurs continue à une carte choroplète en utilisant la fonction `scale_fill_continuous()` du package `{ggplot2}` dans R et comment personnaliser la palette de couleurs.

Mise à l'échelle des Couleurs Discrete

Les données discrètes sont un type de données quantitatives qui ne peuvent prendre que des valeurs spécifiques et séparées. Elles sont souvent le résultat du dénombrement d'objets ou d'événements. Les données discrètes sont importantes dans

les cartes choroplèthes car elles nous permettent de représenter le nombre ou la quantité d'objets ou d'événements dans différentes régions.

La fonction `scale_fill_brewer()` du package `{ggplot2}` dans R est utilisée pour appliquer une échelle de couleurs discrète à une carte choroplète.

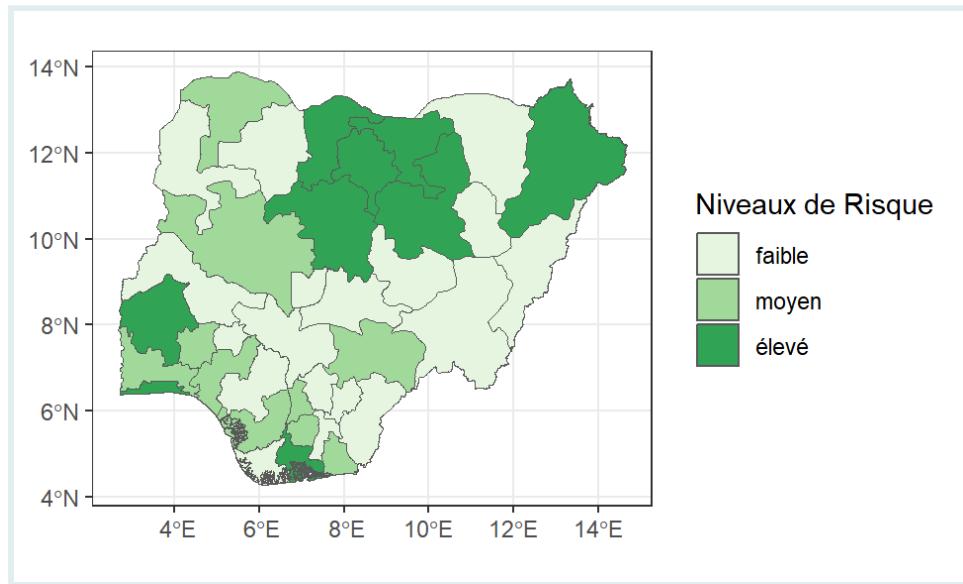
Avant d'appliquer l'échelle de couleurs discrète, nous devrons créer une nouvelle colonne discrète, qui pourrait être un niveau de risque basé sur le nombre de cas. Pour cela, nous pouvons utiliser `mutate()` combiné avec `case_when()`.

```

# Transformer les données : Créer une nouvelle colonne 'risk' basée sur le
# nombre de cas en 2021
malaria3 <- malaria2 %>%
  mutate(risk = case_when(cases_2021 < quantile(cases_2021, 0.5) ~ 'faible',
                          cases_2021 > quantile(cases_2021, 0.75) ~ 'élevé',
                          TRUE ~ 'moyen'))

# Visualiser les données
ggplot(data = malaria3) +
  geom_sf(aes(fill = fct_reorder(risk, cases_2021))) # Les niveaux de risque
  # sont réordonnés en fonction du nombre de cas
  scale_fill_brewer(palette = "Set4", "Niveaux de Risque") +
  theme_bw()

```



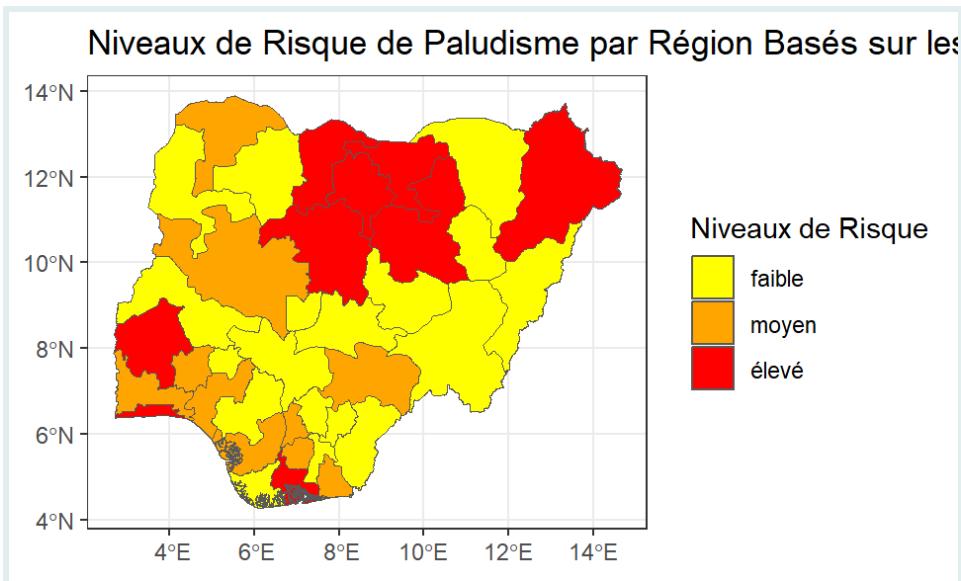
Vous pouvez également créer une palette de couleurs personnalisée pour les variables discrètes.

```

palette_personnalisée <- c("yellow", "orange", "red") # créer une palette de
# couleurs personnalisée manuellement

# Appliquer la palette de couleurs personnalisée
ggplot(data = malaria3) + # Créer un objet ggplot
  geom_sf(aes(fill = fct_reorder(risk, cases_2021))) + # réordonner les
  # étiquettes de risque selon le nombre de cas
  scale_fill_manual(values = palette_personnalisée, name = "Niveaux de
  Risque") +
  coord_sf(expand = TRUE) +
  ggtitle("Niveaux de Risque de Paludisme par Région Basés sur les Cas de
  2021") + # Ajouter un titre ici
  theme_bw()

```



Dans cette section, nous avons appris comment appliquer une échelle de couleurs discrète à une carte choroplèthe en utilisant la fonction `scale_fill_brewer()` du package `{ggplot2}` et comment personnaliser la palette de couleurs.



Q : Créez votre propre palette de couleurs

Créez votre propre palette de couleurs distincte de celle initiale, et affichez les cas de paludisme à travers le Nigeria pour 2000 en utilisant cette palette personnalisée. N'oubliez pas d'intégrer des améliorations esthétiques supplémentaires.

Facet Wrap vs. Grille

- `facet_wrap()`: Cette fonction enveloppe une séquence 1D de panneaux en 2D. C'est utile lorsque vous avez une seule variable avec de nombreux niveaux et que vous souhaitez organiser les graphiques de manière plus efficace en termes d'espace.
- `facet_grid()`: Cette fonction crée une matrice de panneaux définis par des variables de facettement en lignes et en colonnes. Elle est la plus utile lorsque vous avez deux variables discrètes et que toutes les combinaisons de variables existent dans les données.





- `facet_wrap()` est utilisé pour le facettement d'une seule variable, tandis que `facet_grid()` est utilisé pour le facettement de deux variables.
- `facet_wrap()` organise les panneaux en grille 2D, tandis que `facet_grid()` les organise en matrice.

Création de petits multiples avec `facet_wrap()`

Nous pouvons utiliser la fonction `facet_wrap()` pour créer de multiples petits graphiques basés sur une variable donnée. Dans l'exemple suivant, nous pouvons utiliser la variable `année`.

Rappelez-vous que l'échelle des couleurs peut être ajustée par `scale_fill_continuous()` si la variable est continue ou par `scale_fill_brewer()` si la variable est discrète.

Ici, nous devons faire un léger ajustement à la structure de notre jeu de données. Puisque les cas sont présentés par année et que chaque année est représentée dans une colonne séparée, nous utiliserons la fonction `pivot_longer()` pour les consolider dans une seule colonne, accompagnée d'une clé d'identifiant.

```
malaria3_longer <- malaria3 %>%
  pivot_longer(cols = `cases_2000`:`cases_2021`, names_to = "year", values_to =
    "cases")
```

Il semble que la variable `année` dans notre jeu de données ait un préfixe `cases_` que nous aimerais supprimer pour des raisons esthétiques avant de tracer.

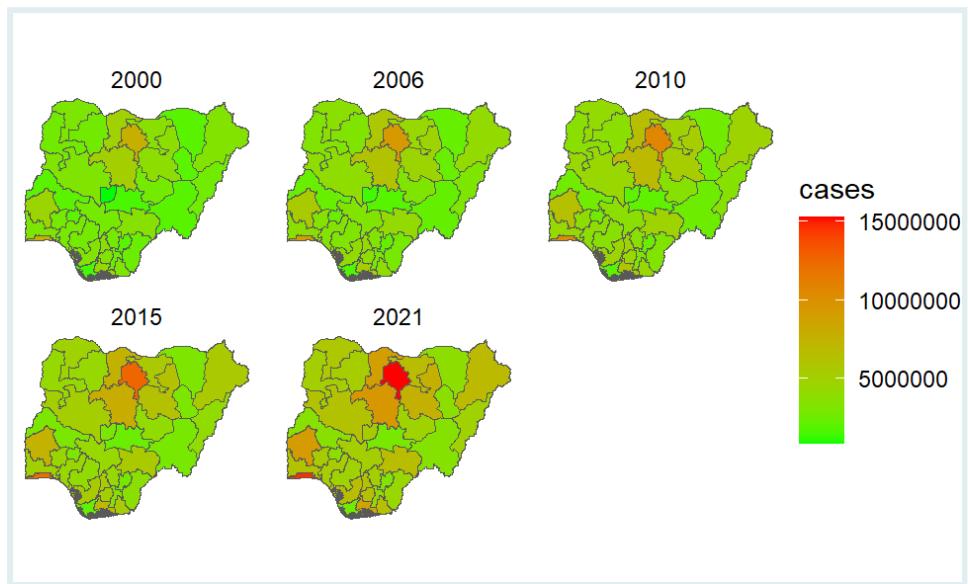
Voici comment nous pouvons le faire en utilisant la fonction `str_replace()` du package `{stringr}`, qui fait partie du `tidyverse` :

```
# Recoder la variable 'année' en supprimant le préfixe 'cases_'
malaria3_longer$year <- str_replace(malaria3_longer$year, "cases_", "")
```

Après avoir exécuté le code ci-dessus, la variable `année` dans notre jeu de données `malaria3_longer` n'aura plus le préfixe `cases_`, la rendant plus propre pour des fins de visualisation.

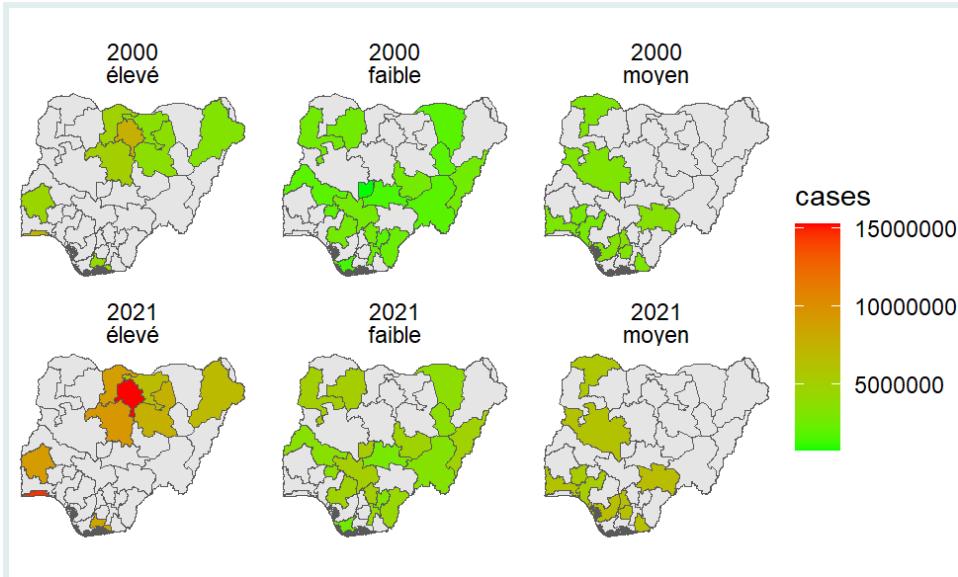
Maintenant, nous voulons créer de multiples petits graphiques en utilisant `facet_wrap()`.

```
# Créer un objet ggplot en utilisant facet_wrap
ggplot(data = malaria3_longer) +
  geom_sf(aes(fill = cases)) +
  facet_wrap(~ year) +
  scale_fill_continuous(low = "green", high = "red") + # appliquer l'échelle
  theme_void()
```



Nous pouvons ajouter une autre variable au `facet_wrap()`. Ici, nous ajoutons `risk`, mais nous utiliserons uniquement les cas de 2000 et 2021 à des fins de démonstration :

```
ggplot() +
  geom_sf(data = nga_adm1) +
  geom_sf(aes(fill = cases), data = filter(malaria3_longer, year %in%
    c("2000", "2021"))) +
  facet_wrap(year ~ risk) +
  coord_sf(expand = TRUE) +
  scale_fill_continuous(low = "green", high = "red") +
  theme_void()
```

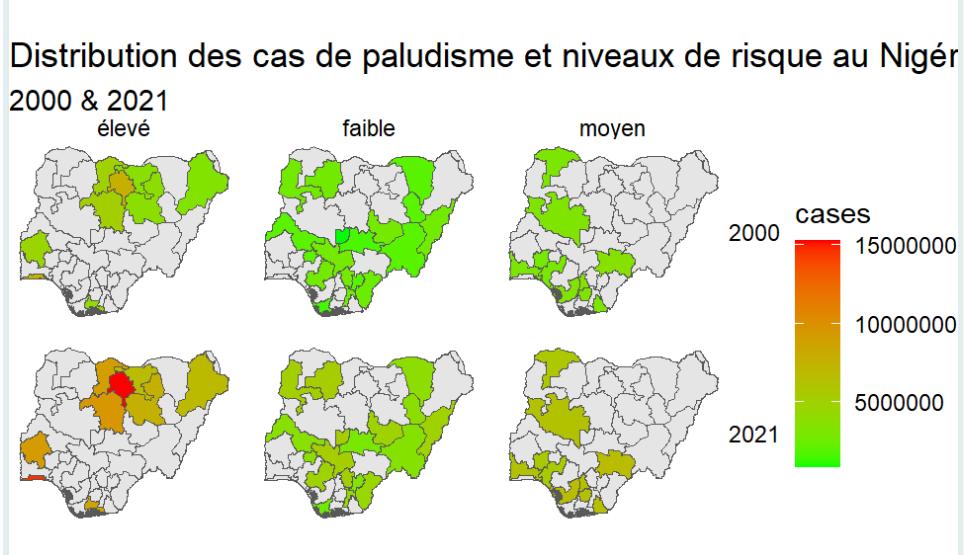


Dans le dernier graphique ci-dessous, vous remarquerez que nous avons utilisé une variable de remplissage continue, mais nous avons également utilisé un `facet_wrap` sur deux autres variables discrètes.

Création de petits multiples avec `facet_grid()`

Maintenant, nous utiliserons `facet_grid()` pour créer une grille de graphiques. Pour chaque année (2000 et 2021), des graphiques séparés sont réalisés pour chaque niveau de risque, offrant une comparaison visuelle facile à la fois sur les années et les risques.

```
ggplot() +
  geom_sf(data = nga_adm1) + # besoin du
  geom_sf(aes(fill = cases), data = filter(malaria3_longer, year
%in% c("2000", "2021"))) +
  facet_grid(year ~ risk) +
  coord_sf(expand = TRUE) +
  scale_fill_continuous(low = "green", high = "red") +
  labs(title = "Distribution des cas de paludisme et niveaux de risque au
    Nigéria",
       subtitle = "2000 & 2021") +
  xlab("Longitude") +
  ylab("Latitude") +
  theme_void()
```



PRO TIP



`facet_grid()` est idéal lorsque vous avez deux facteurs et que vous voulez explorer chaque combinaison. `facet_wrap()` est mieux lorsque vous avez juste un facteur ou lorsque vous avez plusieurs facteurs mais que vous voulez une disposition plus simple.

Le choix entre les deux dépend souvent des besoins spécifiques de vos données et de la manière dont vous souhaitez visualiser les relations.

Q : Analysez la distribution des cas de paludisme

CHALLENGE



Votre objectif maintenant est d'analyser la distribution des cas de paludisme au Nigéria pour les années 2000 et 2021. Mais vous devrez d'abord catégoriser les données en niveaux de risque en utilisant la médiane (haut/bas), puis visualiser ces informations sur une carte.



Dans cette section, nous avons appris les différences entre `facet_wrap()` et `facet_grid()`, et comment créer de multiples petits graphiques en utilisant ces deux fonctions.

En Résumé !

Aujourd’hui, nous sommes allés en profondeur dans le monde des cartes choroplèthes, débloquant le pouvoir de la représentation géographique visuelle.

Vous avez gagné des connaissances sur :

- L’essence et les composants d’une carte choroplète.
- Les avantages et inconvénients de ces cartes.
- Les essentiels de la préparation des données pour la visualisation choroplète.
- La création de cartes avec {ggplot2} et l’application de palettes de couleurs variées.
- L’utilisation de `facet_wrap()` et `facet_grid()` pour des conceptions de cartes complexes.

Résultats d’apprentissage

Félicitations pour avoir terminé cette leçon ! Récapitulons le voyage cognitif que vous avez entrepris :

1. Connaissance & Compréhension :

- Reconnu la définition et les composants d’une carte choroplète.
- Compris les avantages et les limitations des cartes choroplèthes.

2. Application :

- Préparé des données spécifiquement pour la création d’une carte choroplète.
- Utilisé {ggplot2} dans R pour concevoir une carte choroplète.
- Mis en œuvre des techniques d’échelle de couleurs continues et discrètes.

3. Analyse :

- Différencié entre l’échelle de couleurs continues et discrètes.

4. Synthèse :

- Intégré divers composants pour créer de multiples petits graphiques en utilisant `facet_wrap()` et `facet_grid()`.

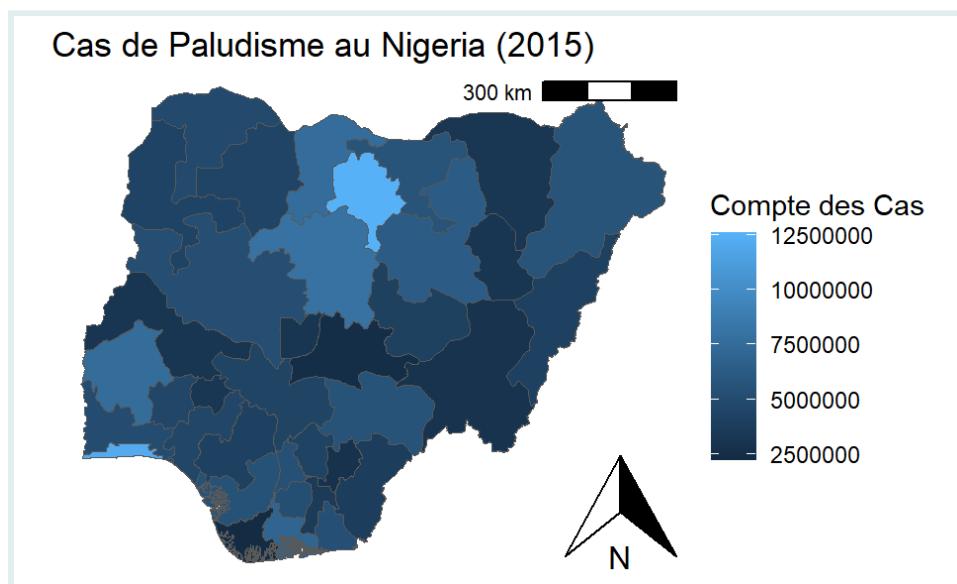
Avec ces compétences en main, vous êtes maintenant équipé pour innover avec différents ensembles de données et les visualiser de manière impactante. Plongez, expérimitez et améliorez votre voyage dans la création de cartes en utilisant {ggplot2} dans R. Bonne cartographie !

Solutions

1. Construisez votre magnifique carte choroplèthe

Construisez une carte choroplèthe pour afficher la répartition des cas de paludisme en 2019, en utilisant la colonne `cases_2019` du jeu de données `malaria2`. Vous pouvez améliorer le design et la clarté de votre carte en incorporant des titres, des étiquettes d'axes et toute autre étiquette pertinente.

```
ggplot(data=malaria2) +  
  geom_sf(aes(fill=cases_2015)) +  
  labs(title = "Cas de Paludisme au Nigeria (2015)",  
       fill = "Compte des Cas",  
       x = "Longitude",  
       y = "Latitude") +  
  ggspatial::annotation_north_arrow(location = "br") +  
  ggspatial::annotation_scale(location = "tr") +  
  theme_void()
```



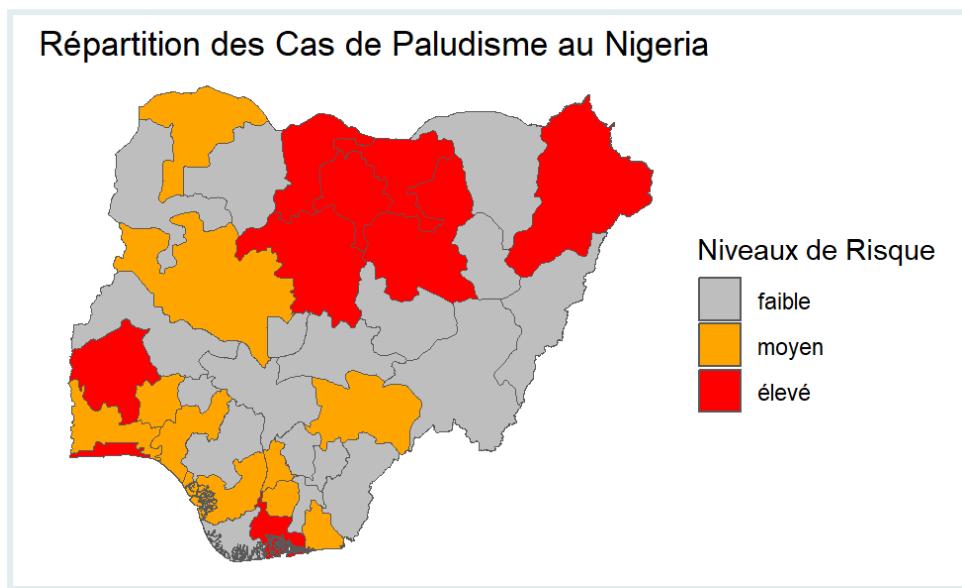
2. Créez votre propre palette de couleurs

Créez votre propre palette de couleurs distincte de celle initialement fournie ci-dessous, et affichez les cas de paludisme à travers le Nigeria pour 2000 en utilisant cette palette de couleurs personnalisée. N'oubliez pas d'incorporer des améliorations esthétiques supplémentaires.

```

new_palette <- c("grey", "orange", "red") # Définir un ensemble différent de
couleurs
ggplot(data=malaria3) +
  geom_sf(aes(fill = fct_reorder(risk, cases_2000))) + # Réordonner les
étiquettes de risque selon les cas de l'année 2000
  scale_fill_manual(values = new_palette, "Niveaux de Risque") +
  labs(title = "Répartition des Cas de Paludisme au Nigeria",
fill = "Compte des Cas",
x = "Longitude",
y = "Latitude") +
  coord_sf(expand = TRUE) +
  theme_void()

```



3. Analysez la distribution des cas de paludisme

Votre objectif est maintenant d'analyser la distribution des cas de paludisme au Nigeria pour les années 2000 et 2021. Mais vous devrez d'abord catégoriser les données en niveaux de risque en utilisant la médiane (haut/bas), puis visualiser ces informations sur une carte.

```

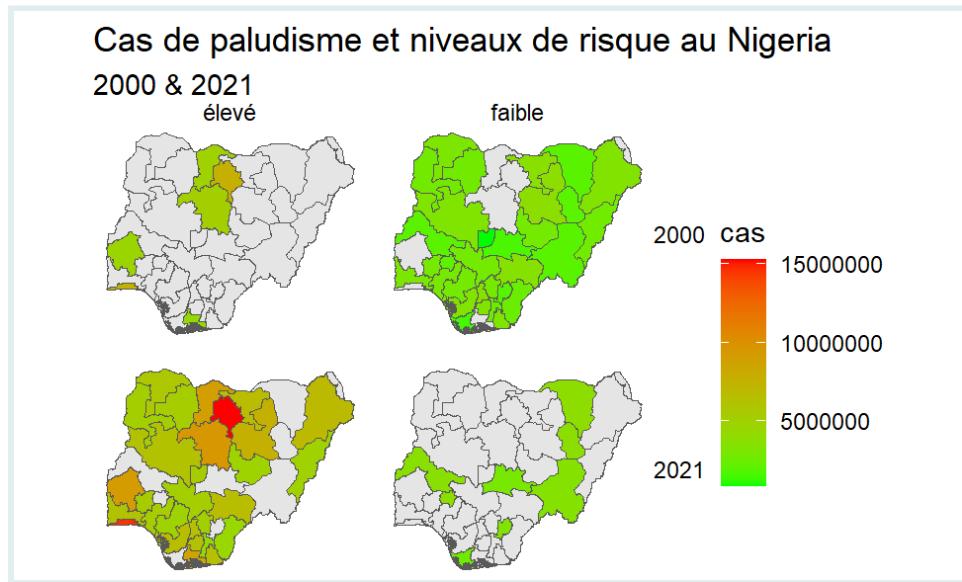
# Pivotement des données
malaria3_longer <- malaria2 %>%
  pivot_longer(cols = `cases_2000`:`cases_2021`, names_to = "année", values_to =
    "cas")

# Catégorisation du risque basée sur la médiane
malaria3_longer %>%
  mutate(risk = case_when(
    cas <= median(cas) ~ 'faible',
    cas > median(cas) ~ 'élevé'
  )) -> malaria3_longer2

# Nettoyage des valeurs d'année
malaria3_longer2$année <- str_replace(malaria3_longer2$année, "cases_", "")

# Tracé des données
ggplot() +
  geom_sf(data = nga_adm1) +
  geom_sf(aes(fill = cas), data = filter(malaria3_longer2, année %in%
    c("2000", "2021"))) +
  facet_grid(année ~ risk) +
  coord_sf(expand = TRUE) +
  scale_fill_continuous(low = "green", high = "red") +
  labs(title = "Cas de paludisme et niveaux de risque au Nigeria",
       subtitle = "2000 & 2021") +
  xlab("Longitude") +
  ylab("Latitude") +
  theme_void()

```



Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



IMAD EL BADISY

Data Science Education Officer
Deeply interested in health data



KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement



JOY VAZ

R Developer and Instructor, the GRAPH Network
Loves doing science and teaching science

Références

- Wickham, Hadley, Winston Chang, and Maintainer Hadley Wickham. “Package ‘ggplot2’.” *Create elegant data visualisations using the grammar of graphics.* Version 2, no. 1 (2016): 1-189.
- Wickham, Hadley, Mine Çetinkaya-Rundel, and Garrett Grolemund. *R for data science.* ” O'Reilly Media, Inc.”, 2023.
- Lovelace, Robin, Jakub Nowosad, Jannes Muenchow. *Geocomputation with R.* CRC Press, 2019.

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.



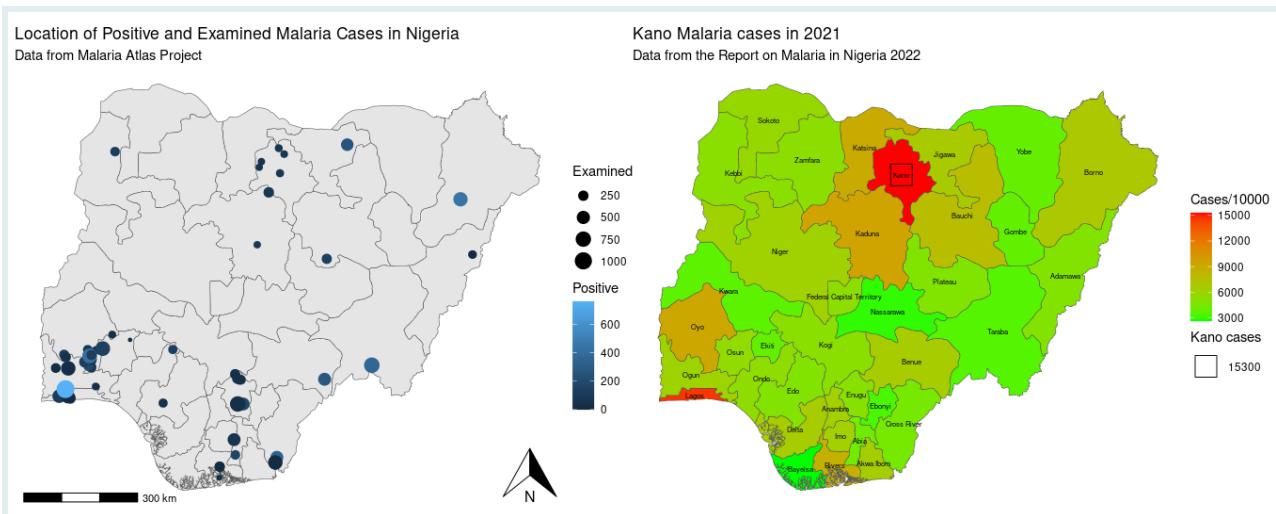
Amélioration des cartographies épidémiques avec des annotations

Introduction
Packages
Préparation des Données
Création d'une Carte Choroplèthe Simple
Ajout d'Indicateurs de Données Continues à la Carte Choroplèthe
Exploration des Taux d'Augmentation des Cas de Paludisme à l'aide de Cartes Choroplèthes
Étiquetage de la Carte Choroplèthe avec les Noms des États
Affichage des Noms des États Combinés et des Taux d'Augmentation sur la Carte Choroplèthe
Mettre en Évidence une Région Spécifique sur la Carte tout en Préservant le Contexte
Étiquetage des Emplacements des Points : Exploration du Taux de Positivité au Paludisme et de l'Incidence
Dernières Réflexions
Récapitulatif
Solutions
Références

Introduction

Dans la visualisation des données géospatiales, les cartes sont des outils puissants pour raconter des histoires. Cependant, une carte sans annotations claires et des étiquettes équivaut à un livre sans titres ni chapitres. Bien que l'histoire puisse toujours être là, il devient considérablement plus difficile de comprendre, d'interpréter et d'apprécier.

Dans cette leçon, nous mettons particulièrement l'accent sur l'importance de l'annotation et de l'étiquetage des cartes. Une annotation appropriée transforme une visualisation simple en un guide informatif, permettant aux spectateurs de saisir rapidement des données spatiales complexes. Grâce à un étiquetage précis, les zones d'intérêt peuvent être immédiatement reconnues, facilitant une compréhension plus claire de la narration des données.



Objectifs d'apprentissage

Objectifs d'apprentissage : Techniques Avancées de Visualisation Géospatiale

À la fin de cette section, vous devriez être capable de :

1. Incorporer des indicateurs de données continues dans les cartes choroplèthes pour une granularité accrue.
2. Calculer et visualiser les taux d'augmentation des cas de paludisme à l'aide de cartes choroplèthes.
3. Superposer efficacement les noms des États sur les cartes choroplèthes, en veillant à la clarté et à la lisibilité.
4. Intégrer sans heurts les noms des États avec les taux d'augmentation sur les cartes sans compromettre la lisibilité.
5. Appliquer des techniques pour mettre en évidence des régions spécifiques sur une carte tout en conservant le contexte global.
6. Déterminer des stratégies optimales de placement des points et les intégrer efficacement dans les visualisations géospatiales.

En maîtrisant ces compétences, vous disposerez des connaissances et outils nécessaires pour créer des visualisations géospatiales riches, détaillées et informatives.

Packages

```
# Charger les bibliothèques
if(!require(pacman)) install.packages("pacman")
pacman::p_load(malariaAtlas,
  ggplot2,
  geodata,
  dplyr,
  here,
  readr,
  sf,
  patchwork)

# Désactiver la notation scientifique
options(scipen=100000)
```

Préparation des Données

Avant de plonger dans toute visualisation ou analyse, il est essentiel de charger et de prétraiter nos données. Cela comprend la lecture des datasets, la fusion des informations connexes et la filtration des entrées inutiles ou non pertinentes.

```
# Lecture des données géographiques au format shapefile
nga_adm1 <- sf::st_read(here::here("data/raw/NGA_adm_shapefile/NGA_adm1.shp"))
```

```
## Reading layer `NGA_adm1' from data source
## 
## C:\GitHub\graph_courses\epi_reports_staging\EPIREP_FR_choropleth_maps_labeling\data\raw\
##   using driver `ESRI Shapefile'
## Simple feature collection with 38 features and 9 fields
## Geometry type: MULTIPOLYGON
## Dimension:      XY
## Bounding box:  xmin: 2.668431 ymin: 4.270418 xmax: 14.67642 ymax: 13.89201
## Geodetic CRS:  WGS 84
```

Les données géographiques au format shapefile des limites administratives du Nigeria (comme les États ou les provinces) sont lues et stockées dans l'objet `nga_adm1`.

```
# Lecture des données attributives relatives aux cas de paludisme
malaria_cases <- read_csv(here::here("data/malaria.csv"))
```

Cette étape charge les données relatives aux cas de paludisme dans différents États du Nigeria.

```
# Filtrage des entrées 'Water body' des données géographiques
nga_adm1 <- filter(nga_adm1, NAME_1 != "Water body")

# Fusion des données géographiques avec les données sur le paludisme
malaria <- malaria_cases %>%
  left_join(nga_adm1, by = c("state_name" = "NAME_1")) %>%
  st_as_sf()
```

Ici, nous combinons les données sur les limites géographiques avec les données sur les cas de paludisme en utilisant les noms des États comme référence. Ces données fusionnées sont ensuite converties dans un format adapté aux visualisations géospatiales.

```
# Filtrage pour ne conserver que les colonnes essentielles pour notre analyse
malaria2 <- malaria %>%
  select(state_name, cases_2000, cases_2006, cases_2010, cases_2015,
         cases_2021, geometry)
```

Nous réduisons notre dataset à des colonnes spécifiques, principalement les noms des États, les cas de paludisme de différentes années et les limites géographiques de ces États (c'est-à-dire `geometry`).

```
# Lecture des données de population pour différentes régions du Nigeria
population_nigeria <- read_csv(here::here("data/population_nigeria.csv"))
```

Cette étape charge les données indiquant la population des différents États ou régions du Nigeria.

```
# Combinaison des données de population avec nos données sur le paludisme
malaria3 <- malaria2 %>%
  left_join(population_nigeria, by = c("state_name")) %>%
  st_as_sf()
```

En fusionnant les données de population avec nos données sur les cas de paludisme, nous enrichissons notre dataset. Ces données combinées permettent des visualisations et des analyses plus complètes, telles que le calcul des taux d'incidence ou l'évaluation des tendances par rapport à la taille de la population.

Création d'une Carte Choroplète Simple

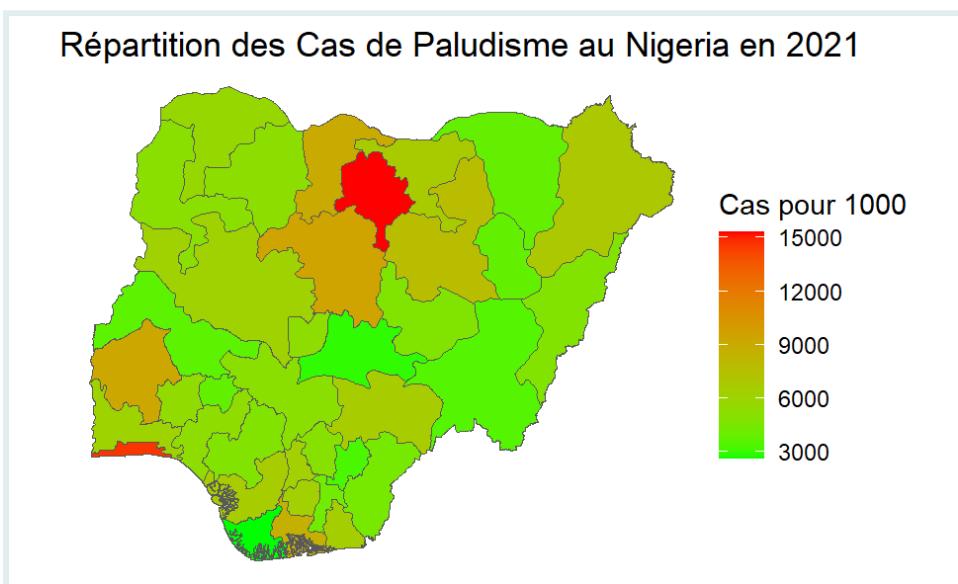
REMINDER



Les cartes choroplèthes sont des outils de visualisation puissants qui affichent des zones géographiques divisées ombrées ou hachurées en proportion de la valeur d'une variable.

Dans cet exemple, nous allons utiliser une carte choroplète pour visualiser la répartition des cas de paludisme dans différentes régions du Nigeria pour l'année 2021.

```
# Construction de la carte choroplète à l'aide de ggplot2
ggplot(data=malaria3) +
  geom_sf(aes(fill=cases_2021/1000)) + # La couleur de remplissage est
  # déterminée par le nombre de cas de paludisme en 2021, mis à l'échelle
  # par 1000
  labs(title = "Répartition des Cas de Paludisme au Nigeria en 2021", fill =
    "Cas pour 1000") + # Ajout de libellés et du titre à la carte
  scale_fill_continuous(low = "green", high = "red") + # Utilisation d'une
  # échelle de couleur continue allant du vert au rouge
  theme_void() # Utilisation d'un thème minimal pour une meilleure
  # visualisation de la carte
```



Voici une explication détaillée du code :

- `ggplot(data=malaria3)`: Initialise un objet ggplot en utilisant le dataset malaria3.
- `geom_sf(aes(fill=cases_2021/1000))`: Ajoute les données géographiques de malaria3 et remplit chaque région en fonction du nombre de cas de paludisme en 2021, mis à l'échelle par un facteur de 1000. Cela représente efficacement le nombre de cas pour mille habitants.
- `labs(title = "Répartition des Cas de Paludisme au Nigeria en 2021", fill = "Cas pour 1000")`: Spécifie le titre de la carte et l'étiquette de l'échelle de couleur.
- `scale_fill_continuous(low = "green", high = "red")`: Applique une échelle de couleur continue où les régions avec moins de cas sont colorées en vert et les régions avec plus de cas sont colorées en rouge.

- `theme_void()`: Supprime le texte des axes, les marques de graduations et autres éléments non essentiels pour mettre en évidence la carte.

Le graphique résultant offre une vue claire de la répartition des cas de paludisme au Nigeria en 2021, avec l'intensité des couleurs indiquant l'ampleur des cas dans chaque région.

Exercice 1 : Modifiez la carte choroplèthe fournie pour visualiser la répartition des cas de paludisme au Nigeria pour l'année 2015

Instructions

1. Mettez à jour la correspondance des données dans la fonction `geom_sf()` pour refléter les cas de paludisme pour l'année 2015.
2. Ajustez le titre dans la fonction `labs()` pour indiquer que la visualisation concerne l'année 2015.
3. Modifiez le dégradé de couleurs dans `scale_fill_continuous()` pour passer du bleu (cas "faibles") au jaune (cas "élevés").



Ci-dessous, un code de départ :

```
# Construction de la carte choroplèthe pour 2015 à l'aide de
# ggplot2
ggplot(data=malaria3) +
  geom_sf(aes(fill=_____)) + # Remplissez la colonne de
  # données correcte pour 2015
  labs(title = "_____", fill = "Cas pour
    1000") + # Mettez à jour le titre de manière appropriée
  scale_fill_continuous(_____) + # Modifiez l'échelle de
  # couleurs
  theme_void()
```

Ajout d'Indicateurs de Données Continues à la Carte Choroplèthe

Lors de l'analyse de données sur les maladies, il est souvent utile de regarder au-delà des chiffres bruts de cas et de se concentrer sur les taux, en particulier les taux d'incidence. Le taux d'incidence fournit une mesure normalisée qui peut prendre en compte les différences de taille de population entre les régions, rendant les comparaisons plus significatives.

Compréhension de l'Incidence

L'incidence d'une maladie est un pilier de la recherche épidémiologique. Elle quantifie le nombre de nouveaux cas d'une maladie survenant au cours d'une période spécifique, généralement une année, par rapport à une population à risque.

Mathématiquement, il est donné par :

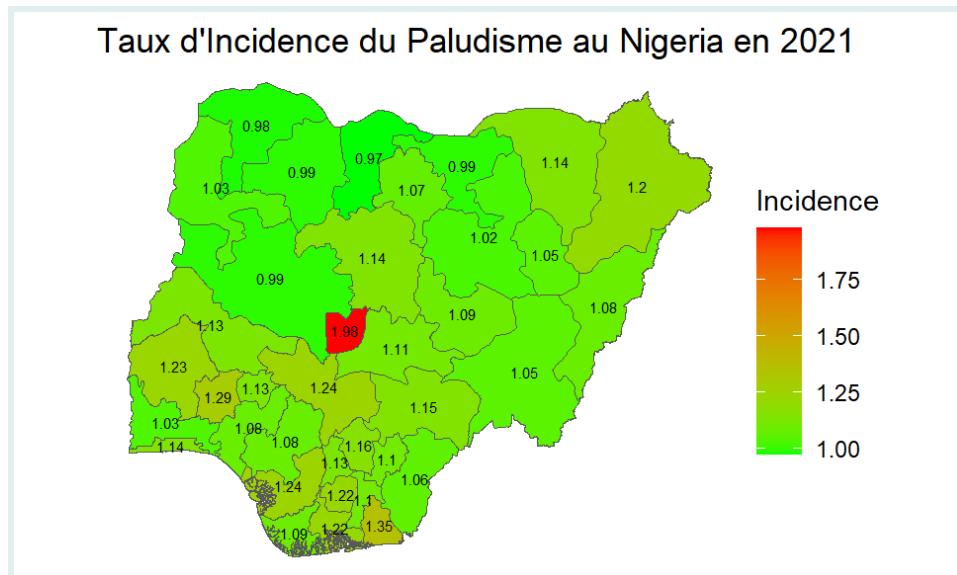
$$\text{Incidence} = \frac{\text{Nombre de nouveaux cas au cours de la période}}{\text{Population à risque au début de la période}}$$

Dans ce contexte, nous examinons l'incidence du paludisme dans différents États du Nigeria pour l'année 2021. Plus précisément, nous calculerons le taux d'incidence en divisant le nombre de nouveaux cas de paludisme en 2021 par la population de chaque État issue du dernier recensement disponible en 2019.

Le code R suivant réalise cela et visualise les données :

```
# Trouver les coordonnées du centre de chaque État pour positionner les étiquettes
centroid_coords <- st_coordinates(st_centroid(malaria2$geometry))

# Visualisation du Taux d'Incidence du Paludisme en 2021 à l'aide d'une Carte Choroplète
ggplot(data = malaria3) +
  # Remplir chaque État avec une couleur représentant le taux d'incidence en 2021.
  geom_sf(aes(fill = round(cases_2021/population_2019, 2))) +
  # Étiqueter chaque État avec son taux d'incidence spécifique.
  geom_text(aes(x = centroid_coords[, 1], y = centroid_coords[, 2], label =
    round(cases_2021/population_2019, 2)), size = 2) +
  # Ajout de titre et de titre de légende.
  labs(title = "Taux d'Incidence du Paludisme au Nigeria en 2021", fill =
    "Incidence") +
  # Utilisation d'une échelle de couleur continue du vert (faible incidence) au rouge (forte incidence).
  scale_fill_continuous(low = "green", high = "red") +
  # Utilisation d'un thème minimaliste pour une visualisation plus claire.
  theme_void()
```



Voici une brève explication de la visualisation :

- La fonction `geom_sf()` crée une carte choroplète où la couleur de chaque État représente son taux d'incidence du paludisme en 2021.
 - La fonction `geom_text()` étiquète chaque État avec son taux d'incidence spécifique, positionnant chaque étiquette au centre de l'État.
 - L'échelle de couleurs, passant du vert au rouge, met en évidence visuellement les régions présentant des taux d'incidence plus élevés.

Cette visualisation offre une compréhension immédiate de la situation du paludisme au Nigeria, révélant les zones à forte incidence qui pourraient nécessiter des interventions de santé publique plus ciblées.

Exploration des Taux d'Augmentation des Cas de Paludisme à l'aide de Cartes Choroplèthes

Comprendre l'évolution du nombre de cas de maladies au fil du temps peut fournir des informations sur l'efficacité des interventions, la progression de la maladie et les zones où des ressources accrues pourraient être nécessaires. Dans ce contexte, nous cherchons à visualiser l'augmentation en pourcentage des cas de paludisme de 2015 à 2021 dans différents États du Nigeria.

Calcul du Taux d'Augmentation

Le taux d'augmentation pour chaque État est calculé comme suit :

$$\text{Taux d'Augmentation} = \left(\frac{\text{Cas en 2021} - \text{Cas en 2015}}{\text{Cas en 2015}} \right) \times 100\%$$

Cette formule fournit la croissance en pourcentage (ou la diminution) des cas de paludisme de 2015 à 2021.

Visualisation des Taux d'Augmentation

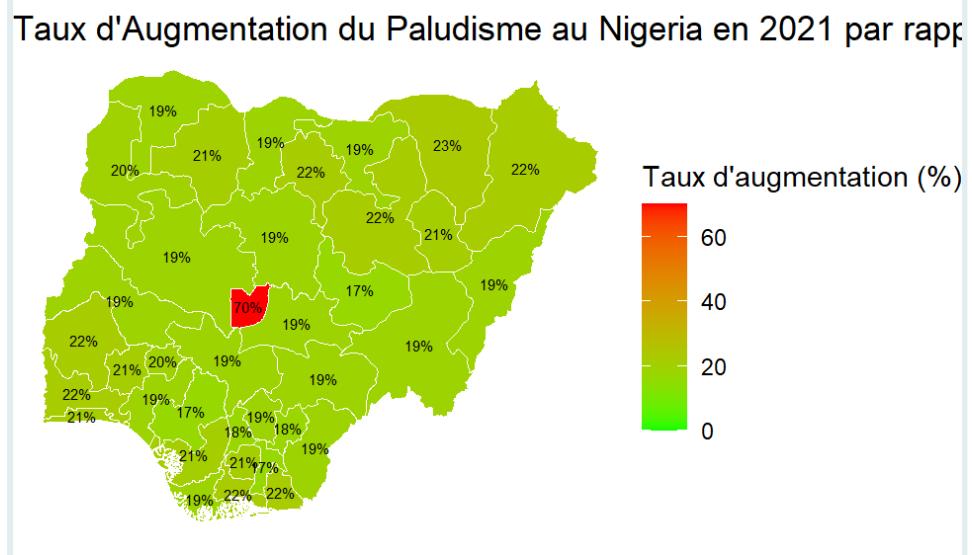
Plongeons dans le code qui réalise cette visualisation :

```
# Calculer le taux d'augmentation pour chaque État
malaria3 %>%
  mutate(increase_rate = round(((cases_2021 - cases_2015) / cases_2015) *
    100)) -> malaria3

# Visualisation des taux d'augmentation à l'aide d'une carte choroplèthe
ggplot(data = malaria3) +
  # Colorer chaque État en fonction de son taux d'augmentation
  geom_sf(aes(fill = increase_rate), color="white", size = 0.2) +
  # Utilisation d'une échelle de couleur continue pour représenter les taux
  # d'augmentation, passant du vert au rouge
  scale_fill_continuous(name="Taux d'augmentation (%)", limits=c(0,70), low =
    "green", high = "red", breaks=c(0, 20, 40, 60))+

  # Étiquetage de chaque État avec son taux d'augmentation en pourcentage
  geom_text(aes(x = centroid_coords[, 1], y = centroid_coords[, 2], label =
    paste0(increase_rate, "%"))), size = 2)+

  # Ajout d'un titre à la carte
  labs(title = "Taux d'Augmentation du Paludisme au Nigeria en 2021 par
    rapport à 2015")+
  theme_void()
```



Dans cette visualisation :

- La fonction `geom_sf()` crée la carte choroplète où l'intensité de couleur de chaque État représente son taux d'augmentation du paludisme.
- `scale_fill_continuous()` définit l'échelle de couleur pour les taux d'augmentation, mettant en évidence davantage les zones à forte augmentation.
- `geom_text()` ajoute des étiquettes à chaque État, fournissant des valeurs exactes en pourcentage.
- La carte résultante nous permet d'identifier rapidement les régions avec une forte croissance des cas de paludisme sur la période sélectionnée.

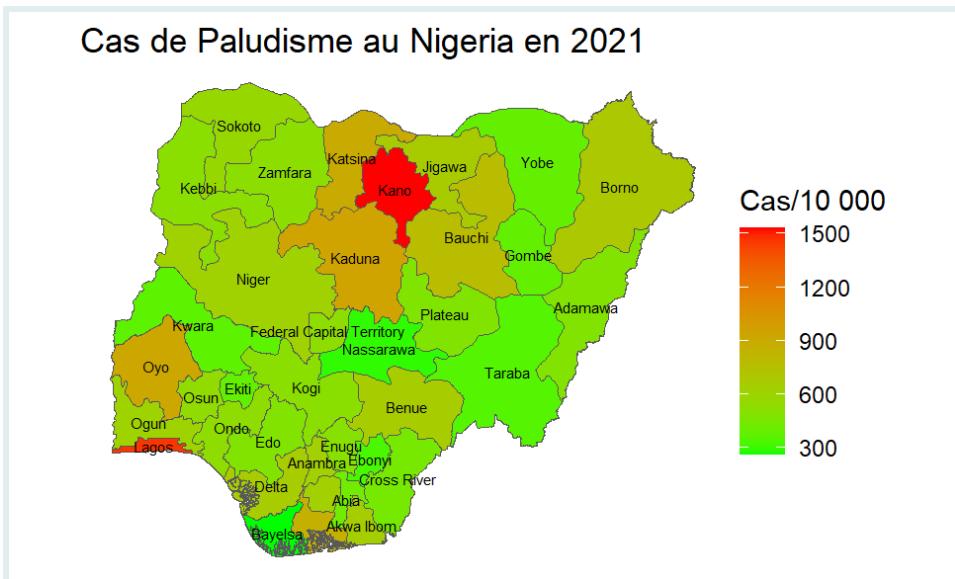
Grâce à cette visualisation, les parties prenantes peuvent cibler les régions où le paludisme est en augmentation et potentiellement allouer plus efficacement des ressources.

Étiquetage de la Carte Choroplète avec les Noms des États

Dans une carte choroplète, bien que les dégradés de couleurs offrent un indice visuel pour comprendre la répartition d'une variable dans différentes régions, l'ajout d'étiquettes peut considérablement améliorer la clarté de la visualisation. Cela est particulièrement vrai lorsque le public n'est peut-être pas familier avec toutes les limites géographiques affichées. Dans le cas de notre dataset sur le paludisme, l'ajout des noms des États à la carte rend les données plus accessibles et compréhensibles.

Analysons le code :

```
# Construction d'une carte choroplète avec les noms des États
ggplot(data = malaria3) +
  # Remplir chaque État en fonction du nombre de cas de paludisme en 2021,
  # normalisé par 10 000
  geom_sf(aes(fill = cases_2021/10000)) +
  # Ajouter le nom de chaque État à son centre
  geom_text(aes(x = centroid_coords[, 1], y = centroid_coords[, 2], label =
    state_name), size = 2, check_overlap = TRUE) +
  # Ajouter des titres et des étiquettes
  labs(title = "Cas de Paludisme au Nigeria en 2021", fill = "Cas/10 000") +
  # Utiliser une échelle de couleur continue du vert (faible nombre de cas) au
  # rouge (nombre élevé de cas)
  scale_fill_continuous(low = "green", high = "red") +
  theme_void()
```



Voici une explication détaillée de la visualisation :

- `geom_sf(aes(fill = cases_2021/10000))` : Cela crée la carte choroplète où la couleur de chaque État représente le nombre de cas de paludisme en 2021, normalisé par un facteur de 10 000.
- `geom_text(aes(...))` : Cette fonction place du texte sur le graphique. Dans ce cas, elle est utilisée pour ajouter le nom de chaque État à son centre géographique. Le paramètre `check_overlap = TRUE` permet à ggplot de vérifier les chevauchements de libellés de texte et de tenter d'éviter les chevauchements.
- `labs(title = "Cas de Paludisme au Nigeria en 2021", fill = "Cas/10 000")` : Cette fonction ajoute un titre au graphique et un libellé à l'échelle de couleur.
- `scale_fill_continuous(low = "green", high = "red")` : Cela définit l'échelle de couleur de la carte, en transitionnant du vert pour les États avec moins de cas au rouge pour ceux avec plus de cas.

La visualisation résultante est une carte claire et informative des cas de paludisme au Nigeria en 2021, avec chaque État étiqueté pour une référence facile.

Affichage des Noms des États Combinés et des Taux d'Augmentation sur la Carte Choroplète

Les visualisations peuvent transmettre une multitude d'informations lorsqu'elles intègrent de multiples points de données de manière intuitive. En associant les noms

des États à leurs taux d'augmentation respectifs, nous pouvons fournir une vue plus riche et plus détaillée des données sans submerger le public.

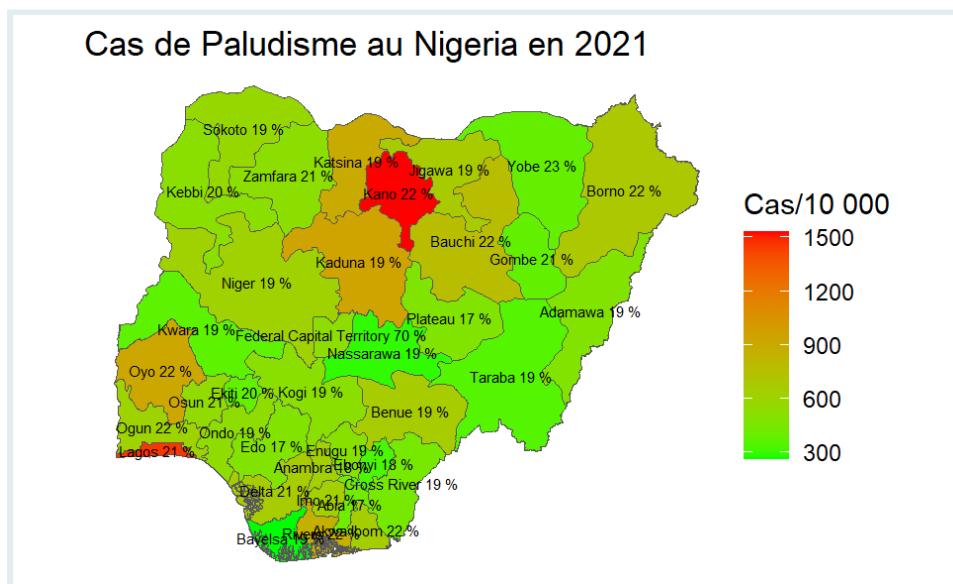
Plongeons dans cette visualisation :

Nous souhaitons présenter une carte choroplèthe montrant les cas de paludisme pour 10 000 habitants dans les États nigérians en 2021, avec des étiquettes combinant les noms des États et leurs taux d'augmentation depuis 2015.

```
# Combinez les noms des États et leurs taux d'augmentation respectifs en une seule étiquette
malaria3$label_text <- paste(malaria3$state_name, malaria3$increase_rate, "%")

# Calculez le centre de chaque État pour servir de points de référence pour les étiquettes
centroid_coords <- st_coordinates(st_centroid(malaria3$geometry))

# Visualisez les données
ggplot(data = malaria3) +
  # Créez une carte choroplèthe ombrée en fonction du nombre de cas de paludisme en 2021 pour 10 000 habitants
  geom_sf(aes(fill = cases_2021/10000)) +
  # Ajoutez des étiquettes combinées (nom de l'État et taux d'augmentation) au centre de chaque État
  geom_text(aes(x = centroid_coords[, 1],
                y = centroid_coords[, 2],
                label = label_text), size = 2) +
  # Ajoutez des titres et personnalisez la légende des couleurs
  labs(title = "Cas de Paludisme au Nigeria en 2021", fill = "Cas/10 000") +
  scale_fill_continuous(low = "green", high = "red") +  # Définir la gradation des couleurs
  theme_void()  # Appliquez un thème minimal pour plus de clarté
```



Dans cette visualisation :

- La ligne `malaria3$label_text <- paste(malaria3$state_name, malaria3$increase_rate, "%")` construit nos étiquettes combinées en concaténant le nom de l'État avec son taux d'augmentation, suivi d'un signe de pourcentage.
- `geom_text()` ajoute ces étiquettes combinées au graphique. Il positionne chaque étiquette au centre de l'État correspondant, en veillant à ce que les étiquettes soient centrées et facilement associées à leurs régions respectives.
- `scale_fill_continuous(low = "green", high = "red")` attribue une gradation de couleurs en fonction du nombre de cas de paludisme. Les États avec moins de cas seront colorés en vert, passant au rouge pour les États avec plus de cas.

Cette approche nous permet de communiquer efficacement deux informations essentielles (cas pour 10 000 habitants et taux d'augmentation) dans la même visualisation tout en conservant le nom des États pour une identification aisée.

~~Mettre en~~ Évidence une Région Spécifique sur la Carte tout en Préservant le Contexte

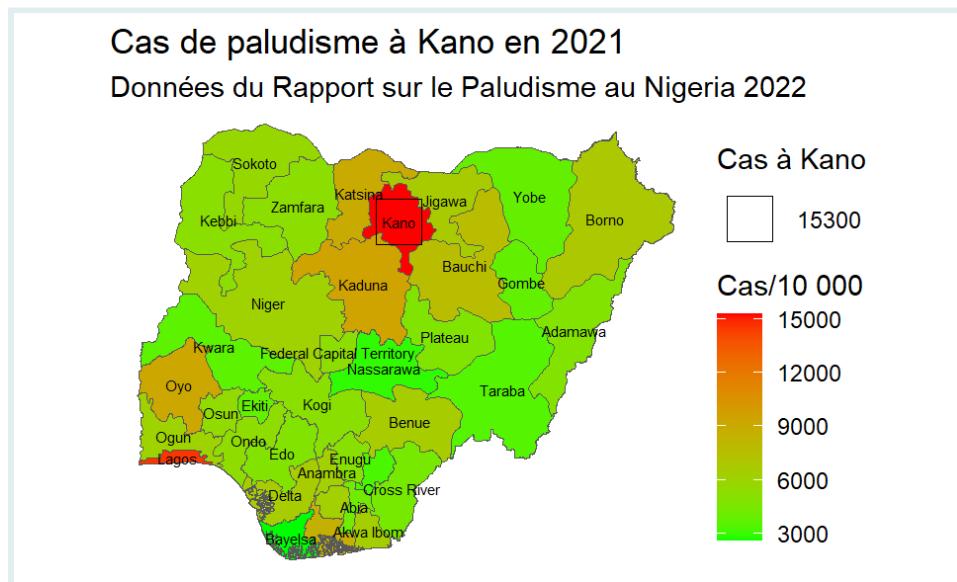
Parfois, vous souhaitez attirer l'attention sur une zone ou une région particulière de votre carte sans omettre les détails des régions environnantes. En utilisant des éléments graphiques spécifiques, tels que des marqueurs plus grands ou des couleurs distinctes, vous pouvez mettre l'accent sur certaines régions tout en mettant en valeur les données plus larges. Dans cet exemple, nous mettons l'accent sur la région de “Kano” au Nigéria.

```

# Calcul des coordonnées du centroïde pour l'étiquetage
centroid_coords <- st_coordinates(st_centroid(malaria3$geometry))

# Visualisation des cas de paludisme au Nigeria avec une mise en avant de Kano
ggplot(data = malaria3) +
  # Création d'une carte choroplète avec des couleurs basées sur les cas de
  # paludisme en 2021
  geom_sf(aes(fill = cases_2021/1000)) +
  # Définition d'une échelle de couleurs continues du vert au rouge
  scale_fill_continuous(low = "green", high = "red") +
  # Superposition d'un point sur Kano pour le mettre en avant. La taille du
  # point correspond au nombre de cas
  geom_point(data = subset(malaria3, state_name == "Kano"),
             aes(x = st_coordinates(st_centroid(geometry))[1],
                  y = st_coordinates(st_centroid(geometry))[2], size =
round(cases_2021/1000)),
             color = "black", shape = 22, fill = "transparent") +
  # Étiquetage de chaque région avec son nom
  geom_text(aes(x = centroid_coords[, 1], y = centroid_coords[, 2], label =
state_name), size = 2, check_overlap = TRUE) +
  # Personnalisation de l'échelle de taille du point mis en avant
  scale_size_continuous(range = c(1, 12)) +
  # Ajout du titre et des légendes
  labs(title = "Cas de paludisme à Kano en 2021", subtitle = "Données du
Rapport sur le Paludisme au Nigeria 2022", size = "Cas à Kano", fill =
"Cas/10 000",) +
  # Application d'un thème minimal pour plus de clarté
  theme_void()

```



Dans cette visualisation :

- La fonction `geom_point()` est utilisée pour placer un cercle sur la région de Kano. La taille du cercle représente le nombre de cas à Kano en 2021. Le cercle est conçu en mode transparent (`fill = "transparent"`) avec une bordure noire audacieuse (`color = "black"`) pour le mettre en évidence.
- `geom_text()` ajoute les noms de toutes les régions sur la carte. Il place chaque étiquette près du centre géographique de la région respective. Le paramètre `check_overlap = TRUE` empêche les étiquettes de se chevaucher autant que possible, garantissant ainsi la lisibilité.

RECAP



Tandis que “Kano” est mis en évidence, toutes les autres régions sont également affichées avec leur ombrage de couleur respectif basé sur les cas de paludisme. Cela offre une vue holistique de la situation à travers le Nigéria, permettant aux téléspectateurs de comparer Kano avec les autres régions.

Une telle approche est inestimable lorsque vous souhaitez mettre en lumière des détails spécifiques ou des zones d'intérêt sans négliger le jeu de données plus large, enrichissant ainsi vos présentations de données.

Étiquetage des Emplacements des Points : Exploration du Taux de Positivité au Paludisme et de l'Incidence

La cartographie et la visualisation de données spécifiques sur une carte géographique peuvent fournir des informations cruciales, notamment lors de la manipulation de données épidémiologiques. Plongeons dans le code pour comprendre les processus et la visualisation que nous cherchons à réaliser :

```

# Récupération des Données
# Le package malariaAtlas fournit la fonction `getPR` pour accéder au taux de
# parasite (PR).
# Le PR est un indicateur essentiel de la prévalence du paludisme.

# Récupération des données pour le Nigéria pour les deux espèces de paludisme
nigeria_pr <- malariaAtlas::getPR(ISO = "NGA", species = "both") %>%
  # Filtrer les enregistrements sans valeurs de PR
  filter(!is.na(pr)) %>%
  # Supprimer toutes les lignes avec des valeurs manquantes de longitude ou de
  # latitude
  drop_na(longitude, latitude) %>%
  # Convertir les données en un dataframe spatial pour faciliter la
  # cartographie
  st_as_sf(coords = c("longitude", "latitude"), crs = 4326)

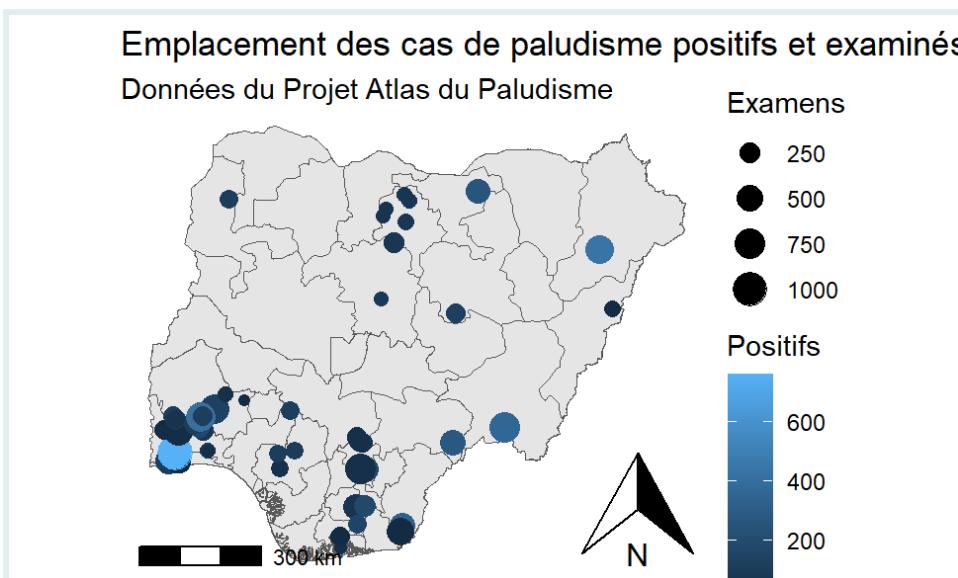
```

Ce bloc de code récupère les données de prévalence du paludisme pour le Nigéria. Après la récupération, les données sont nettoyées en filtrant les valeurs manquantes. Ensuite, elles sont transformées en un format adapté à la visualisation géospatiale (objet `sf`).

```

# Configuration d'une visualisation géospatiale avec ggplot2
ggplot() +
  # Traçage des limites administratives du Nigeria
  geom_sf(data = nga_adm1) +
  # Ajout de points pour chaque site de test
  # La couleur de chaque point indique si le test est positif, et la taille
  # représente le nombre de personnes testées
  geom_sf(aes(size = examined, color = positive), data = nigeria_pr) +
  # Définition des titres, des étiquettes des axes et des légendes du
  # graphique
  labs(title = "Emplacement des cas de paludisme positifs et examinés au
    Nigeria",
       subtitle = "Données du Projet Atlas du Paludisme",
       color = "Positifs",
       size = "Examens") +
  # Ajout d'étiquettes pour la longitude et la latitude
  xlab("Longitude") +
  ylab("Latitude") +
  # Intégration d'une flèche indiquant le nord pour l'orientation
  ggspatial::annotation_north_arrow(location = "br") +
  # Intégration d'une barre d'échelle pour aider à l'interprétation des
  # distances
  ggspatial::annotation_scale(location = "bl") +
  # Application d'un thème minimaliste pour une clarté visuelle
  theme_void()

```



RECAP



Ce bloc de code visualise les données sur le paludisme sur une carte. Il met en évidence les régions en fonction du nombre de tests de dépistage du paludisme et de leurs résultats. Des outils spécifiques du package `ggspatial` sont utilisés pour ajouter des éléments cartographiques, garantissant que la carte est informative.

PRACTICE



(in RMD)

Visualiser le Taux Positif par Taille et Couleur

Votre dernier défi est de créer une visualisation représentant le taux positif de paludisme pour chaque lieu. Ajustez la taille des points pour refléter le taux positif. Cette tâche testera votre compréhension de la combinaison de différents indicateurs de données sur une carte. Bonne chance !

```
# Visualisation du taux positif
ggplot() +
  _____ +
  _____ +
  _____ +
  _____
```

Dernières Réflexions

La visualisation des données géospatiales, c'est bien plus que de représenter des données sur une carte. C'est raconter une histoire ancrée dans un lieu et un espace. Cette leçon s'est plongée dans une approche narrative basée sur les couches, de l'importance d'annotations claires à l'intégration de différents types de données pour une compréhension plus approfondie.

Récapitulatif

Dans cette leçon sur l'étiquetage des cartes, nous avons exploré en profondeur les points suivants :

RECAP



- **Annotations et Étiquettes** : Des annotations claires transforment les cartes, rendant les données spatiales complexes facilement compréhensibles.
- **Techniques de Cartographie Avancées** : Nous avons exploré comment intégrer des indicateurs continus, superposer des étiquettes, mettre l'accent sur des régions et déterminer les placements optimaux des points sur les cartes.
- **Application Pratique** : En utilisant un dataset sur le paludisme, nous avons démontré la préparation des données, la visualisation et l'interprétation dans R.
- **Packages R** : Des packages tels que `ggplot2`, `sf` et `ggspatial` facilitent les visualisations géospatiales avancées.
- **Mettre en Évidence avec le Contexte** : Il est essentiel de fournir une vue d'ensemble, même lorsque l'on met l'accent sur des zones spécifiques.

Solutions

Solution pour l'exercice 1

Pour visualiser la répartition des cas de paludisme au Nigéria en 2015, suivez les instructions fournies dans l'exercice. Voici la solution complète :

```
# Construction de la carte choroplète pour 2015 avec ggplot2
ggplot(data=malaria3) +
  geom_sf(aes(fill=cases_2015/1000)) + # Colonne de données mise à jour pour
  # refléter l'année 2015
  labs(title = "Répartition des cas de paludisme au Nigéria en 2015", fill =
    "Cas pour 1000") + # Titre mis à jour pour 2015
  scale_fill_continuous(low = "blue", high = "yellow") + # Échelle de couleurs
  # modifiée pour un dégradé de bleu à jaune
  theme_void()
```

Solution pour l'exercice 2

Pour combiner la visualisation des taux d'augmentation avec les noms des états, suivez les étapes ci-dessous :

```
# Combinaison de la visualisation des taux d'augmentation avec les noms des
# états
ggplot(data = malaria3) +
  # Création d'une carte choroplète avec une couleur de remplissage basée sur
  # le taux d'augmentation
  geom_sf(aes(fill = increase_rate), color="white", size = 0.2) +

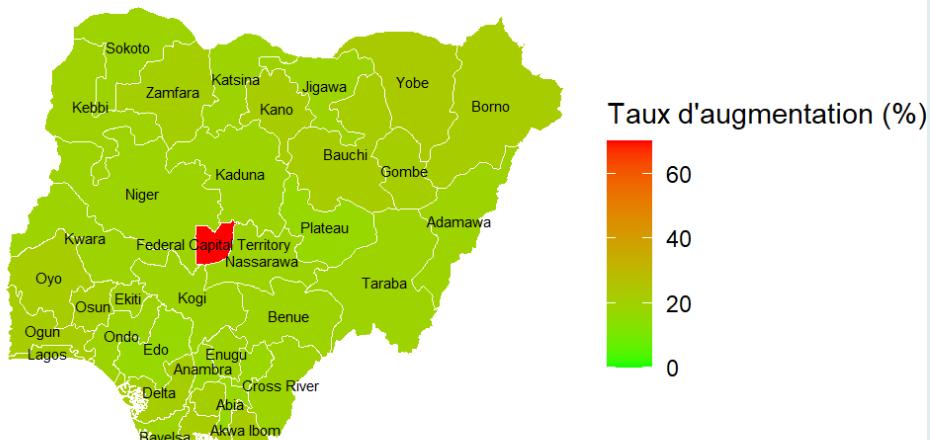
  # Étiquetage de chaque région avec son nom d'état
  geom_text(aes(x = centroid_coords[, 1],
                y = centroid_coords[, 2],
                label = state_name), size = 2, check_overlap = TRUE) +

  # Spécification de l'échelle de couleurs pour les taux d'augmentation
  scale_fill_continuous(name="Taux d'augmentation (%)",
                        limits=c(0,70),
                        low = "green",
                        high = "red",
                        breaks=c(0, 20, 40, 60)) +

  # Ajout d'un titre et d'une légende à la carte
  labs(title = "Taux d'augmentation du paludisme au Nigéria de 2015 à 2021",
       fill = "Taux d'augmentation (%)") +

  theme_void() # Application d'un thème minimaliste pour la clarté
```

Taux d'augmentation du paludisme au Nigéria de 2015 à 2021



Dans cette solution, la carte choroplèthe est créée en fonction du taux d'augmentation des cas de paludisme de 2015 à 2021. Chaque état du Nigéria est étiqueté par son nom, et le dégradé de couleurs (du vert au rouge) met en évidence l'ampleur du taux d'augmentation. La carte est enrichie d'un titre et d'une légende pour assurer la clarté et la compréhension.

Solution pour l'exercice 3

Créez une visualisation qui représente le taux de positivité :

```
# Ajout d'une colonne de taux de positivité
nigeria_pr$positive_rate <- (nigeria_pr$positive / nigeria_pr$examined) * 100

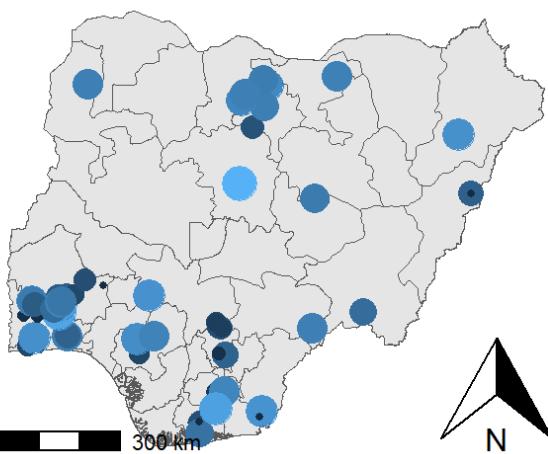
ggplot() +
  geom_sf(data = nga_adm1) +
  geom_sf(aes(size = positive_rate, color = positive_rate), data =
    nigeria_pr)+
  labs(title = "Emplacement et Taux de Positivité des Cas de Paludisme au
    Nigéria",
    subtitle = "Données du Projet Atlas du Paludisme",
    color = "Taux de Positivité (%)",
    size = "Taux de Positivité (%)")+
  xlab("Longitude")+
  ylab("Latitude")+
  ggspatial::annotation_north_arrow(location = "br")+
  ggspatial::annotation_scale(location = "bl")+
  theme_void()
```

Emplacement et Taux de Positivité des Cas de Paludisme au Données du Projet Atlas du Paludisme

Taux de Positivité (%)

- 0
- 25
- 50
- 75

Taux de Positivité (%)



Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



IMAD EL BADISY

Data Science Education Officer
Deeply interested in health data



KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement



JOY VAZ

R Developer and Instructor, the GRAPH Network
Loves doing science and teaching science

Références

1. Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis* (3e). Available at: <https://ggplot2-book.org/>

2. Hadley Wickham and Garrett Grolemund. *R for Data Science*. Available at: <https://r4ds.had.co.nz/>
3. Robin Lovelace, Jakub Nowosad, and Jannes Muenchow. *Geocomputation with R*. Available at: <https://r.geocompx.org/>

Optimisation des tables gt pour une meilleure visualisation

Introduction
Objectifs d'apprentissage
Packages
Précédemment dans la partie 1
Le jeu de données
Thèmes
Formatage des valeurs dans le tableau
Mise en forme conditionnelle
Police et texte
Bordures
Conclusion
Corrigé
Ressources externes et paquets

Introduction

La leçon précédente sur {gt} se concentrerait principalement sur les composants de la table, sa structure et comment la manipuler correctement. Cette leçon, qui présente la deuxième partie de la série {gt}, se focalisera sur l'utilisation du package pour peaufiner, styliser et personnaliser les effets visuels des tables d'une manière qui améliore la qualité et l'efficacité de vos rapports.

Allons-y.

Objectifs d'apprentissage

- 1. Utilisation de Thèmes Prédéfinis avec la Fonction opt_stylize:** Comprendre comment appliquer des styles et des thèmes de couleurs prédéfinis aux tableaux en utilisant la fonction `opt_stylize`. Explorer diverses options de style, y compris le choix du numéro de style et de la couleur, pour renforcer l'attrait visuel des tableaux
- 2. Formatage des Valeurs dans le tableau avec la Fonction data_color:** Apprendre à distinguer visuellement les valeurs dans des colonnes spécifiques en utilisant la fonction `data_color`. Cette technique est utile dans les grands tableaux pour mettre en évidence les données importantes.
- 3. Formatage Conditionnel Utilisant la Fonction tab_style:** Maîtriser le concept de formatage conditionnel dans les tableaux. Apprendre à appliquer des styles aux cellules en fonction de leurs valeurs lorsqu'elles atteignent un certain seuil.
- 4. Amélioration du Texte du tableau avec des Police et des Couleurs:** Explorer la personnalisation du texte du tableau en utilisant la fonction `gt:::tab_style()`. Apprendre à appliquer différentes polices et couleurs aux titres des tableaux, sous-

titres et autres éléments textuels pour une présentation plus attrayante visuellement.

5. Ajout de Bordures avec les Fonctions `tab_style` et `cell_borders`: Acquérir des compétences dans le tracé de bordures dans les tableaux pour souligner des zones spécifiques. Comprendre comment utiliser `tab_style` en conjonction avec `cell_borders` pour ajouter des lignes verticales et horizontales de différentes couleurs et épaisseurs.

À la fin de cette leçon, vous aurez les compétences pour styliser artistiquement vos tableaux `{gt}` selon vos préférences spécifiques, atteignant un niveau de détail similaire à ceci :

HIV Testing in Malawi				
	New tests		Previous tests	
	Positive	Negative	Positive	Negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293

Source: Malawi HIV Program

Packages

Dans cette leçon, nous utiliserons les packages suivants :

- `{gt}` : pour créer nos tableaux.
- `{dplyr}`, `{tidyverse}` et `{purrr}` : pour manipuler les données et automatiser les fonctions.
- `{janitor}` : pour un nettoyage rapide des noms de données.
- `{Palettes}`, `{ggsci}` : pour appeler des palettes prédéfinies.

```
pacman::p_load(tidyverse, janitor, gt, here)
```

Précédemment dans la partie 1

::: recap Dans la leçon précédente sur {gt}, nous avons eu l'opportunité de :

- Découvrir les données de prévalence du VIH au Malawi.
- Découvrir la grammaire des tableaux et le package {gt}.
- Créer un tableau simple.
- Ajouter des détails comme un titre et une note de bas de page au tableau.
- Regrouper des colonnes dans des en-têtes de groupe.
- Créer des lignes récapitulatives.

Sum of HIV Tests in Malawi

from Q1 2019 to Q4 2019

	New tests		Previous tests	
	Positive	Negative	Positive	Negative
Central Region				
2019 Q1	2004	123018	3682	2562
2019 Q2	1913	116443	3603	1839
2019 Q3	1916	127799	4002	2645
2019 Q4	1691	124728	3754	1052
sum	7524.00	491988.00	15041.00	8098.00
mean	1881.00	122997.00	3760.25	2024.50
Northern Region				
2019 Q1	664	36196	1197	675
2019 Q2	582	35315	1084	590
2019 Q3	570	36850	1191	542
2019 Q4	519	34322	1132	346
sum	2335.00	142683.00	4604.00	2153.00
mean	583.75	35670.75	1151.00	538.25
Southern Region				
2019 Q1	3531	125480	9937	3358
2019 Q2	3637	130491	10414	3176
2019 Q3	3421	135880	10606	3304
2019 Q4	3436	132572	10814	4895

Le jeu de données

Dans cette leçon, nous utiliserons les mêmes données que dans la leçon précédente. Vous pouvez revenir en arrière pour une description détaillée des données et du processus de préparation que nous avons effectué.

Voici les détails complets des colonnes que nous utiliserons :



- **region** : La région géographique ou la zone où les données ont été collectées ou sont analysées.
- **period** : Une période spécifique associée aux données, souvent utilisée pour l'analyse temporelle.
- **previous_negative** : Le nombre d'individus avec un résultat de test négatif précédent.
- **previous_positive** : Le nombre d'individus avec un résultat de test positif précédent.
- **new_negative** : Le nombre de nouveaux cas diagnostiqués avec un résultat négatif.
- **new_positive** : Le nombre de nouveaux cas diagnostiqués avec un résultat positif.

Mais pour les besoins de cette leçon, nous utiliserons directement les tableaux, ceci est le tableau que nous avons créé avec les bons en-têtes de colonnes et étiquettes, nous baserons le reste de notre leçon sur celui-ci en particulier.

```
hiv_malawi_summary <-  
read_rds(here::here("data/clean/malawi_hiv_summary_l2_t11.rds"))  
  
hiv_malawi_summary
```

HIV Testing in Malawi
Q1 to Q2 2019

	New tests		Previous tests	
	Positive	Negative	Positive	Negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293

Source: Malawi HIV Program

Thèmes

Puisque l'objectif de cette leçon est principalement le style, commençons par utiliser un thème prédéfini pour ajouter plus de visuels et de couleurs au tableau et à ses composants. Pour ce faire, nous utilisons la fonction `opt_stylize`. La fonction contient plusieurs styles prédéfinis et peut également accepter une couleur. Dans notre cas, nous avons choisi d'aller avec le style numéro 6 et la couleur 'gris', mais vous pouvez régler ces arguments selon vos préférences.

```
t1 <- hiv_malawi_summary %>%
  opt_stylize(
    style = 1,
    color = 'cyan'
  ) %>%
  tab_options(
    stub.background.color = '#F4F4F4',
  )

t1
```

HIV Testing in Malawi				
	New tests		Previous tests	
	Positive	Negative	Positive	Negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293

Source: Malawi HIV Program



Pour des thèmes et un style plus sophistiqués, vous pouvez vous référer à la fonction `tab_options` (documentation [ici](#)) qui est essentiellement l'équivalent de la fonction `theme` dans `ggplot2`. Cette fonction contient des arguments et des options sur chaque couche et composant du tableau. Pour les besoins de cette leçon, nous n'irons pas plus loin dans ce domaine.

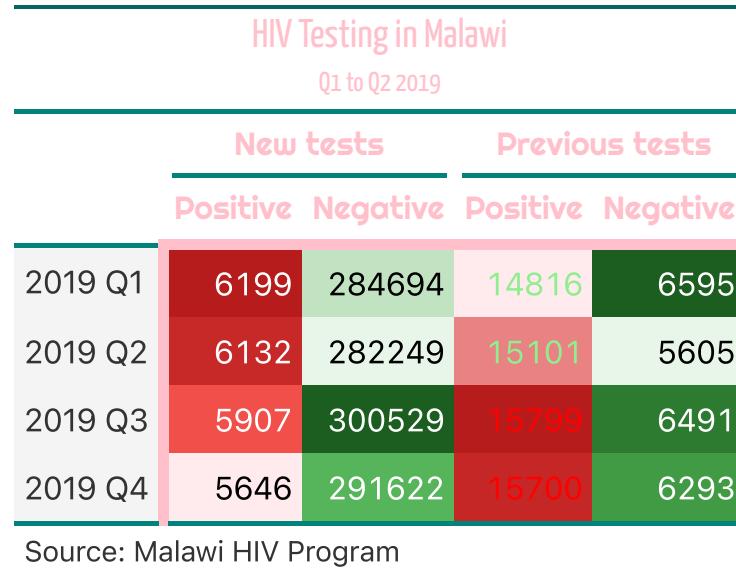
Formatage des valeurs dans le tableau

Ne serait-il pas utile de visualiser en couleurs la différence entre les valeurs dans une colonne spécifique ? Dans de nombreux rapports, ce genre de tableaux est très utile, surtout si le nombre de lignes est assez grand. Faisons cela pour notre tableau de sorte que la colonne `new_positive` soit formatée en rouge.

Nous pouvons le faire au moyen de la fonction `data_color` pour laquelle nous devons spécifier deux arguments, `columns` (c'est-à-dire dans quelle colonne ce style sera appliqué ?) et `palette` comme la palette de couleurs que nous avons l'intention d'utiliser.

```
t2 <- t1 %>%
  data_color(
    columns = new_positive, # la colonne ou les colonnes comme nous le
verrons plus tard
    palette = "ggsci::red_material" # la palette du package ggsci.
  )

t2
```


SIDE NOTE


`ggsci::red_material` n'est pas la seule palette que nous pouvons utiliser, en fait, il y a des centaines de palettes conçues pour être utilisées dans R. Vous pouvez en trouver beaucoup plus dans le package `palettereer` dans la documentation ici, ou dans la documentation officielle `data_color` ici.

Nous pouvons également faire cela pour la colonne `previous_negative`. Nous pouvons utiliser un type différent de palette, j'utilise pour ce cas la palette verte du même package : `ggsci::green_material`, la palette que vous choisissez est une question de commodité et de goût personnel, vous pouvez en savoir plus à ce sujet si vous vous référez à la note latérale ci-dessus.

```
t2 %>%
  data_color(
    columns = previous_negative,
    palette = "ggsci::green_material"
  )
```

HIV Testing in Malawi				
	New tests		Previous tests	
	Positive	Negative	Positive	Negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293

Source: Malawi HIV Program

De même, nous pouvons également colorer plusieurs colonnes à la fois, par exemple nous pouvons styliser les colonnes avec des cas positifs en rouge et celles avec des cas négatifs en vert. Pour ce faire, nous devons écrire *deux* instructions `data_color`, une pour chaque style de couleur :

```
t4 <- t1 %>%
  data_color(
    columns = ends_with("positive"), # sélectionnant les colonnes se terminant par le mot positif
    palette = "ggsci::red_material" # palette rouge
  ) %>%
  data_color(
    columns = ends_with("negative"), # sélectionnant les colonnes se terminant par le mot négatif
    palette = "ggsci::green_material" # palette verte
  )
t4
```

HIV Testing in Malawi				
	New tests		Previous tests	
	Positive	Negative	Positive	Negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293

Source: Malawi HIV Program

REMINDER



Rappelez-vous que dans la leçon précédente nous avons utilisé les fonctions `tidyselect` pour sélectionner les colonnes, dans le code ci-dessus nous avons utilisé la fonction `ends_with` pour sélectionner les colonnes se terminant soit par le mot 'négatif' soit par 'positif', ce qui est parfait pour l'objectif de notre tableau.

Encore une fois, les étiquettes des colonnes dans le tableau `{gt}` et les noms réels des colonnes dans le `data.frame` peuvent être différents, dans notre cas nous nous référons aux noms dans les données.

Mise en forme conditionnelle

Nous pouvons également configurer la table pour changer conditionnellement le style d'une cellule en fonction de sa valeur. Dans notre cas, nous souhaitons mettre en évidence les valeurs de la colonne `previous_positive` selon un seuil (la valeur 15700). Les valeurs supérieures ou égales au seuil doivent être en vert.

Pour ce faire, nous utilisons la fonction `tab_style` où nous spécifions deux arguments :

- `style` : où nous spécifions la couleur dans la fonction `cell_text` puisque nous avons l'intention de manipuler le texte à l'intérieur des cellules.
- `location` : où nous spécifions les colonnes et les lignes de notre manipulation dans `cells_body` puisque ces cellules sont dans le corps principal du tableau.

Utilisons le tableau t2 comme exemple :

```
t5 <- t2 %>%
  tab_style(
    style = cell_text(
      color = "red",
    ),
    locations = cells_body(
      columns = previous_positive,
      rows = previous_positive >= 15700
    )
  )
t5
```

HIV Testing in Malawi				
Q1 to Q2 2019				
	New tests		Previous tests	
	Positive	Negative	Positive	Negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293

Source: Malawi HIV Program

Dans le code ci-dessus, la condition sur laquelle le style sera appliqué est énoncée dans :

```
locations = cells_body(columns = previous_positive, rows =
previous_positive >= 15700 )
```

Notez également que nous pouvons passer plus d'arguments à la fonction `cell_text`, tels que la taille et la police des cellules que nous avons l'intention de styliser.

Que faire si nous voulons avoir une condition à double sens sur le même seuil ? Peut-on avoir des cellules avec des valeurs supérieures ou égales au seuil stylisées en vert, et simultanément d'autres cellules avec des valeurs inférieures au seuil stylisées en... cyan ?

Nous le pouvons absolument, nous avons déjà fait la première partie (dans le morceau de code précédent), nous devons juste ajouter une seconde condition de manière similaire mais dans une déclaration `tab_style` différente :

```

t6 <- t5 %>%
  tab_style(
    style = cell_text(
      color = 'cyan'
    ),
    location = cells_body(
      columns = 'previous_positive',
      rows = previous_positive < 15700
    )
  )

```

t6

HIV Testing in Malawi				
	New tests		Previous tests	
	Positive	Negative	Positive	Negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293

Source: Malawi HIV Program

Question 1 : Mise en forme conditionnelle Pour mettre en évidence (en jaune) les lignes dans un tableau **{gt}** où la colonne “hiv_positive” dépasse 1 000, quel extrait de code R devriez-vous utiliser ?

A.

```

data %>%
  gt() %>%
  tab_style(
    style = cells_body(),
    columns = "Sales",
    conditions = style_number(Sales > 1000, background = "yellow")
  )

```

B.

```

data %>%
  gt() %>%
  tab_style(
    style = cells_data(columns = "Sales"),
    conditions = style_number(Sales > 1000, background = "yellow")
  )

```

C.

```

data %>%
  gt() %>%
  tab_style(
    style = cell_fill(
      color = "yellow"
    ),
    locations = cells_body(
      columns = "hiv_positive",
      rows = hiv_positive > 1000
    )
  )

```

D.

```

data %>%
  gt() %>%
  tab_style(
    style = cells_data(columns = "Sales"),
    conditions = style_text(Sales > 1000, background = "yellow")
  )

```

Question 2 : Coloration des cellules

En utilisant le jeu de données **hiv_malawi**, créez un tableau **{gt}** qui affiche le total (**somme**) des cas “*new_positive*” pour chaque “*region*”. Mettez en évidence les cellules avec des valeurs de plus de 50 cas en *rouge* et les cellules avec des valeurs inférieures ou égales à 50 en *vert*. Complétez les parties manquantes (_____) de ce code pour y parvenir.

```

# Calculez le résumé de total_new_pos``{r eval=FALSE, echo=TRUE}
# Calculez le résumé de total_new_pos
total_summary <- hiv_malawi %>%
  group_by(_____) %>%
  summarize(total_new_positive = _____)

# Créez un tableau gt et appliquez la coloration des cellules
summary_table <- total_summary %>%
  gt() %>%
  tab_style(
    style = cell_fill(color = "red"),
    locations = _____(
      columns = "new_positive",
      rows = _____
    )
  ) %>%
  tab_style(
    style = _____,
    locations = cells_body(
      columns = "new_positive",
      _____ new_positive <= 50
    )
  )

```

Police et texte

Maintenant, nous allons améliorer l'attrait visuel du texte de notre tableau. Pour ce faire, nous utiliserons à nouveau la fonction `gt::tab_style()`.

Modifions la police et la couleur du titre et du sous-titre. Nous sélectionnerons la police Yanone Kaffeesatz de Google Fonts, une ressource offrant un vaste choix de polices qui peuvent ajouter une touche unique à votre tableau, au-delà des options standard d'Excel.

Pour appliquer ces changements, nous configurerons la fonction `gt::tab_style()` comme suit :

- L'argument `style` est assigné à la fonction `cell_text()`, qui contient deux autres arguments :
 - `font` est assigné à la fonction `google_font()` avec le nom de notre police choisie.
 - `color` est défini sur un code couleur hexadécimal qui correspond à la couleur de texte désirée.
- L'argument `locations` est assigné à la fonction `cells_title()` :

- Nous spécifions `title` et `subtitle` dans l'argument `groups` en utilisant la notation vectorielle `c(...)`.

Pour modifier spécifiquement le titre ou le sous-titre, vous pouvez utiliser `locations = cells_title(groups = "title")` ou `locations = cells_title(groups = "subtitle")`, respectivement, sans avoir besoin de `c(...)`.

SIDE NOTE



Utiliser des listes pour passer des arguments dans gt : Les listes en R sont une partie intégrante du langage et sont extrêmement polyvalentes. Une liste peut contenir des éléments de différents types (nombres, chaînes, vecteurs, et même d'autres listes) et chaque élément peut être accédé par son indice. Dans le contexte de notre tableau {gt}, nous utilisons des listes pour regrouper ensemble des propriétés de style (avec l'argument `style`) et pour spécifier plusieurs emplacements dans le tableau où ces styles doivent être appliqués (avec l'argument `locations`).

Utiliser des Codes Couleurs Hexadécimaux : Les couleurs dans de nombreux langages de programmation, y compris R, peuvent être spécifiées à l'aide de codes couleurs hexadécimaux. Ces codes commencent par un symbole dièse (#) et sont suivis de six chiffres hexadécimaux. Les deux premiers chiffres représentent la composante rouge, les deux suivants la composante verte, et les deux derniers la composante bleue. Ainsi, lorsque nous définissons `color = "#00353f"`, nous spécifions une couleur qui n'a pas de rouge, un peu de vert, et une bonne quantité de bleu, ce qui résulte en une couleur bleu profond. Cela nous permet de contrôler précisément les couleurs que nous utilisons dans nos tableaux.

```
t7 <- t4 %>%
  tab_style(
    style = cell_text(
      font = google_font(name = 'Yanone Kaffeesatz'),
      color = "#00353f"
    ),
    locations = cells_title(groups = c("title", "subtitle"))
  )
t7
```

HIV Testing in Malawi				
	New tests		Previous tests	
	Positive	Negative	Positive	Negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293

Source: Malawi HIV Program

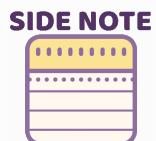
Nous pouvons étendre notre personnalisation pour inclure les étiquettes des colonnes, des en-têtes de groupe et des lignes d'attente, ainsi que la note source. Dans l'argument `locations`, nous fournirons une liste indiquant les emplacements spécifiques pour ces changements. Pour une compréhension complète des emplacements, veuillez vous référer à l'Annexe (Liste 1).

```
t8 <- t7 %>%
  tab_style(
    style = list(
      cell_text(
        font = google_font(name = "Montserrat"),
        color = "#00353f"
      )
    ),
    locations = list(
      cells_column_labels(columns = everything(), # sélectionner
chaque colonne
        cells_column_spanners(spanners = everything()), # sélectionner
tous les en-têtes de groupe
        cells_source_notes(),
        cells_stub()
      )
    )
  )
t8
```

HIV Testing in Malawi				
	New tests		Previous tests	
	Positive	Negative	Positive	Negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293

Source: Malawi HIV Program

Si vous souhaitez changer la couleur de fond du titre, vous pouvez le faire en ajustant l'argument `locations` pour pointer sur `cells_title(groups = "title")`. Voici comment vous pourriez le faire :



```
t9 <- t7 %>%
  tab_style(
    style = cell_fill(color = "#ffffff"),
    locations = cells_title(groups = "title")
  )
t9
```

Dans ce code, `cell_fill(color = "#ffffff")` change la couleur de fond en blanc, et `locations = cells_title(groups = "title")` applique ce changement spécifiquement au titre du tableau.

Question 3 : Polices et Texte Quel extrait de code R permet de changer la taille de la police du texte de bas de page dans un tableau `{gt}` ?

A.

```
data %>%
  gt() %>%
  tab_header(font.size = px(16))
```

B.

```
data %>%
  gt() %>%
  tab_style(
    style = cell_text(
      size = 16
    ),
    locations = cells_footnotes()
  )
```

C.

```
data %>%
  gt() %>%
  tab_style(
    style = cells_header(),
    css = "font-size: 16px;"
  )
```

D.

```
data %>%
  gt() %>%
  tab_style(
    style = cells_header(),
    css = "font-size: 16px;"
  )
```

Bordures

Dans {gt}, il est également possible de dessiner des bordures dans les tableaux pour aider l'utilisateur final à se concentrer sur une zone spécifique du tableau. Pour ajouter des bordures à un tableau {gt}, nous utiliserons, encore une fois, la fonction `tab_style` et, encore une fois, spécifierons l'argument `style` et `locations`. La seule différence maintenant est que nous utiliserons la fonction d'assistance `cell_borders` et l'assignerons à l'argument de `style`. Voici comment :

Ajoutons d'abord une ligne verticale :

```
t10 <- t8 %>%
  tab_style(
    style = cell_borders( # nous ajoutons une bordure
      sides = "left",     # à gauche de l'emplacement sélectionné
      color = "#45785e",   # avec une couleur vert foncé
      weight = px(5)       # et cinq pixels d'épaisseur
    ),
    locations = cells_body(columns = 2) # ajouter cette ligne de
bordure à gauche de la colonne 2
  )
t10
```

HIV Testing in Malawi
Q1 to Q2 2019

	New tests		Previous tests	
	Positive	Negative	Positive	Negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293

Source: Malawi HIV Program

Maintenant, ajoutons une autre ligne de bordure horizontale rose :

```
t11 <- t10 %>%
  tab_style(
    style = cell_borders( # nous ajoutons une bordure
      sides = "left",     # à gauche de l'emplacement sélectionné
      color = "#45785e",   # avec une couleur vert foncé
      weight = px(5)       # et cinq pixels d'épaisseur
    ),
    locations = list(
      cells_column_labels(columns = everything()), # ajouter cette
ligne de bordure en bas des étiquettes des colonnes
      cells_stubhead()                           # et au stubhead
    )
  )

t11
```

HIV Testing in Malawi				
	New tests		Previous tests	
	Positive	Negative	Positive	Negative
2019 Q1	6199	284694	14816	6595
2019 Q2	6132	282249	15101	5605
2019 Q3	5907	300529	15799	6491
2019 Q4	5646	291622	15700	6293

Source: Malawi HIV Program

Question 4 : Bordures Pour ajouter une bordure pleine autour de la totalité du tableau `{gt}`, quel extrait de code R devriez-vous utiliser ?

Indice : nous pouvons utiliser une fonction qui définit des options pour l'ensemble du tableau, tout comme la fonction `theme` pour le package `ggplot`.

A.

CHALLENGE



```
donnees %>%
  gt() %>%
  tab_options(table.border.top.style = "solid")
```

B.

```
donnees %>%
  gt() %>%
  tab_options(table.border.style = "solid")
```

C.

```
donnees %>%
  gt() %>%
  tab_style(
    style = cells_table(),
    css = "border: 1px solid black;"
  )
```



D.

```
donnees %>%
  gt() %>%
  tab_style(
    style = cells_body(),
    css = "border: 1px solid black;"
  )
```

Conclusion

Pour conclure notre série, nous avons commencé par un récapitulatif complet de la première partie, posant les bases pour des techniques avancées de stylisation de tableaux. Nous avons exploré l'utilisation de la fonction `opt_stylize()` pour appliquer de manière élégante des styles et des couleurs prédéfinis, améliorant l'attrait visuel de nos tableaux. Une partie clé de notre discussion a introduit la fonction `data_color`, un outil puissant pour appliquer des distinctions codées par couleur aux valeurs du tableau, ce qui aide à une évaluation rapide des données et au suivi visuel. Nous avons ensuite abordé le formatage conditionnel, en utilisant la fonction `tab_style` pour modifier dynamiquement les styles des cellules en fonction des données qu'elles contiennent, une étape qui attire l'attention sur les métriques et les tendances critiques. Au-delà de la fonctionnalité, nous nous sommes concentrés sur l'esthétique avec `gt::tab_style()`, montrant comment des polices uniques de Google Fonts peuvent améliorer considérablement la lisibilité et l'apparence du texte dans les tableaux. Enfin, nous avons couvert l'ajout stratégique de bordures en utilisant la fonction `tab_style` en conjonction avec la fonction d'assistance `cell_borders`, guidant l'attention des utilisateurs sur des zones spécifiques du tableau et améliorant l'interprétation globale des données. Chacune de ces techniques contribue à une présentation de tableau plus engageante et informative, garantissant que les données ne sont pas seulement accessibles mais également captivantes.

Corrigé

1. Question 1 : Formatage Conditionnel

◦ C

2. Question 2 : Coloration de Remplissage des Cellules

```
# Les solutions sont là où les lignes sont numérotées

# Calculer le résumé total_new_pos
total_summary <- hiv_malawi %>%
  group_by(region) %>% ##1
  summarize(total_new_positive = new_positive) ##2

# Créer un tableau gt et appliquer la coloration des cellules
summary_table <- total_summary %>%
  gt() %>% ##3
  tab_style(
    style = cell_fill(color = "red"),
    locations = cells_body( ##4
      columns = "new_positive",
      rows = new_positive >= 50 ##5
    )
  ) %>%
  tab_style(
    style = cell_fill(color = "green"), ##6
    locations = cells_body(
      columns = "new_positive",
      rows = new_positive < 50 ##7
    )
  )
)
```

3. Question 3 : Polices et Texte

◦ B

4. Question 4 : Bordures

◦ B

Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



BENNOUR HSIN

Data Science Education Officer
Data Visualization enthusiast



JOY VAZ

R Developer and Instructor, the GRAPH Network

Loves doing science and teaching science



GUY WAFEU

R Instructor and Public Health Physician
Committed to improving the quality of data analysis

Ressources externes et paquets

- Le guide définitif de {gt} par Tom Mock : <https://themockup.blog/static/resources/gt-cookbook.html#introduction>
- L'article sur la Grammaire des Tableaux : <https://themockup.blog/posts/2020-05-16-gt-a-grammar-of-tables/#add-titles>
- Page de documentation officielle de {gt} : <https://gt.rstudio.com/articles/intro-creating-gt-tables.html>
- Créer une table HTML impressionnante avec le livre knitr::kable et kableExtra par Hao Zhu : https://cran.r-project.org/web/packages/kableExtra/vignettes/awesome_table_in_html.html#Overview

Annexe

Le package {gt} en R fournit une variété de fonctions pour spécifier les emplacements dans un tableau où certains styles ou mises en forme doivent être appliqués. Voici certaines d'entre elles :

- `cells_body()` : Cette fonction cible les cellules à l'intérieur du corps du tableau. Vous pouvez spécifier davantage les lignes et les colonnes pour cibler un sous-ensemble du corps.
- `cells_column_labels()` : Cette fonction cible les cellules qui contiennent les étiquettes des colonnes.
- `cells_column_spanners()` : Cette fonction cible les cellules qui s'étendent sur plusieurs colonnes.
- `cells_footnotes()` : Cette fonction cible les cellules qui contiennent des notes de bas de page.

- `cells_grand_summary()` : Cette fonction cible les cellules qui contiennent les lignes de grand résumé.
- `cells_group()` : Cette fonction cible les cellules qui contiennent les lignes d'étiquettes de groupe.
- `cells_row_groups()` : Cette fonction cible les cellules qui contiennent les lignes d'étiquettes de groupes de lignes.
- `cells_source_notes()` : Cette fonction cible les cellules qui contiennent les notes sources.
- `cells_stub()` : Cette fonction cible les cellules dans le talon du tableau (les étiquettes dans la première colonne du tableau).
- `cells_stubhead()` : Cette fonction cible la cellule qui contient l'en-tête du talon.
- `cells_stub_summary()` : Cette fonction cible les cellules qui contiennent les lignes de résumé du talon.
- `cells_title()` : Cette fonction cible les cellules qui contiennent le titre et le sous-titre du tableau.
- `cells_summary()` : Cette fonction cible les cellules qui contiennent les lignes de résumé.

Ces fonctions peuvent être utilisées dans l'argument `locations` de la fonction `tab_style()` pour appliquer des styles spécifiques à différentes parties du tableau.

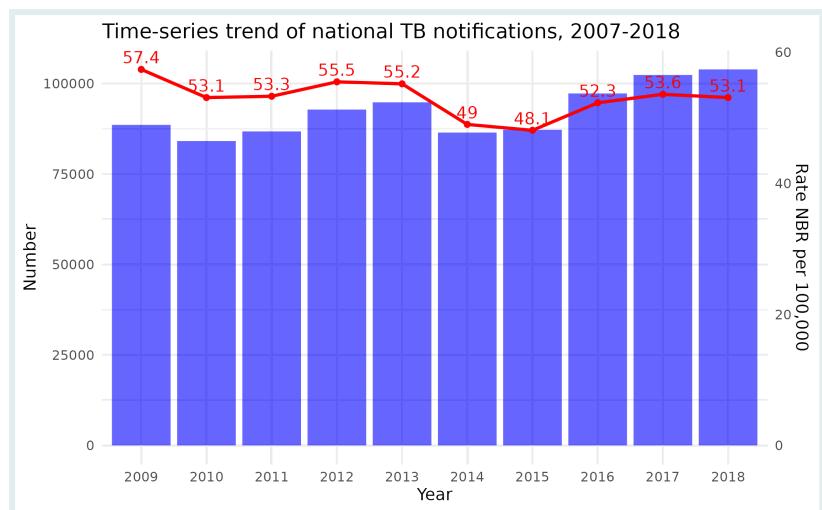
...

Visualisation de Séries Temporelles Épidémiologiques

Introduction
Objectifs d'Apprentissage
Packages
Introduction aux Graphiques Linéaires pour les Données de Séries Temporelles
Préparation des Données : Agrégation et Pivotement
Un Graphique Linéaire Groupé Basique
Améliorations Esthétiques des Graphiques Linéaires
Réduction de la Fréquence des Étiquettes
Alternance des Étiquettes
<code>ggrepel::geom_text_repel()</code>
Personnalisation de la Palette de Couleurs
Ajout d'Annotations au Graphique
Représentation des Intervalles de Confiance avec <code>geom_ribbon()</code>
Lissage des Tendances Bruyantes
Création d'un Tableau d'Incidence à partir d'une Liste de Cas
Lissage avec <code>geom_smooth()</code>
Lissage par Agrégation
Lissage avec des Moyennes Mobiles
Axes Secondaires
Comprendre le Concept d'un Axe Y Secondaire
Création d'un Graphique avec un Axe Y Secondaire
Conclusion !
Solutions
Contributeurs

Introduction

En analysant des données de séries temporelles, des données collectées à intervalles réguliers, les épidémiologistes peuvent générer des prévisions, informer les décisions politiques et, finalement, améliorer les mesures de prévention et de contrôle des maladies. Dans cette leçon, nous explorons comment créer et interpréter des séries temporelles épidémiologiques, en utilisant R pour visualiser les données de manière efficace.



Objectifs d'Apprentissage

À la fin de cette leçon, vous serez capable de :

- Remodeler les données de séries temporelles pour la représentation graphique avec `pivot_longer()`
- Créer des graphiques linéaires dans `ggplot2` en associant le temps à l'axe x et les valeurs à l'axe y
- Améliorer l'esthétique des graphiques linéaires avec des techniques telles que les étiquettes personnalisées, les palettes de couleurs, les annotations
- Visualiser les intervalles de confiance avec `geom_ribbon()`
- Mettre en évidence les modèles dans des données bruyantes en utilisant le lissage et l'agrégation
- Comparer des séries temporelles avec des échelles distinctes en utilisant des axes doubles et `sec_axis()`

Packages

Installez et chargez les paquets nécessaires avec le bloc de code suivant :

```
# Charger les paquets
if(!require(pacman)) install.packages("pacman")
pacman::p_load(dplyr, ggplot2, tidyr, lubridate, outbreaks, scales, ggrepel,
               ggthemes, zoo, here)
options(scipen=999)
```





Définir `options(scipen = 999)` empêche l'utilisation de la notation scientifique dans nos graphiques, rendant les nombres longs plus faciles à lire et à interpréter.

Introduction aux Graphiques Linéaires pour les Données de Séries Temporelles

Pour commencer à visualiser les données de séries temporelles, nous allons examiner la dynamique des notifications de tuberculose (TB) en Australie au fil du temps, en comparant les notifications dans les zones urbaines et rurales. Le jeu de données source est accessible [ici](#)

Notifications est un terme technique pour le nombre de cas d'une maladie qui sont signalés aux autorités de santé publique.

Préparation des Données : Agrégation et Pivotement

Commençons par charger et inspecter les données :

```
tb_data_aus <- read_csv(here::here("data/aus_tb_notifs.csv"))
head(tb_data_aus)
```

```
## # A tibble: 6 × 3
##   period rural urban
##   <chr>   <dbl> <dbl>
## 1 1993Q1     6    51
## 2 1993Q2    11    52
## 3 1993Q3    13    67
## 4 1993Q4    14    82
## 5 1994Q1    16    63
## 6 1994Q2    15    65
```

Ce jeu de données comprend les colonnes `period` (temps en format trimestriel, par exemple, '1993Q1'), `rural` (cas dans les zones rurales) et `urban` (cas dans les zones urbaines).

Nous aimerais visualiser le nombre de notifications annuelles de TB dans les zones urbaines et rurales, mais les données sont actuellement au format trimestriel. Ainsi, nous devons agréger les données par année.

Commençons par extraire l'année de la colonne `period`. Nous faisons cela en utilisant la fonction `str_sub()` du paquet `stringr` :

```
tb_data_aus %>%
  mutate(year = str_sub(period, 1, 4)) %>%
  # reconvertir en numérique
  mutate(year = as.numeric(year))
```

```
## # A tibble: 120 × 4
##   period rural urban year
##   <chr>   <dbl> <dbl> <dbl>
## 1 1993Q1     6    51  1993
## 2 1993Q2    11    52  1993
## 3 1993Q3    13    67  1993
## 4 1993Q4    14    82  1993
## 5 1994Q1    16    63  1994
## 6 1994Q2    15    65  1994
## 7 1994Q3    18    60  1994
## 8 1994Q4    25    71  1994
## 9 1995Q1     7    77  1995
## 10 1995Q2    9    52  1995
## # i 110 more rows
```

La fonction `str_sub()` prend trois arguments : la chaîne à partir de laquelle nous voulons extraire, la position de départ et la position de fin. Dans ce cas, nous voulons extraire les quatre premiers caractères de la colonne `period`, qui correspondent à l'année.

Maintenant, agrégeons les données par année. Nous pouvons le faire en utilisant les fonctions `group_by()` et `summarise()` :

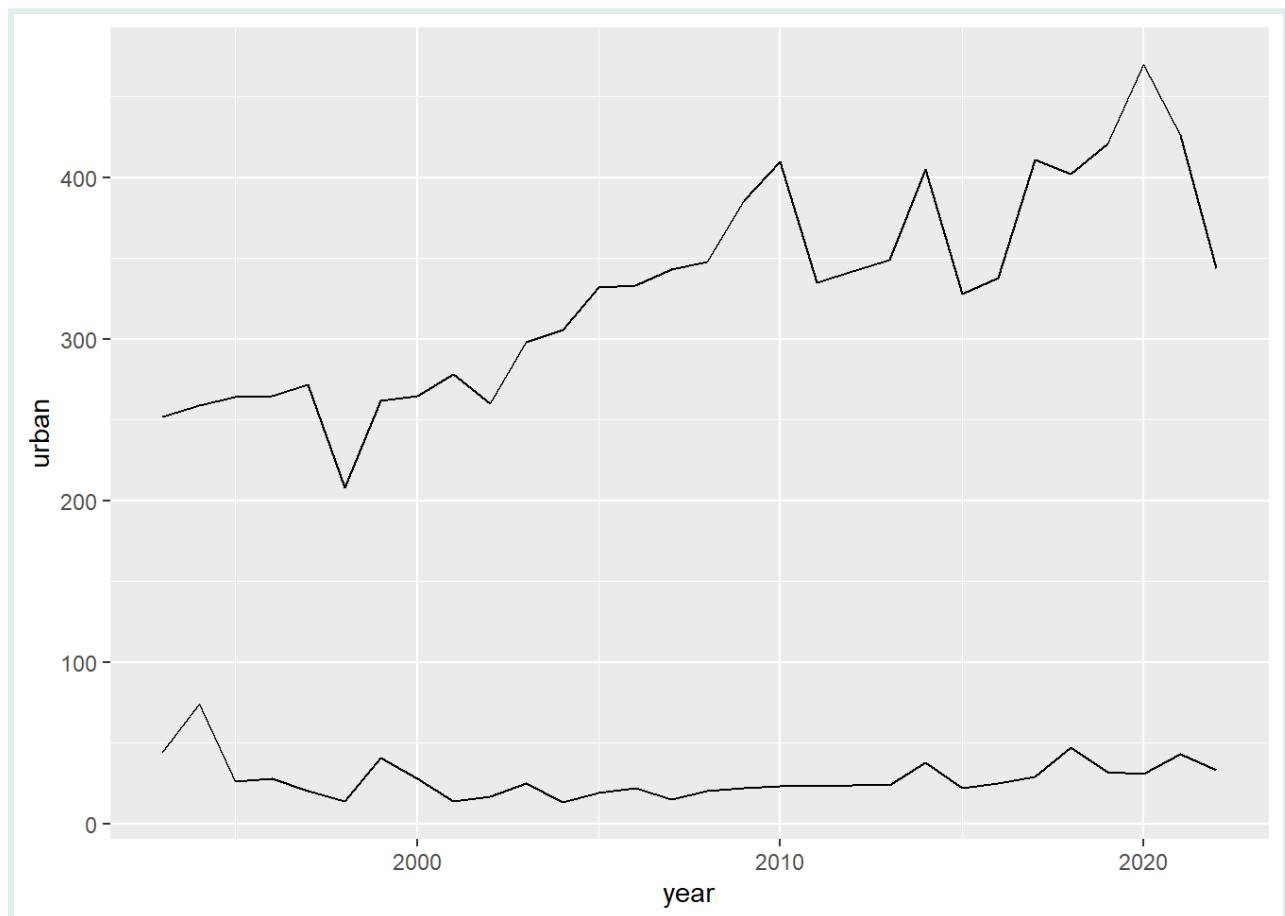
```
annual_data_aus <- tb_data_aus %>%
  mutate(year = str_sub(period, 1, 4)) %>%
  mutate(year = as.numeric(year)) %>%
  # grouper par année
  group_by(year) %>%
  # sommer le nombre de cas chaque année
  summarise(rural = sum(rural),
            urban = sum(urban))
annual_data_aus
```

```
## # A tibble: 30 × 3
##   year rural urban
##   <dbl> <dbl> <dbl>
## 1 1993     44    252
## 2 1994     74    259
## 3 1995     26    264
## 4 1996     28    265
## 5 1997     20    272
## 6 1998     14    208
## 7 1999     41    262
## 8 2000     28    265
## 9 2001     14    278
```

```
## 10 2002 17 260
## # i 20 more rows
```

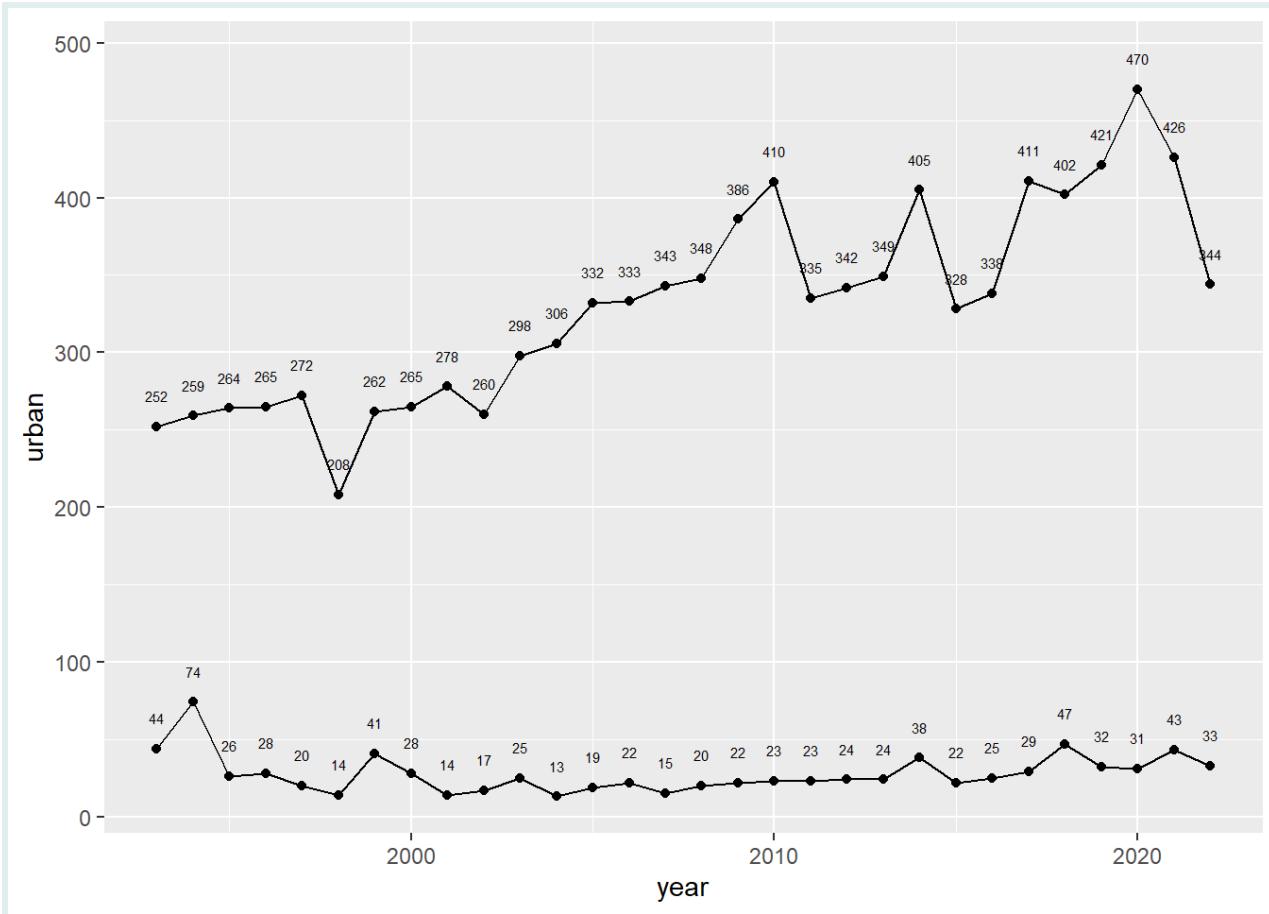
Maintenant que nous semblons avoir les données dans le format souhaité, faisons un premier graphique linéaire :

```
ggplot(annual_data_aus, aes(x = year)) +
  geom_line(aes(y = urban, couleur = "Urbain")) +
  geom_line(aes(y = rural, couleur = "Rural"))
```



C'est un graphique informatif, cependant, il y a une certaine redondance inutile dans le code, bien que vous ne le réalisiez peut-être pas encore. Cela deviendra plus clair si nous essayons d'ajouter d'autres géométries, comme des points ou du texte :

```
ggplot(annual_data_aus, aes(x = year)) +
  geom_line(aes(y = urban, couleur = "Urbain")) +
  geom_line(aes(y = rural, couleur = "Rural")) +
  geom_point(aes(y = urban, couleur = "Urbain")) +
  geom_point(aes(y = rural, couleur = "Rural")) +
  geom_text(aes(y = urban, label = urban), size = 2, nudge_y = 20) +
  geom_text(aes(y = rural, label = rural), size = 2, nudge_y = 20)
```



Comme vous pouvez le voir, nous devons répéter les mêmes lignes de code pour chaque géométrie. Cela est non seulement fastidieux, mais rend également le code plus difficile à lire et à interpréter. Si nous avions plus de deux catégories, comme cela arrive souvent, cela serait encore plus encombrant.

Heureusement, il existe une meilleure façon de faire. Nous pouvons utiliser la fonction `pivot_longer()` du package `{tidyverse}` pour remodeler les données dans un format plus adapté pour la représentation graphique :

```
# Utilisation de `pivot_longer` de tidyverse pour remodeler les données
annual_data_aus %>%
  pivot_longer(cols = c("urban", "rural"))
```

```
## # A tibble: 60 × 3
##       year name  value
##   <dbl> <chr> <dbl>
## 1 1993 urban  252
## 2 1993 rural   44
## 3 1994 urban  259
## 4 1994 rural   74
## 5 1995 urban  264
## 6 1995 rural   26
## 7 1996 urban  265
## 8 1996 rural   28
## # ... with 52 more rows
```

```

##   9 1997 urban    272
## 10 1997 rural     20
## # i 50 more rows

```

Le code ci-dessus a converti les données d'un format "large" à un format "long". C'est un format plus adapté pour la représentation graphique, car il nous permet d'associer une colonne spécifique à l'esthétique `couleur`.

Avant de représenter ce jeu de données long, renommons les colonnes pour les rendre plus informatives :

```

aus_long <- annual_data_aus %>%
  pivot_longer(cols = c("urban", "rural")) %>%
  rename(region = name, cases = value)

```

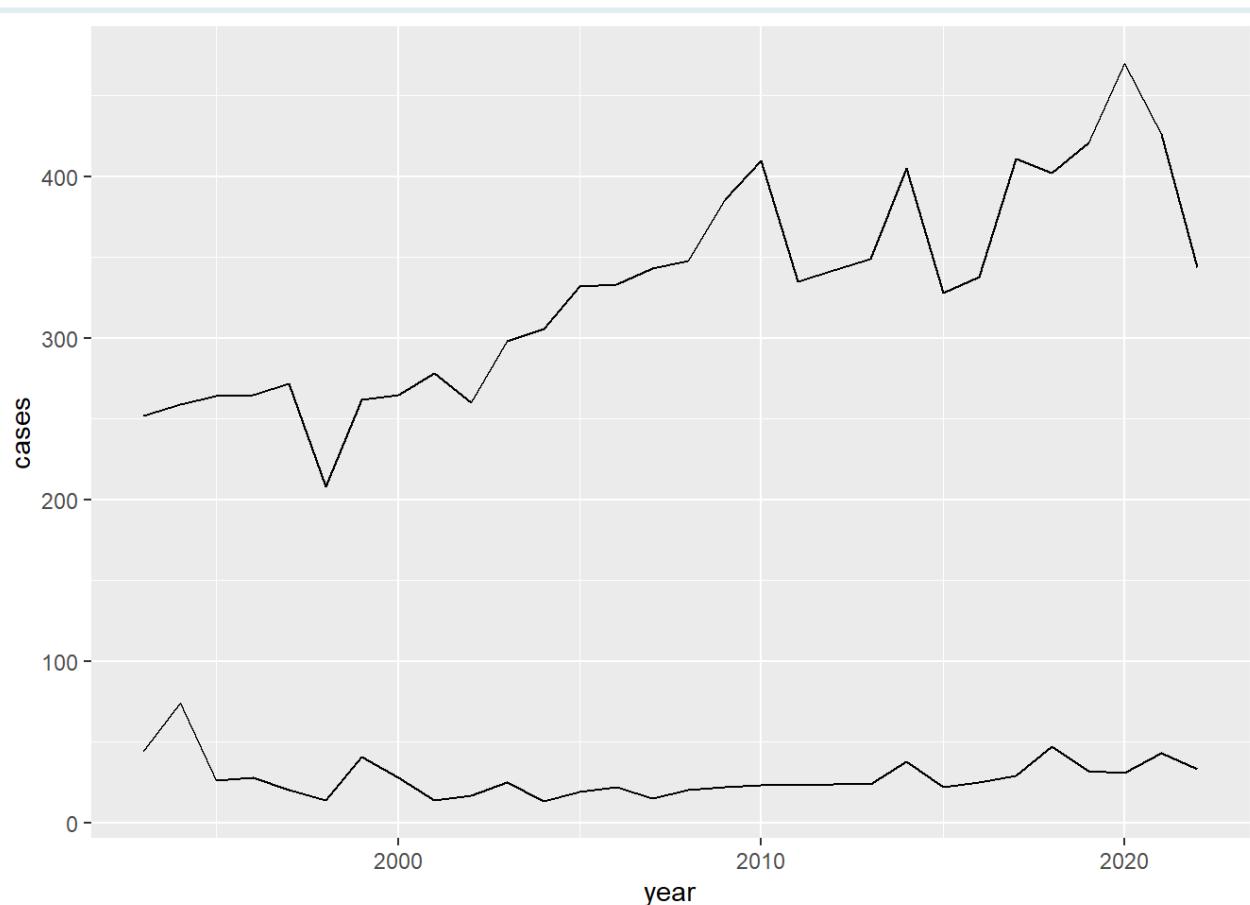
Un Graphique Linéaire Groupé Basique

Nous sommes prêts à représenter à nouveau les données. Nous associons les esthétiques couleur et groupe à la colonne `region`, qui contient les deux catégories d'intérêt : urbain et rural.

```

ggplot(aus_long, aes(x = year, y = cases, colour = region, group = region))
  +
  geom_line()

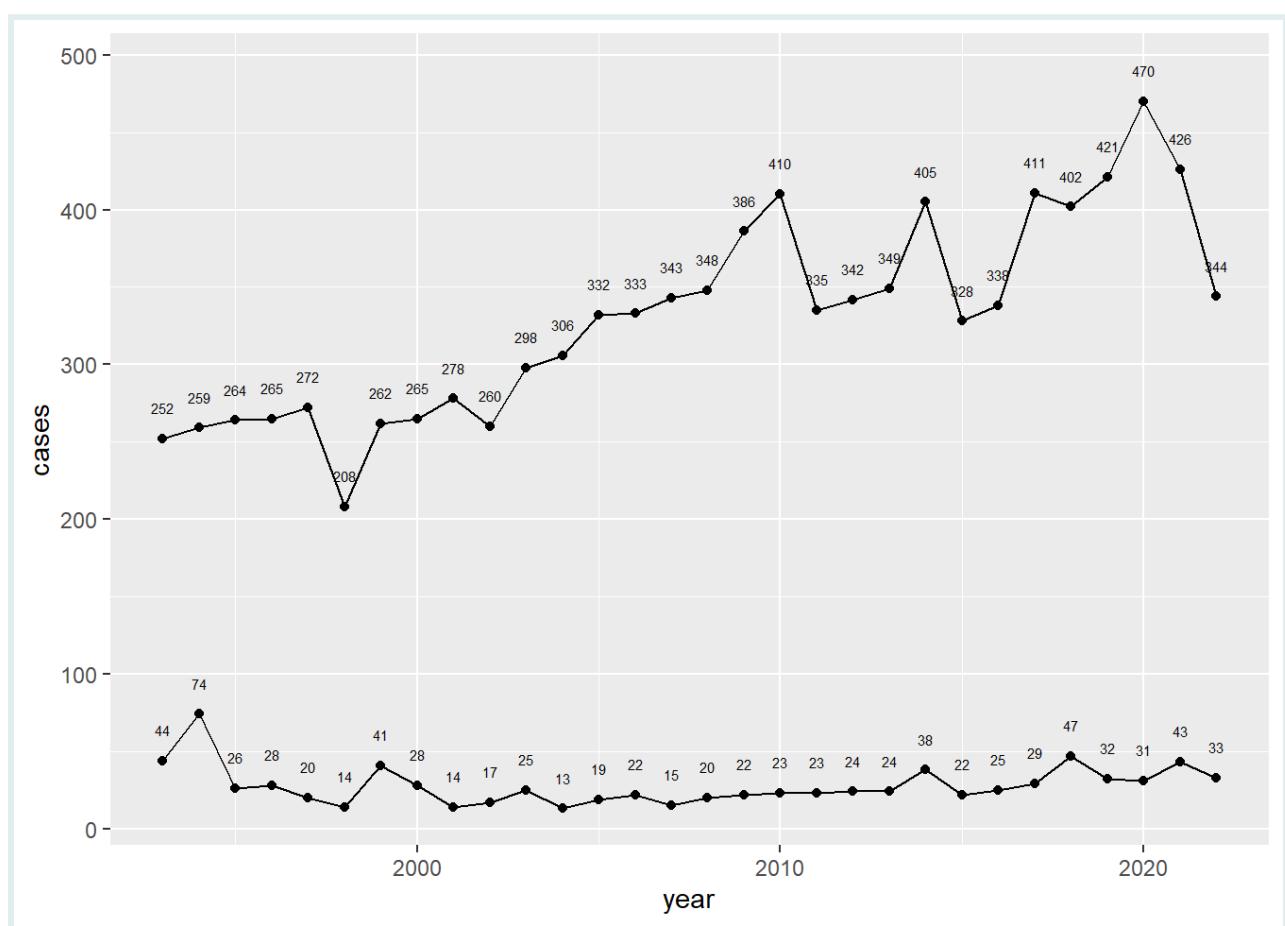
```



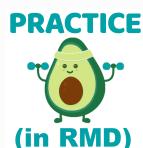
Le code de tracé est maintenant plus concis, grâce à l'opération de pivotement effectuée précédemment.

Nous pouvons maintenant également ajouter des points et des étiquettes textuelles avec beaucoup moins de code :

```
ggplot(aus_long, aes(x = year, y = cases, colour = region, group = region)) +  
  geom_line() +  
  geom_point() +  
  geom_text(aes(label = cases), size = 2, nudge_y = 20)
```



Super ! Nous avons maintenant une vue claire des tendances des notifications annuelles de cas de tuberculose (TB) dans les zones rurales et urbaines au fil du temps. Cependant, il y a encore des améliorations esthétiques que nous pouvons apporter ; nous les aborderons dans la section suivante.



Q : Remodelage et Représentation des Données sur la Tuberculose

Considérez le jeu de données du Bénin montré ci-dessous, qui contient des informations sur les cas de tuberculose confirmés

bactériologiquement et diagnostiqués cliniquement pendant plusieurs années au Bénin. (Les données proviennent d'un article [ici](#)

```
tb_data_benin <- read_csv(here("data/benin_tb_notifs.csv"))
tb_data_benin
```



```
## # A tibble: 15 × 3
##   year new_clindx new_labconf
##   <dbl>     <dbl>      <dbl>
## 1 2000       289      2280
## 2 2001       286      2289
## 3 2002       338      2428
## 4 2003       350      2449
## 5 2004       330      2577
## 6 2005       346      2731
## 7 2006       261      2950
## 8 2007       294      2755
## 9 2008       239      2983
## 10 2009      279      2950
## 11 2010      307      2958
## 12 2011      346      3326
## 13 2012      277      3086
## 14 2013      285      3219
## 15 2014      318      3062
```

Remodelez le jeu de données en utilisant `pivot_longer()`, puis créez un graphique avec deux lignes, une pour chaque type de diagnostic de cas de tuberculose. Ajoutez des points et des étiquettes textuelles au graphique.

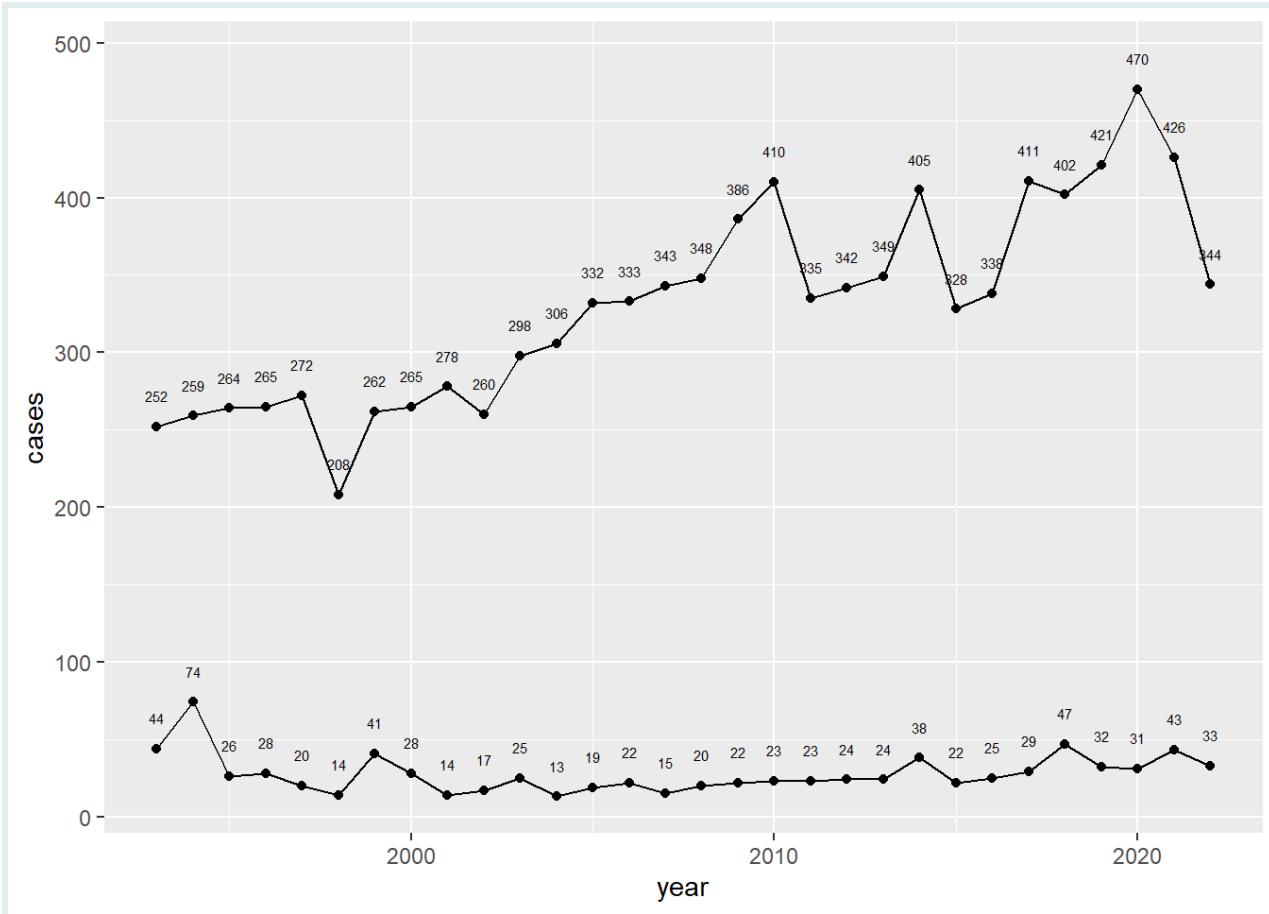
Améliorations Esthétiques des Graphiques Linéaires

Dans cette section, nous allons nous concentrer sur l'amélioration de l'esthétique des graphiques linéaires de séries temporelles pour renforcer leur clarté et leur attrait visuel.

Réduction de la Fréquence des Étiquettes

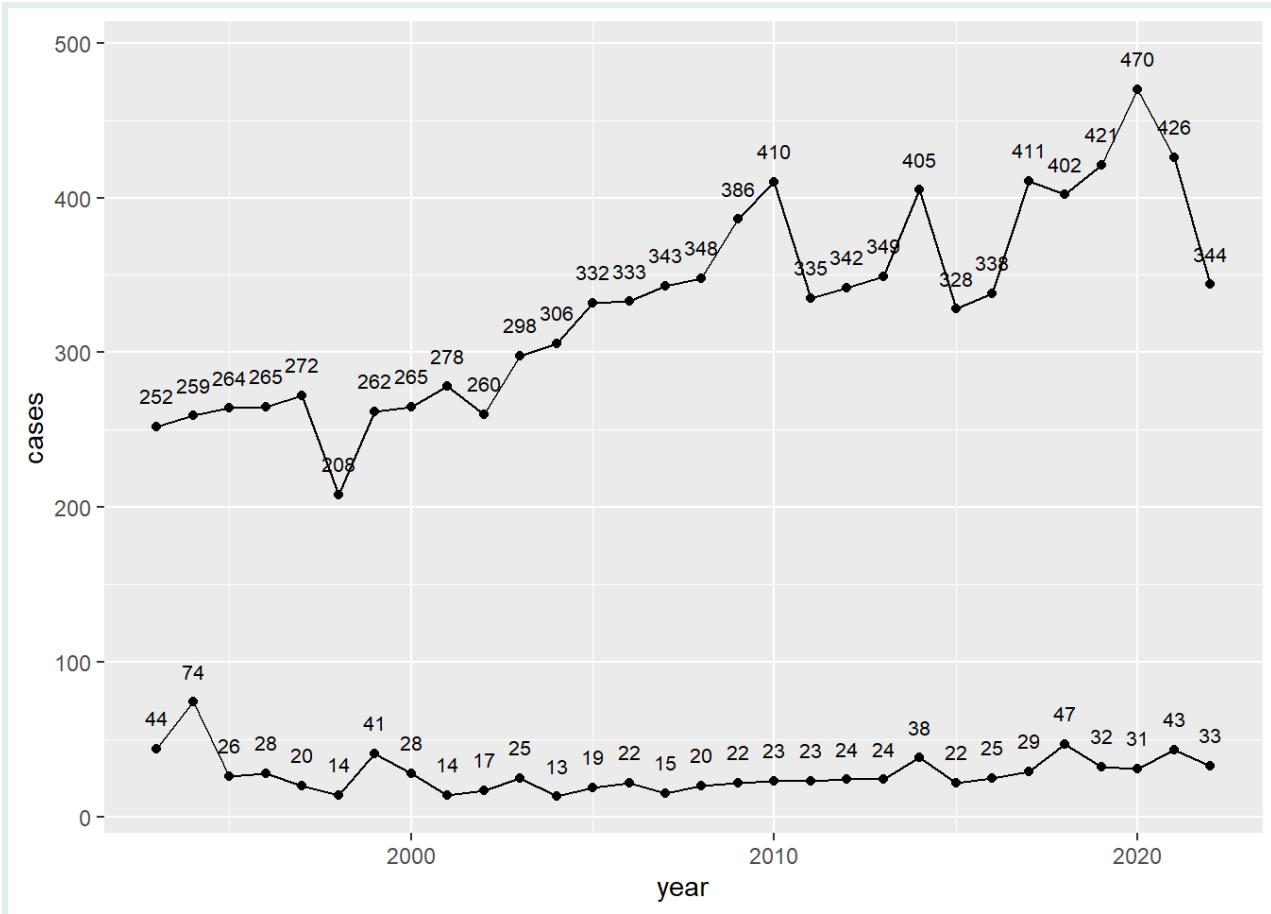
Lorsque nous avons laissé le graphique, il ressemblait à ceci :

```
ggplot(aus_long, aes(x = year, y = cases, colour = region, groupe = region))
  +
  geom_line() +
  geom_point() +
  geom_text(aes(label = cases), size = 2, nudge_y = 20)
```



Un problème avec ce graphique est que les étiquettes textuelles sont un peu trop petites. De telles étiquettes minuscules ne sont pas idéales pour un graphique destiné au public, car elles sont difficiles à lire. Cependant, si nous augmentons la taille des étiquettes, celles-ci commenceront à se chevaucher, comme illustré ci-dessous :

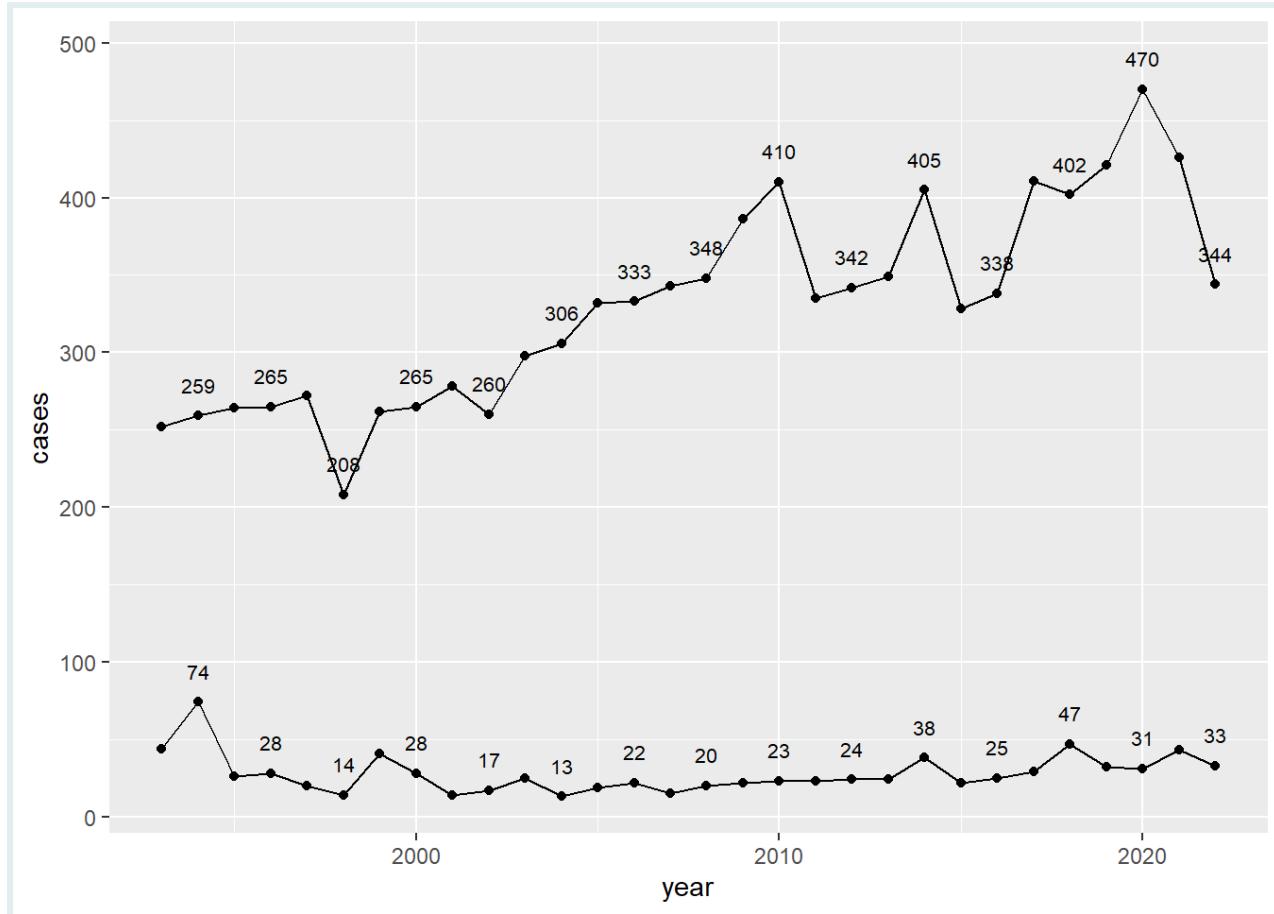
```
ggplot(aus_long, aes(x = year, y = cases, colour = region, group = region))
  +
  geom_line() +
  geom_point() +
  geom_text(aes(label = cases), size = 2.8, nudge_y = 20)
```



Pour éviter cet encombrement, une technique pratique consiste à afficher des étiquettes seulement pour certaines années. Pour ce faire, nous pouvons fournir un jeu de données personnalisé à la fonction `geom_text()`. Dans ce cas, nous allons créer un jeu de données qui contient uniquement les années paires :

```
even_years <- aus_long %>%
  filter(year %% 2 == 0) # Garder uniquement les années qui sont des multiples de 2

ggplot(aus_long, aes(x = year, y = cases, colour = region, group = region)) +
  geom_line() +
  geom_point() +
  geom_text(data = even_years, aes(label = cases),
            size = 2.8, nudge_y = 20)
```



Super, nous avons maintenant des étiquettes plus grandes et elles ne se chevauchent pas.

Alternance des Étiquettes

Bien que le graphique ci-dessus soit une amélioration, il serait encore mieux si nous pouvions afficher les étiquettes pour *toutes* les années. Nous pouvons le faire en affichant les étiquettes des années paires au-dessus des points de données, et les étiquettes des années impaires en dessous.

Inclure de nombreux points de données (dans la limite du raisonnable) dans vos graphiques est utile pour les responsables de la santé publique ; car ils peuvent rapidement tirer des chiffres du graphique lorsqu'ils essaient de prendre des décisions, sans avoir besoin de consulter les ensembles de données de référence.

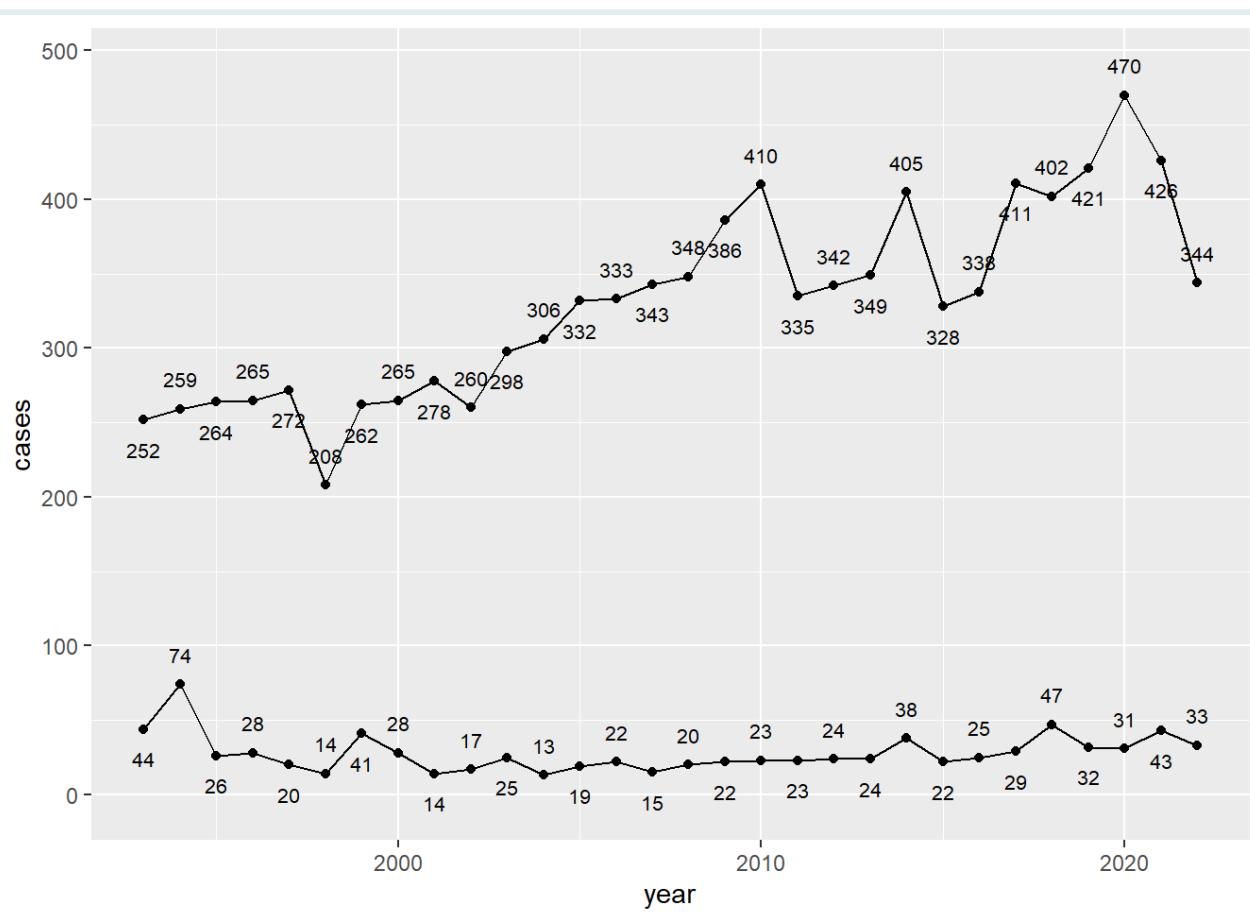
Pour cela, créons un jeu de données filtré pour les années impaires, puis utilisons `geom_text()` deux fois, une fois pour chaque jeu de données filtré.

```

odd_years <- aus_long %>%
  filter(year %% 2 != 0) # Garder uniquement les années qui NE SONT PAS des
                        # multiples de 2

ggplot(aus_long, aes(x = year, y = cases, colour = region, groupe = region)) +
  geom_line() +
  geom_point() +
  geom_text(data = even_years, aes(label = cases),
            nudge_y = 20, size = 2.8) +
  geom_text(data = odd_years, aes(label = cases),
            nudge_y = -20, size = 2.8)

```



ggrepel::geom_text_repel()

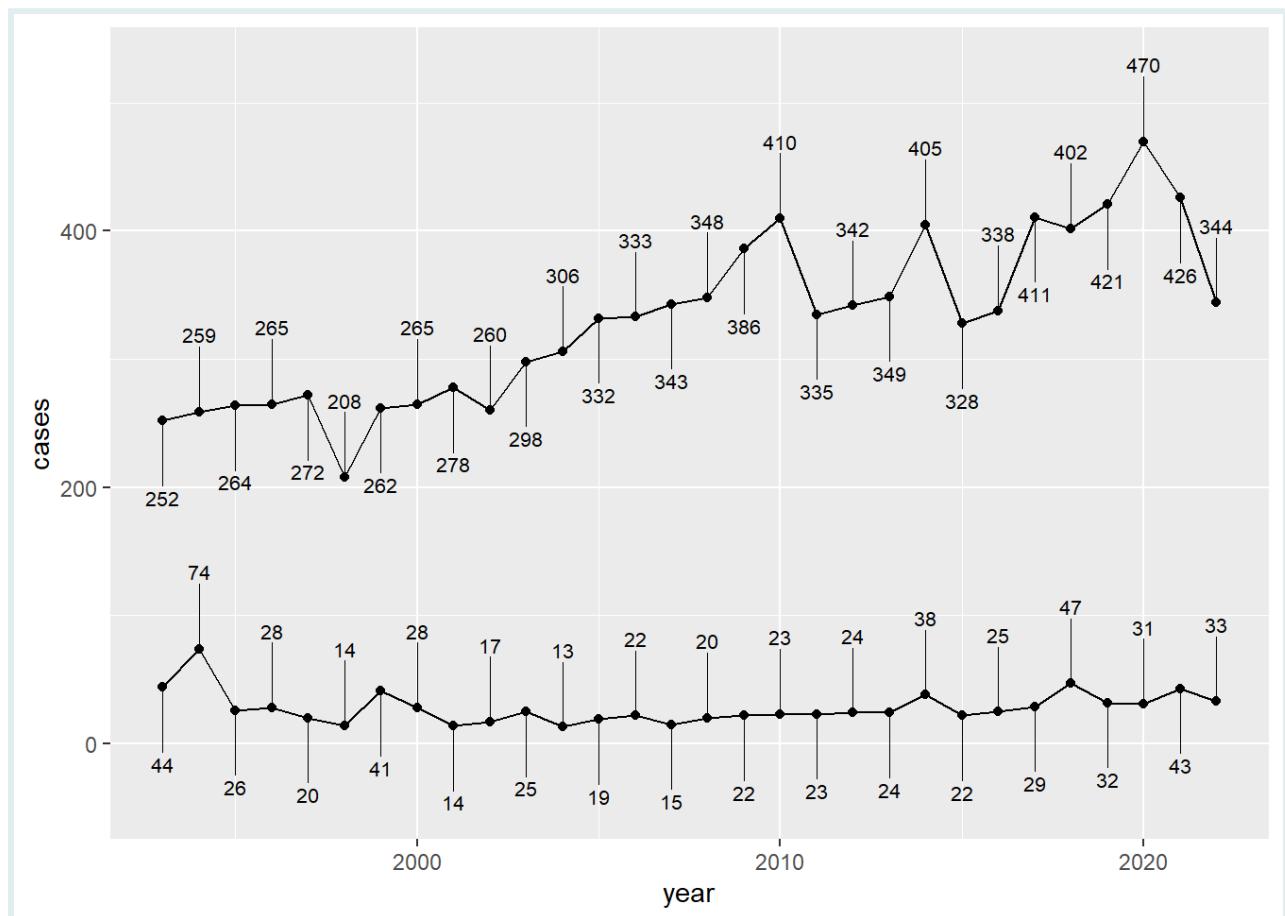
Le graphique ci-dessus est clair, mais il y a encore un certain chevauchement entre les étiquettes et la ligne.

Pour améliorer encore plus la clarté, nous pouvons utiliser la fonction `geom_text_repel()` du package `{ggrepel}`.

Cette fonction décale individuellement les étiquettes pour éviter le chevauchement et relie les étiquettes à leurs points de données avec des lignes, rendant plus facile de

voir à quel point de données chaque étiquette correspond, et nous permettant d'augmenter la distance entre les étiquettes et les points de données.

```
ggplot(aus_long, aes(x = year, y = cases, colour = region, group = region)) +
  geom_line() +
  geom_point() +
  geom_text_repel(data = even_years, aes(label = cases),
                  nudge_y = 60, size = 2.8, segment.size = 0.1) +
  geom_text_repel(data = odd_years, aes(label = cases),
                  nudge_y = -60, size = 2.8, segment.size = 0.1)
```



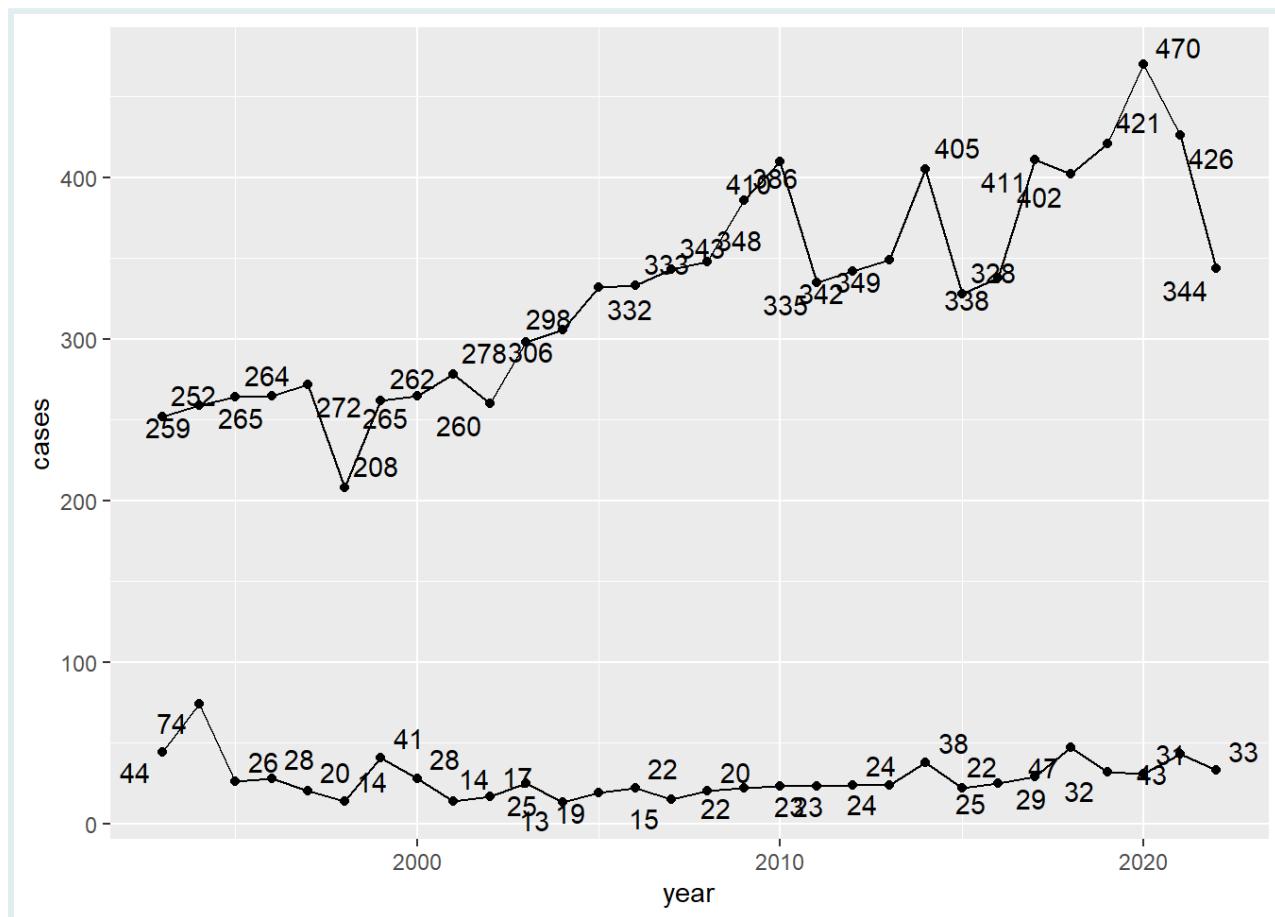
Comme vous pouvez le voir, la fonction `geom_text_repel()` prend essentiellement les mêmes arguments que `geom_text()`. L'argument supplémentaire, `segment.size`, contrôle la largeur des lignes reliant les étiquettes aux points de données.

Personnalisation de la Palette de Couleurs

Il est souvent utile de personnaliser la palette de couleurs de vos graphiques, pour qu'elle corresponde, par exemple, à la charte graphique de votre organisation.

Nous pouvons personnaliser les couleurs des lignes en utilisant la fonction `scale_color_manual()`. Ci-dessous, nous spécifions deux couleurs, une pour chaque région :

```
ggplot(aus_long, aes(x = year, y = cases, colour = region, group = region)) +
  geom_line() +
  geom_point() +
  geom_text_repel(data = even_years, aes(label = cases),
                  décalage_y = 60, taille = 2.8, segment.size = 0.1) +
  geom_text_repel(data = odd_years, aes(label = cases),
                  décalage_y = -60, taille = 2.8, segment.size = 0.1) +
  scale_color_manual(values = c("urbain" = "#0fa3b1",
                               "rural" = "#2F2C4E"))
```



Succès !

Ajout d'Annotations au Graphique

Enfin, ajoutons une série de touches finales. Nous allons annoter le graphique avec des titres appropriés, des étiquettes d'axe et des légendes, et modifier le thème :

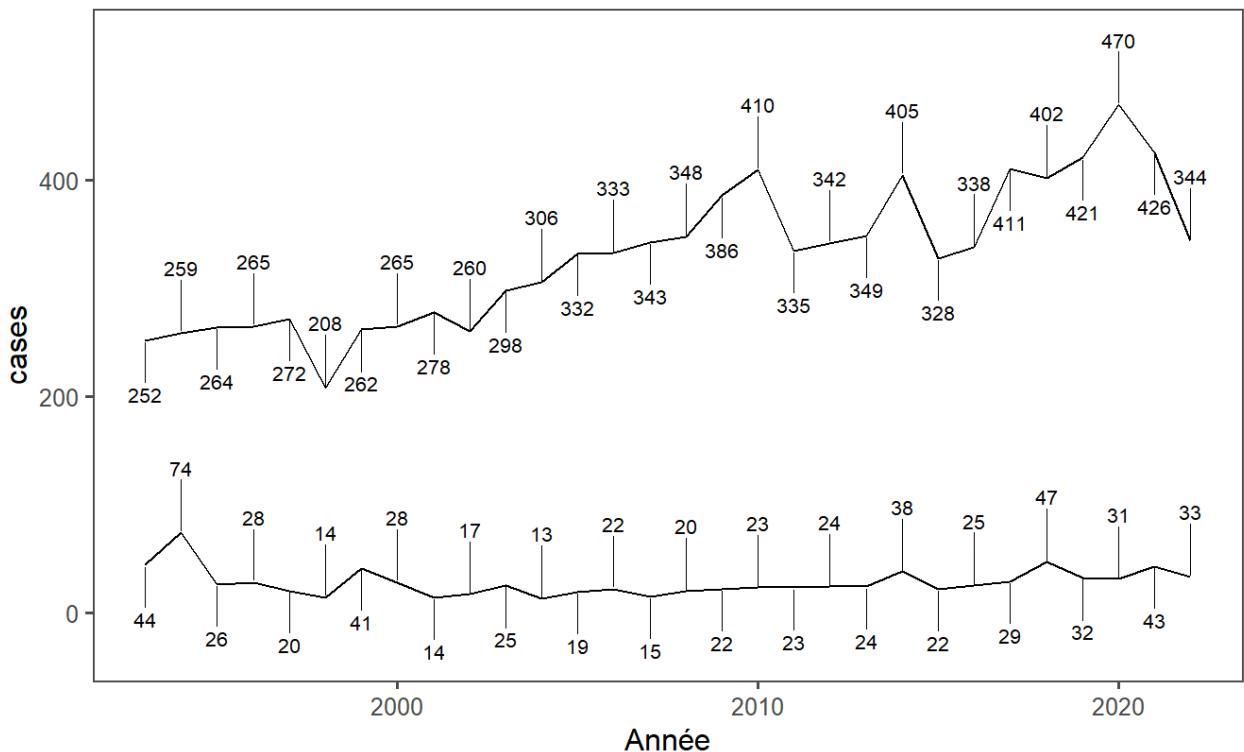
```

ggplot(aus_long, aes(x = year, y = cases, colour = region, groupe = region)) +
  geom_line(largeur_ligne = 1) +
  geom_text_repel(data = even_years, aes(label = cases),
                  nudge_y = 60, size = 2.8, segment.size = 0.08) +
  geom_text_repel(data = odd_years, aes(label = cases),
                  nudge_y = -50, size = 2.8, segment.size = 0.08) +
  scale_color_manual(values = c("urbain" = "#0fa3b1", "rural" = "#2F2C4E")) +
  labs(title = "Notifications de Tuberculose en Australie",
       subtitle = "1993-2022",
       caption = "Source : Département de la Santé du gouvernement de l'État de Victoria",
       x = "Année",
       couleur = "Région") +
  ggthemes::theme_few() +
  theme(legend.position = "droite")

```

Notifications de Tuberculose en Australie

1993-2022



Source : Département de la Santé du gouvernement de l'État de Victoria

Cela couvre certaines options pour améliorer l'esthétique des graphiques linéaires ! N'hésitez pas à ajuster davantage les visuels en fonction de vos besoins d'analyse spécifiques.

RECAP





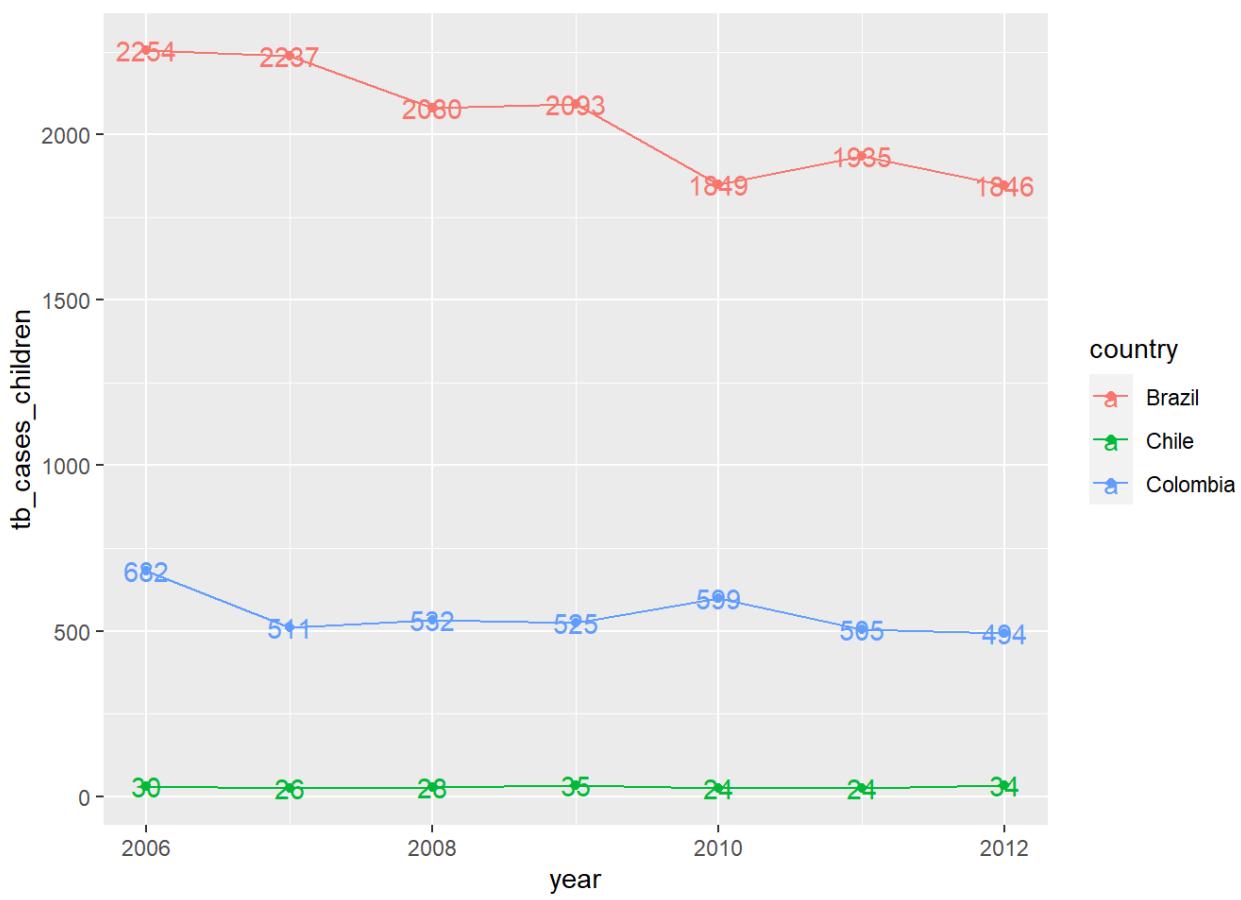
de tuberculose en Australie. Nous avons équilibré le besoin d'informations détaillées avec une présentation claire, rendant notre graphique à la fois informatif et accessible.

Q : Améliorations esthétiques

Considérez le graphique suivant, qui montre le nombre de cas de tuberculose chez les enfants dans trois pays au fil du temps :

```
tb_child_cases_southam <- tidyverse::who2 %>%
  transmute(country, year,
            tb_cases_children = sp_m_014 + sp_f_014 + sn_m_014 + sn_f_014) %>%
  filter(country %in% c("Brazil", "Colombia", "Chile")) %>%
  filter(!is.na(tb_cases_children))

tb_child_cases_southam %>%
  ggplot(aes(x = year, y = tb_cases_children, color = country)) +
  geom_line() +
  geom_point() +
  geom_text(aes(label = tb_cases_children))
```



Améliorez ce graphique en implémentant les améliorations suivantes :

- Réglez les étiquettes `geom_text` pour alterner au-dessus et en dessous des lignes, semblable à l'exemple que nous avons vu ci-dessus.
- Utilisez la palette de couleurs suivante `c("#212738", "#F97068", "#067BC2")`
- Appliquez `theme_classic()`
- Ajoutez un titre, un sous-titre et une légende pour fournir un contexte et des informations sur les données. (Vous pouvez taper `?tidyverse::whois` dans la console pour en savoir plus sur la source des données.)

Représentation des Intervalles de Confiance avec `geom_ribbon()`

Dans les visualisations de séries temporelles, il est souvent important de représenter les intervalles de confiance pour indiquer le niveau d'incertitude de vos données.

Nous allons démontrer comment faire cela en utilisant un jeu de données sur les nouvelles infections par le VIH au Brésil, qui comprend les nombres estimés pour les cas masculins et féminins ainsi que les intervalles de confiance. Le jeu de données est issu de l'Organisation Mondiale de la Santé (OMS) et peut être consulté [ici](#).

Commençons par charger et inspecter le jeu de données :

```
hiv_data_brazil <-
  rio::import(here("data/new_hiv_infections_gho.xlsx"),
             sheet = "Brazil") %>%
  as_tibble() %>%
  janitor::clean_names()
hiv_data_brazil
```

```
## # A tibble: 89 × 5
##   continent country year sex
##   <chr>     <chr>  <dbl> <chr>
## 1 Americas  Brazil    2022 Female
## 2 Americas  Brazil    2022 Male
## 3 Americas  Brazil    2022 Both sexes
## 4 Americas  Brazil    2021 Female
## 5 Americas  Brazil    2021 Male
## 6 Americas  Brazil    2021 Both sexes
## 7 Americas  Brazil    2020 Female
## 8 Americas  Brazil    2020 Male
## 9 Americas  Brazil    2020 Both sexes
## 10 Americas Brazil    2019 Female
##   new_hiv_cases
##   <chr>
## 1 13 000 [12 000 – 15 000]
## 2 37 000 [35 000 – 40 000]
## 3 51 000 [47 000 – 54 000]
## 4 14 000 [12 000 – 15 000]
```

```

## 5 37 000 [34 000 – 39 000]
## 6 50 000 [47 000 – 53 000]
## 7 14 000 [13 000 – 15 000]
## 8 35 000 [32 000 – 37 000]
## 9 49 000 [46 000 – 52 000]
## 10 14 000 [13 000 – 15 000]
## # i 79 more rows

```

Nous pouvons voir que la colonne `new_hiv_cases` contient à la fois le nombre de cas et les intervalles de confiance correspondants entre crochets. Ce format ne peut pas être utilisé directement pour la représentation graphique, donc nous devons les extraire sous forme numérique pure.

Tout d'abord, pour séparer ces valeurs, nous pouvons utiliser la fonction `separate()` du package `{tidyverse}` :

```

hiv_data_brazil %>%
  separate(new_hiv_cases,
          into = c("cases", "cases_lower", "cases_upper"),
          sep = "\\[|]")

```

```

## # A tibble: 89 × 7
##   continent country year sex      cases
##   <chr>     <chr>  <dbl> <chr>    <chr>
## 1 Americas   Brazil   2022 Female   "13 000 "
## 2 Americas   Brazil   2022 Male    "37 000 "
## 3 Americas   Brazil   2022 Both sexes "51 000 "
## 4 Americas   Brazil   2021 Female   "14 000 "
## 5 Americas   Brazil   2021 Male    "37 000 "
## 6 Americas   Brazil   2021 Both sexes "50 000 "
## 7 Americas   Brazil   2020 Female   "14 000 "
## 8 Americas   Brazil   2020 Male    "35 000 "
## 9 Americas   Brazil   2020 Both sexes "49 000 "
## 10 Americas  Brazil   2019 Female  "14 000 "
## 
##   cases_lower cases_upper
##   <chr>        <chr>
## 1 "12 000 "   " 15 000]"
## 2 "35 000 "   " 40 000]"
## 3 "47 000 "   " 54 000]"
## 4 "12 000 "   " 15 000]"
## 5 "34 000 "   " 39 000]"
## 6 "47 000 "   " 53 000]"
## 7 "13 000 "   " 15 000]"
## 8 "32 000 "   " 37 000]"
## 9 "46 000 "   " 52 000]"
## 10 "13 000 "  " 15 000]"
## # i 79 more rows

```

Dans le code ci-dessus, nous divisons la colonne `cases_upper` en trois nouvelles colonnes : `cases`, `cases_lower` et `cases_upper`. Nous utilisons `[` et `-` comme séparateurs. Le double antislash `\\"` est utilisé pour échapper au crochet, qui a une

signification spéciale dans les expressions régulières. Et le | est utilisé pour indiquer que soit [soit – peut être utilisé comme séparateur.



Les modèles de langage de grande taille comme ChatGPT sont excellents pour comprendre les expressions régulières. Si vous êtes bloqué avec un code comme `sep = "\\| -"` et que vous voulez comprendre ce qu'il fait, vous pouvez demander à ChatGPT de vous l'expliquer. Et si vous avez besoin de générer de telles expressions vous-même, vous pouvez demander à ChatGPT de les générer pour vous.

Ensuite, nous devons convertir ces valeurs en chaîne de caractères en valeurs numériques, en supprimant tous les caractères non numériques.

```
hiv_data_brazil_clean <-
  hiv_data_brazil %>%
  separate(new_hiv_cases,
    into = c("cases", "cases_lower", "cases_upper"),
    sep = "\\| -") %>%
  mutate(across(c("cases", "cases_lower", "cases_upper"),
    ~ str_replace_all(.x, "[^0-9]", "") %>%
      as.numeric()))
```

```
hiv_data_brazil_clean
```

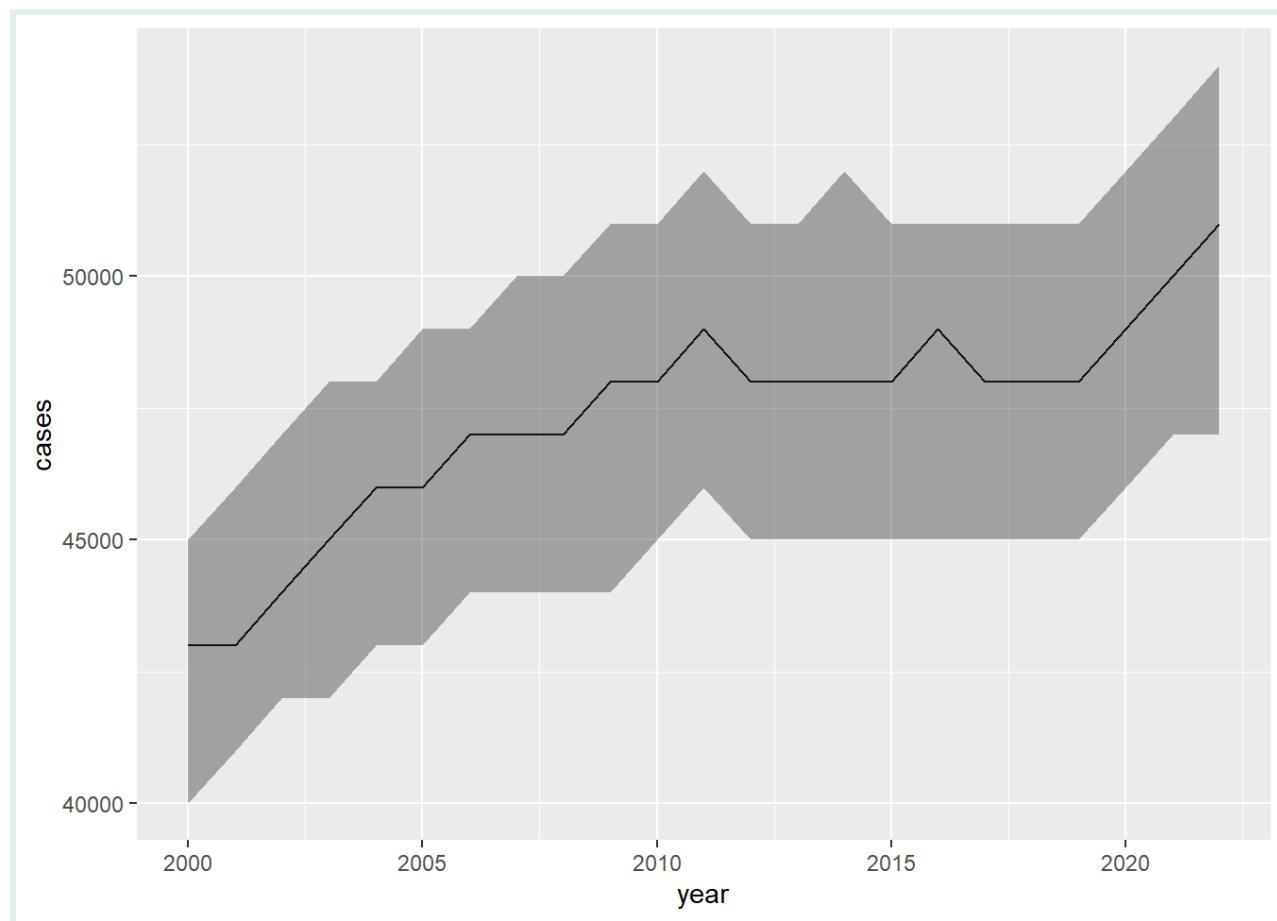
```
## # A tibble: 89 × 7
##   continent country year sex      cases cases_lower
##   <chr>     <chr> <dbl> <chr>     <dbl>      <dbl>
## 1 Americas   Brazil  2022 Female    13000     12000
## 2 Americas   Brazil  2022 Male     37000     35000
## 3 Americas   Brazil  2022 Both sexes 51000     47000
## 4 Americas   Brazil  2021 Female    14000     12000
## 5 Americas   Brazil  2021 Male     37000     34000
## 6 Americas   Brazil  2021 Both sexes 50000     47000
## 7 Americas   Brazil  2020 Female    14000     13000
## 8 Americas   Brazil  2020 Male     35000     32000
## 9 Americas   Brazil  2020 Both sexes 49000     46000
## 10 Americas  Brazil  2019 Female   14000     13000
##   cases_upper
##   <dbl>
## 1 15000
## 2 40000
## 3 54000
## 4 15000
## 5 39000
## 6 53000
## 7 15000
## 8 37000
## 9 52000
```

```
## 10      15000
## # i 79 more rows
```

Le code ci-dessus semble complexe, mais essentiellement, il nettoie les données en ne conservant que les caractères numériques, puis convertit ces nombres en valeurs numériques réelles. Voir notre leçon sur la fonction `across()` pour plus de détails.

Nous sommes enfin prêts à représenter les données. Nous utiliserons `geom_ribbon()` de ggplot pour afficher les intervalles de confiance :

```
hiv_data_brazil_clean %>%
  filter(sex == "Both sexes") %>%
  ggplot(aes(x = year, y = cases)) +
  geom_line() +
  geom_ribbon(aes(ymin = cases_lower, ymax = cases_upper), alpha = 0.4)
```



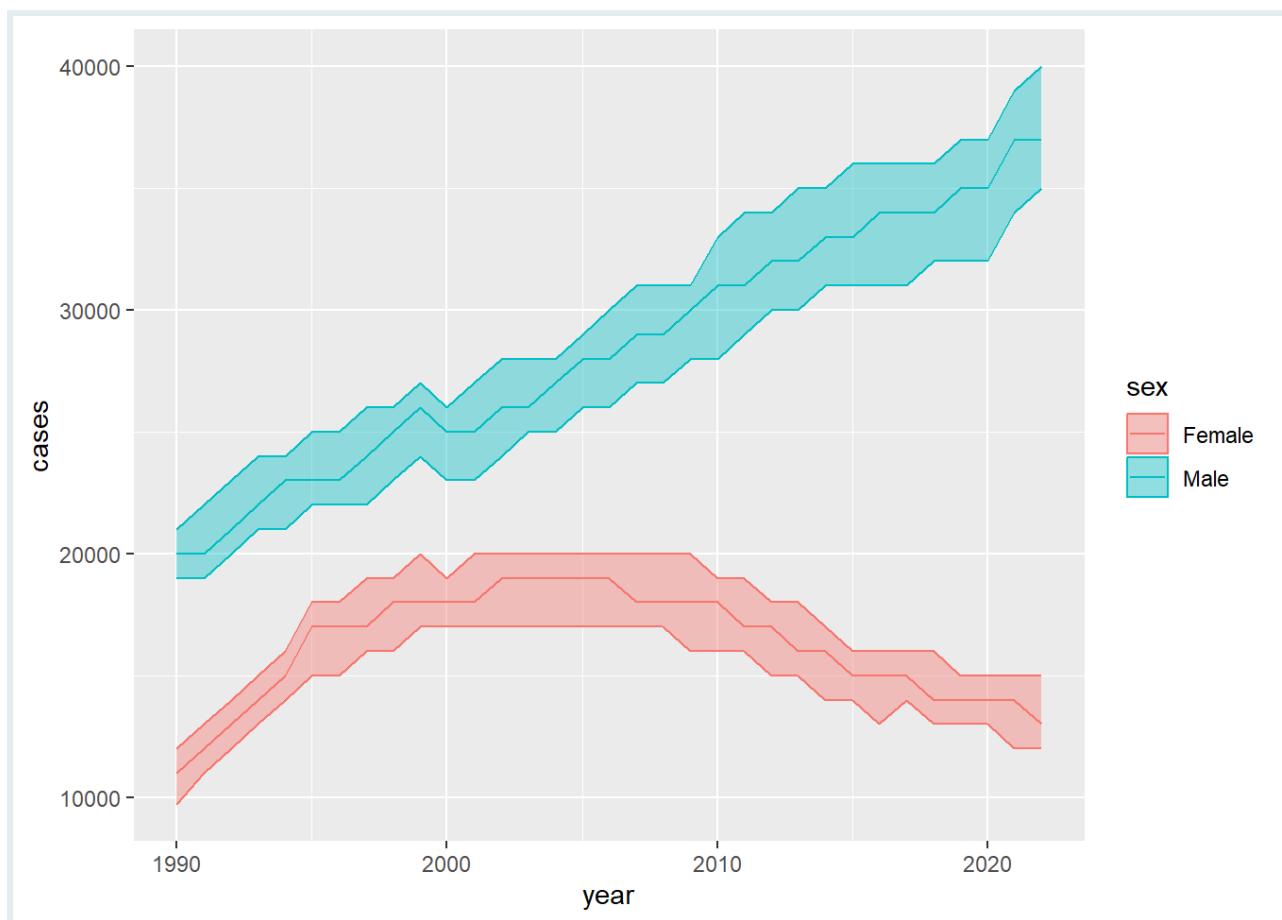
La fonction `geom_ribbon()` prend les esthétiques `x` et `y` comme `geom_line()`, mais elle prend également en compte les esthétiques `ymin` et `ymax`, pour déterminer l'étendue verticale du ruban. Nous réglons également la transparence du ruban à l'aide de l'argument `alpha`.

Nous pouvons créer un ruban séparé pour les hommes et les femmes pour comparer leurs tendances d'infection.

```

hiv_data_brazil_clean %>%
  filter(sex != "Both sexes") %>%
  ggplot(aes(x = year, y = cases, color = sex, fill = sex)) +
  geom_line() +
  geom_ribbon(aes(ymin = cases_lower, ymax = cases_upper), alpha = 0.4)

```



Il est à noter que les taux d'infection par le VIH chez les femmes ont diminué ces dernières années, mais ceux chez les hommes ont augmenté.

Q : Représentation des intervalles de confiance

PRACTICE



Considérez le jeu de données suivant qui montre le nombre de cas annuels de paludisme au Kenya et au Nigeria. Les données proviennent du dépôt de données de l'Observatoire mondial de la santé de l'OMS et peuvent être consultées [ici](#).

```

nig_ken_mal <- read_csv("data/nigeria_kenya_malaria.csv")
nig_ken_mal

```



```
## # A tibble: 44 × 3
##   country year malaria_cases
##   <chr>    <dbl>     <chr>
## 1 Kenya      2021 3 419 698 (2 478 000 to 4 641 000)
## 2 Nigeria    2021 65 399 501 (47 520 000 to 89 000 000)
## 3 Kenya      2020 3 302 189 (2 385 000 to 4 466 000)
## 4 Nigeria    2020 65 133 759 (46 800 000 to 88 650 000)
## 5 Kenya      2019 3 037 541 (2 168 000 to 4 130 000)
## 6 Nigeria    2019 61 379 283 (47 440 000 to 78 810 000)
## 7 Kenya      2018 3 068 062 (2 148 000 to 4 260 000)
## 8 Nigeria    2018 59 652 248 (46 930 000 to 75 230 000)
## 9 Kenya      2017 3 155 636 (2 217 000 to 4 375 000)
## 10 Nigeria   2017 57 869 533 (45 870 000 to 72 050 000)
## # ... with 34 more rows
```

Écrivez du code pour extraire les intervalles de confiance de la colonne “malaria_cases” et créez un graphique avec des intervalles de confiance en utilisant `geom_ribbon()`. Utilisez une couleur différente pour chaque pays.

Lissage des Tendances Bruyantes

Lors de l’analyse de données de séries temporelles, il est courant que les mesures quotidiennes ou granulaires montrent beaucoup de bruit et de variabilité, ce qui peut masquer les tendances importantes qui nous intéressent réellement. Les techniques de lissage peuvent aider à mettre en évidence ces tendances et motifs. Nous allons explorer plusieurs techniques à cet effet dans les sections ci-dessous.

Tout d’abord, faisons quelques préparations de données !

Création d’un Tableau d’Incidence à partir d’une Liste de Cas

Considérez la liste de cas suivante des admissions pédiatriques pour la malaria dans quatre hôpitaux au Mozambique ([Source de données](#)) :

```
mal <-  
  rio::import(here("data/pediatric_malaria_data_joao_2021.xlsx")) %>%  
  as_tibble() %>%  
  mutate(date_positive_test = as.Date(date_positive_test)) %>%  
  # Keep data from 2019-2020  
  filter(date_positive_test >= as.Date("2019-01-01"),  
         date_positive_test <= as.Date("2020-12-31"))  
mal
```

```

## # A tibble: 20,939 × 4
##   date_positive_test neighbourhood sex    age
##   <date>              <chr>        <chr> <chr>
## 1 2020-01-22          25 de Junho  1 M     6-11 meses
## 2 2020-01-22          Chicueu      F     5-14 anos
## 3 2020-01-22          Mussessa     M     5-14 anos
## 4 2020-01-22          Nhamizara    M     12-23 meses
## 5 2020-01-22          Nhamizara    F     12-23 meses
## 6 2020-01-22          Unidade      F     5-14 anos
## 7 2020-01-22          Bapua        F     12-23 meses
## 8 2020-01-22          Bapua        F     5-14 anos
## 9 2020-01-22          7 de Abril    M     12-23 meses
## 10 2020-01-22         Nhanguzue    F     5-14 anos
## # i 20,929 more rows

```

Chaque ligne correspond à un cas unique de malaria, et la colonne `date_test_positif` indique la date à laquelle l'enfant a été testé positif pour la malaria.

Pour obtenir un décompte des cas par jour - c'est-à-dire, un tableau d'incidence - nous pouvons simplement utiliser `count()` pour agréger les cas par date de test positif :

```

mal %>%
  count(date_positive_test, name = "cases")

```

```

## # A tibble: 235 × 2
##   date_positive_test cases
##   <date>           <int>
## 1 2019-01-01        67
## 2 2019-01-02       120
## 3 2019-01-03       112
## 4 2019-01-04       203
## 5 2019-01-05        85
## 6 2019-01-07       115
## 7 2019-01-08       196
## 8 2019-01-10        89
## 9 2019-01-11        55
## 10 2019-01-12       69
## # i 225 more rows

```

Cependant, de nombreuses dates manquent - les jours où aucun enfant n'a été admis. Pour créer un tableau d'incidence complet, nous devrions utiliser `complete()` pour insérer les dates manquantes, puis remplir les valeurs manquantes avec 0 :

```

mal_notif_count <- mal %>%
  count(date_positive_test, name = "cases") %>%
  complete(date_positive_test = seq.Date(min(date_positive_test),
                                         max(date_positive_test),
                                         by = "day"),
           fill = list(cases = 0))

mal_notif_count

```

```

## # A tibble: 406 × 2
##   date_positive_test cases
##   <date>             <int>
## 1 2019-01-01          67
## 2 2019-01-02         120
## 3 2019-01-03         112
## 4 2019-01-04         203
## 5 2019-01-05          85
## 6 2019-01-06           0
## 7 2019-01-07         115
## 8 2019-01-08         196
## 9 2019-01-09           0
## 10 2019-01-10          89
## # ... with 396 more rows

```

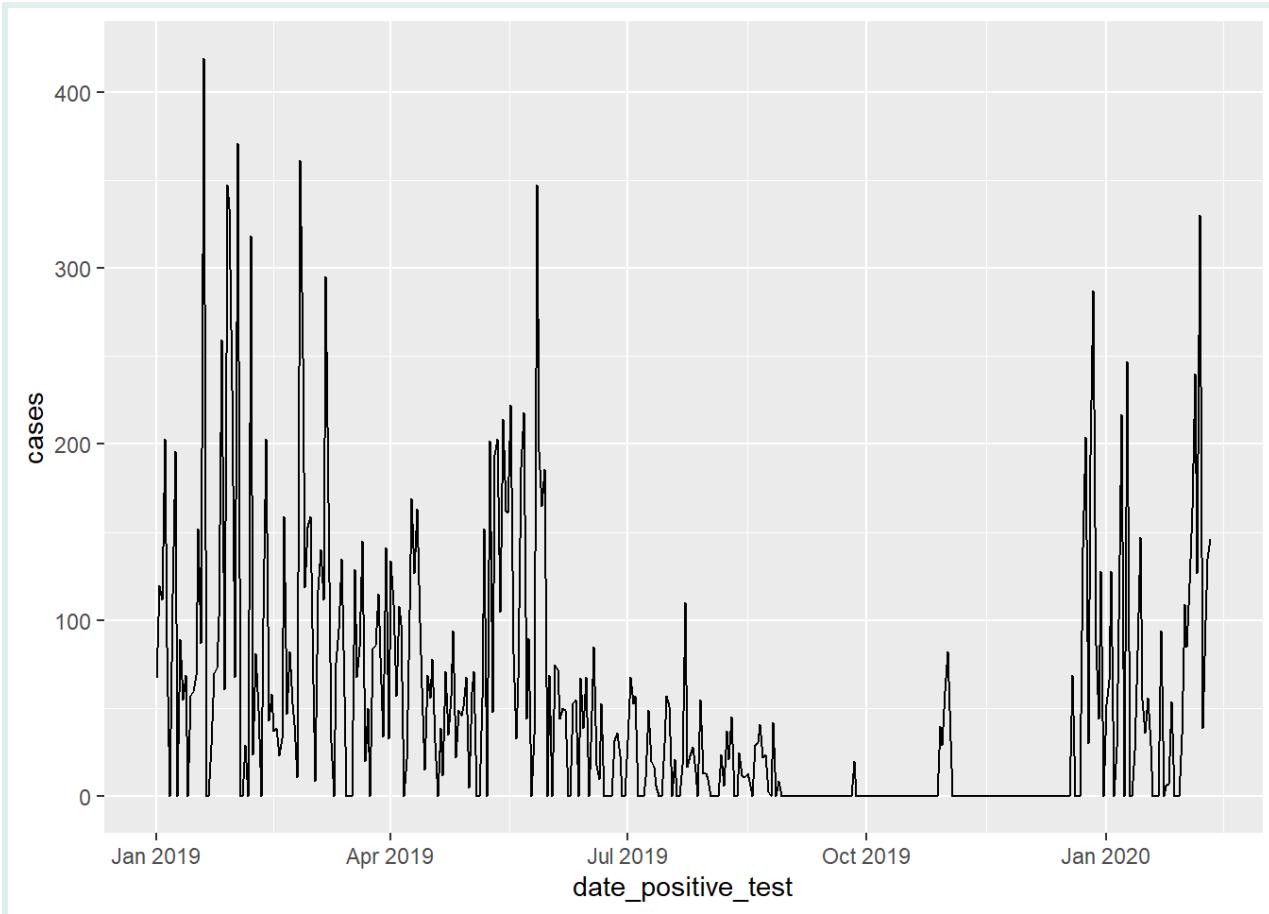
Maintenant, nous avons un tableau d'incidence complet avec le nombre de cas sur 406 jours consécutifs.

Nous pouvons maintenant tracer les données pour voir la tendance globale :

```

# Créer une épicourbe basique en utilisant ggplot2
ggplot(mal_notif_count, aes(x = date_positive_test, y = cases)) +
  geom_line()

```

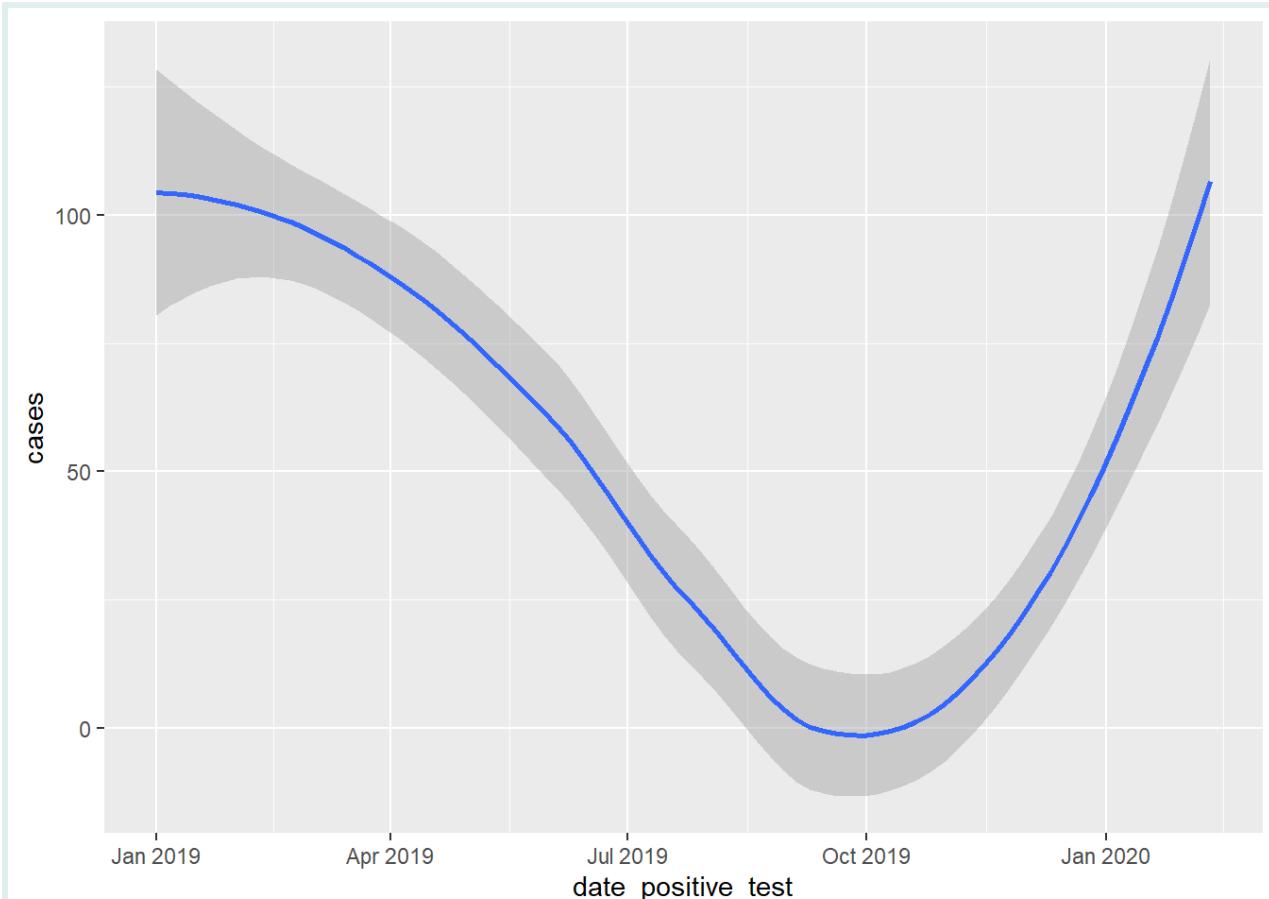


Nous avons une épicurve valide, mais comme vous pouvez le remarquer, la variabilité quotidienne rend difficile de voir la tendance globale. Lissage à suivre.

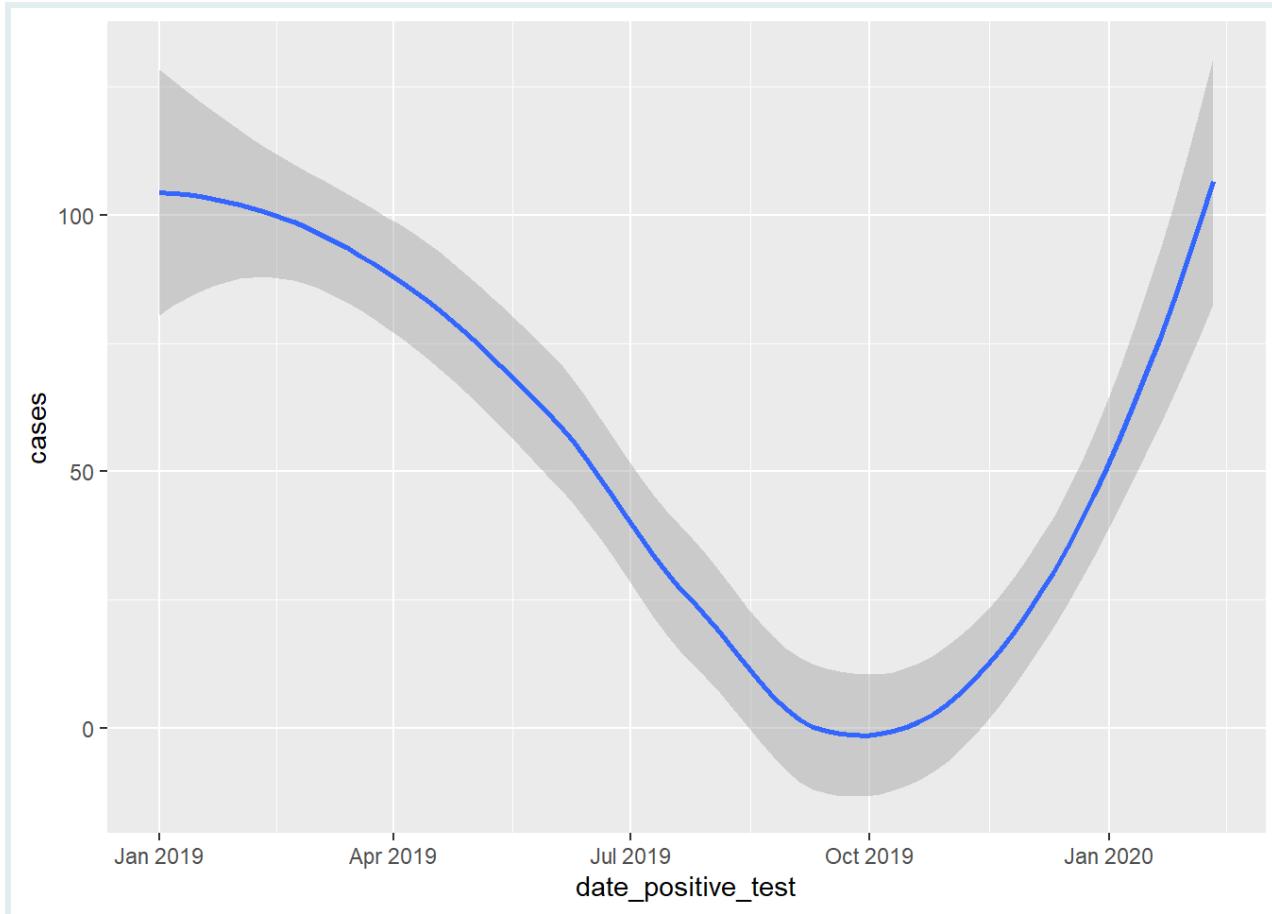
Lissage avec `geom_smooth()`

Une option pour le lissage est la fonction `geom_smooth()`, qui peut effectuer une régression locale avec `loess` pour lisser la série temporelle. Essayons :

```
ggplot(mal_notif_count, aes(x = date_positive_test, y = cases)) +
  geom_smooth()
```



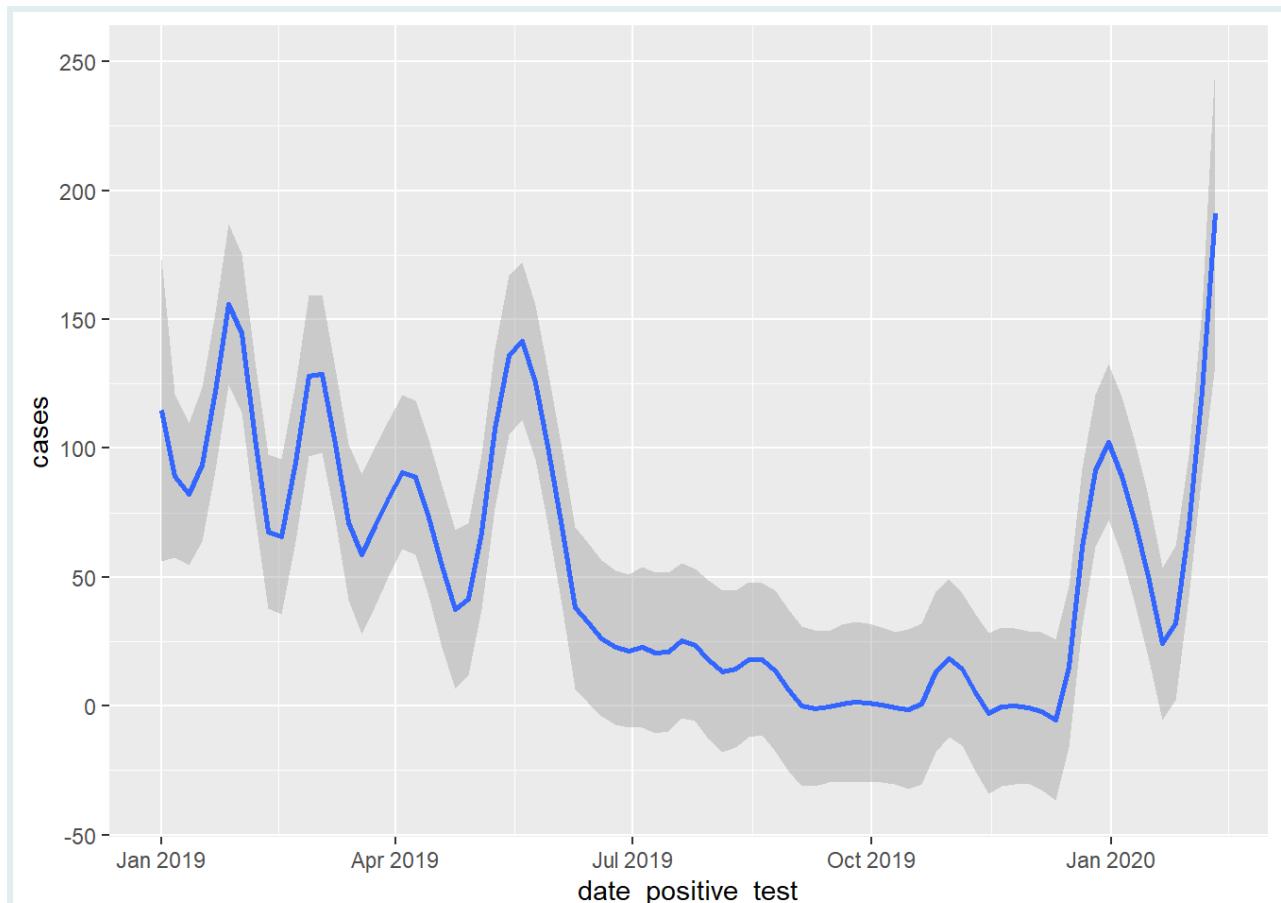
```
# Or we can specify the method explicitly  
ggplot(mal_notif_count, aes(x = date_positive_test, y = cases)) +  
  geom_smooth(method = "loess")
```



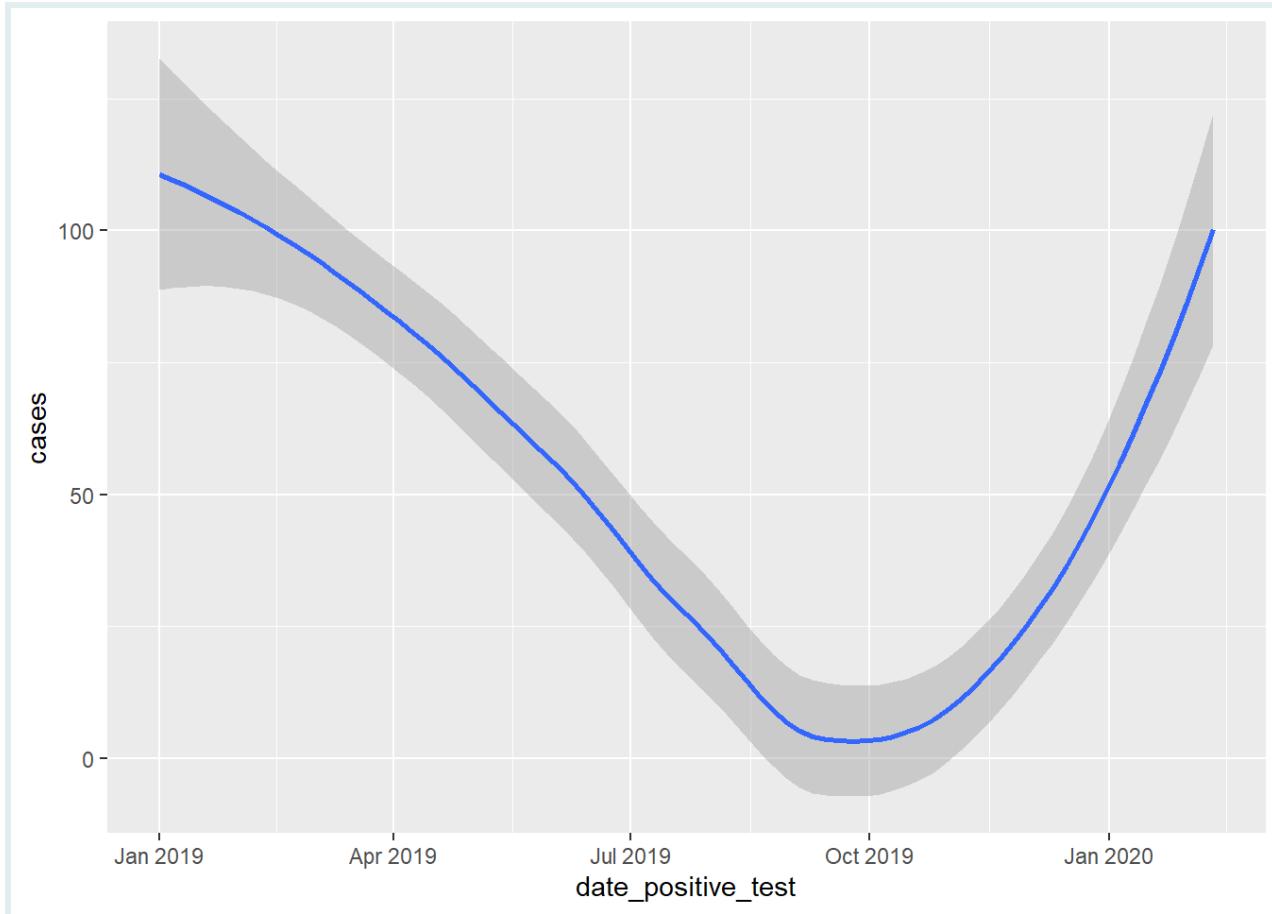
La méthode `loess`, qui signifie “locally weighted scatterplot smoothing”, ajuste une courbe lisse aux données en calculant des moyennes pondérées pour les points proches.

Vous pouvez ajuster la sensibilité du lissage en modifiant l’argument `span`. Un `span` de 0.1 donnera un lissage plus sensible, tandis qu’un `span` de 0.9 donnera un lissage moins sensible.

```
# Ajuster la sensibilité du lissage
ggplot(mal_notif_count, aes(x = date_positive_test, y = cases)) +
  geom_smooth(method = "loess", span = 0.1)
```



```
ggplot(mal_notif_count, aes(x = date_positive_test, y = cases)) +  
  geom_smooth(method = "loess", span = 0.9)
```



Lissage par Agrégation

Une autre façon de lisser les données est de les agréger - regrouper les données en intervalles de temps plus grands et calculer des statistiques récapitulatives pour chaque intervalle.

Nous avons déjà vu cela au début de la leçon, lorsque nous avons agrégé les données trimestrielles en données annuelles.

Appliquons-le à nouveau, cette fois en agrégeant l'incidence quotidienne de la malaria en incidence mensuelle. Pour cela, nous utilisons la fonction `floor_date()` du package `lubridate` pour arrondir les dates au mois le plus proche :

```
mal_notif_count %>%
  mutate(month = floor_date(date_positive_test, unit = "month"))
```

```
## # A tibble: 406 x 3
##   date_positive_test cases month
##   <date>           <int> <date>
## 1 2019-01-01          67 2019-01-01
## 2 2019-01-02         120 2019-01-01
## 3 2019-01-03         112 2019-01-01
## 4 2019-01-04         203 2019-01-01
```

```

##   5 2019-01-05          85 2019-01-01
##   6 2019-01-06           0 2019-01-01
##   7 2019-01-07          115 2019-01-01
##   8 2019-01-08          196 2019-01-01
##   9 2019-01-09           0 2019-01-01
##  10 2019-01-10          89 2019-01-01
## # i 396 more rows

```

Nous pouvons ensuite utiliser `group_by()` et `summarize()` pour calculer le nombre total de cas par mois :

```

mal_monthly <-
  mal_notif_count %>%
  mutate(month = floor_date(date_positive_test, unit = "month")) %>%
  group_by(month) %>%
  summarize(cases = sum(cases))

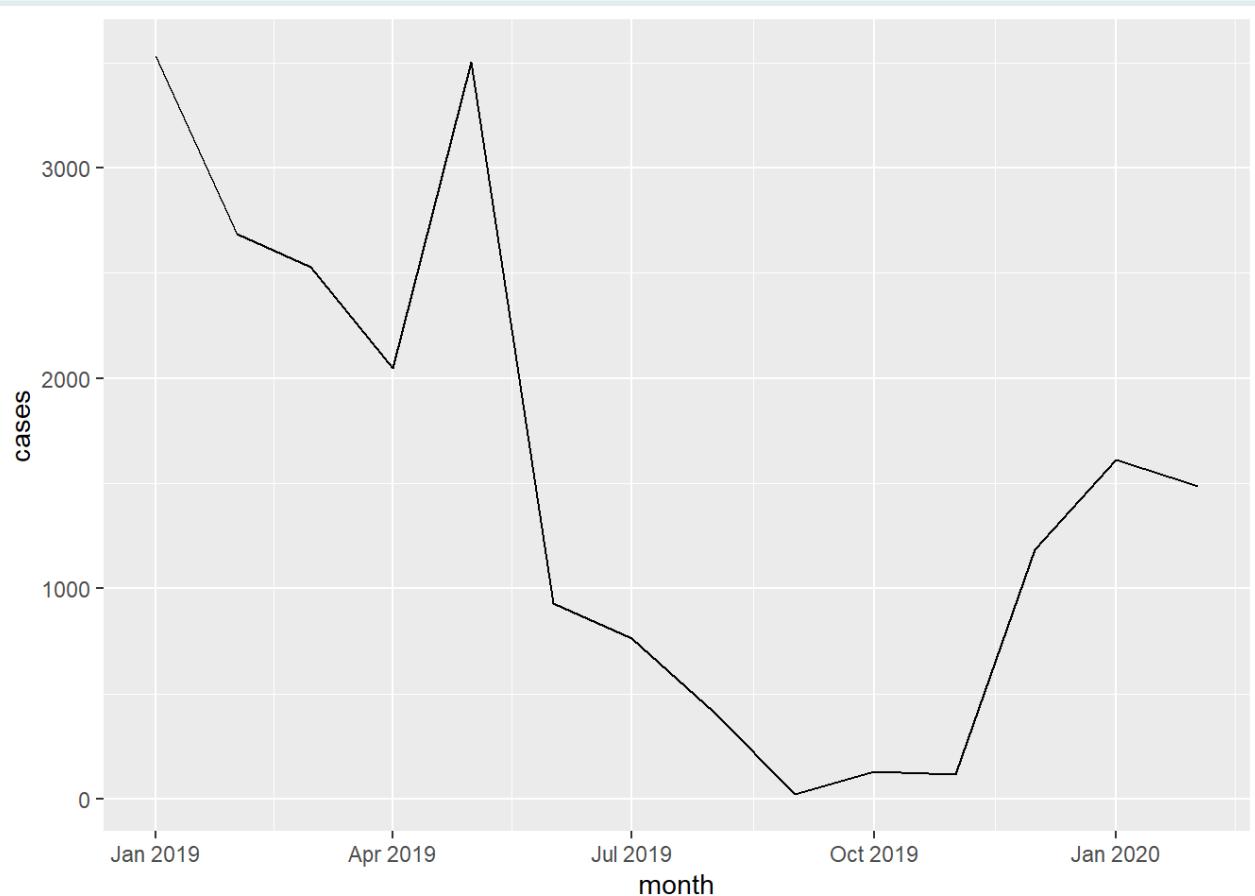
```

Cela nous donne un tableau d'incidence mensuelle, que nous pouvons tracer pour voir la tendance globale :

```

ggplot(mal_monthly, aes(x = month, y = cases)) +
  geom_line()

```



Voilà ! Une image beaucoup plus claire.

Q : Lissage des Données sur les Décès liés au VIH en Colombie

Considérez ce jeu de données sur les personnes décédées du VIH en Colombie entre 2010 et 2016, provenant de [cette URL](#).

```
colom_hiv_deaths <-  
  read_csv(here("data/colombia_hiv_deaths_2010_to_2016.csv")) %>%  
  mutate(date_death = ymd(paste(death_year, death_month, death_day, sep = "-")))  
colom_hiv_deaths
```

```
## # A tibble: 445 × 26  
##   municipality_type death_location birth_date birth_year  
##   <chr>              <chr>          <date>        <dbl>  
## 1 Municipal head   Hospital/clinic 1956-05-26  1956  
## 2 Municipal head   Hospital/clinic 1983-10-10  1983  
## 3 Municipal head   Hospital/clinic 1967-11-22  1967  
## 4 Municipal head   Home/address    1964-03-14  1964  
## 5 Municipal head   Hospital/clinic 1960-06-27  1960  
## 6 Municipal head   Hospital/clinic 1982-03-23  1982  
## 7 Municipal head   Hospital/clinic 1964-12-09  1964  
## 8 Municipal head   Hospital/clinic 1975-01-15  1975  
## 9 Municipal head   Hospital/clinic 1988-02-15  1988  
## 10 Municipal head  Hospital/clinic NA            NA  
## # ... with 435 more rows,  
## # ... and 17 more variables: age_at_death <dbl>, gender <chr>, ...
```

En utilisant les étapes enseignées ci-dessus :

1. Créez un tableau qui compte les décès liés au VIH par mois.
2. Tracez une épicourbe des décès par mois
3. Appliquez `geom_smooth` à l'épicourbe pour une visualisation plus lisse. Assurez-vous de choisir une portée appropriée pour le lissage.

Lissage avec des Moyennes Mobiles

Une autre technique pour lisser les données bruyantes de séries temporelles est de calculer des **moyennes mobiles**. Cela prend la moyenne d'un nombre fixe de

points, centrée autour de chaque point de données.

La fonction `rollmean()` du package `{zoo}` sera votre principal outil pour calculer les moyennes mobiles.

Appliquons une moyenne mobile sur 14 jours pour lisser nos données quotidiennes de cas de malaria :

```
mal_notif_count <- mal_notif_count %>%
  mutate(roll_cases = rollmean(cases, k = 14, fill = NA))
mal_notif_count
```

```
## # A tibble: 406 × 3
##   date_positive_test cases roll_cases
##   <date>           <int>     <dbl>
## 1 2019-01-01          67      NA
## 2 2019-01-02         120      NA
## 3 2019-01-03         112      NA
## 4 2019-01-04         203      NA
## 5 2019-01-05          85      NA
## 6 2019-01-06           0      NA
## 7 2019-01-07         115    83.4
## 8 2019-01-08         196    82.9
## 9 2019-01-09           0    79.3
## 10 2019-01-10          89    82.1
## # ... i 396 more rows
```

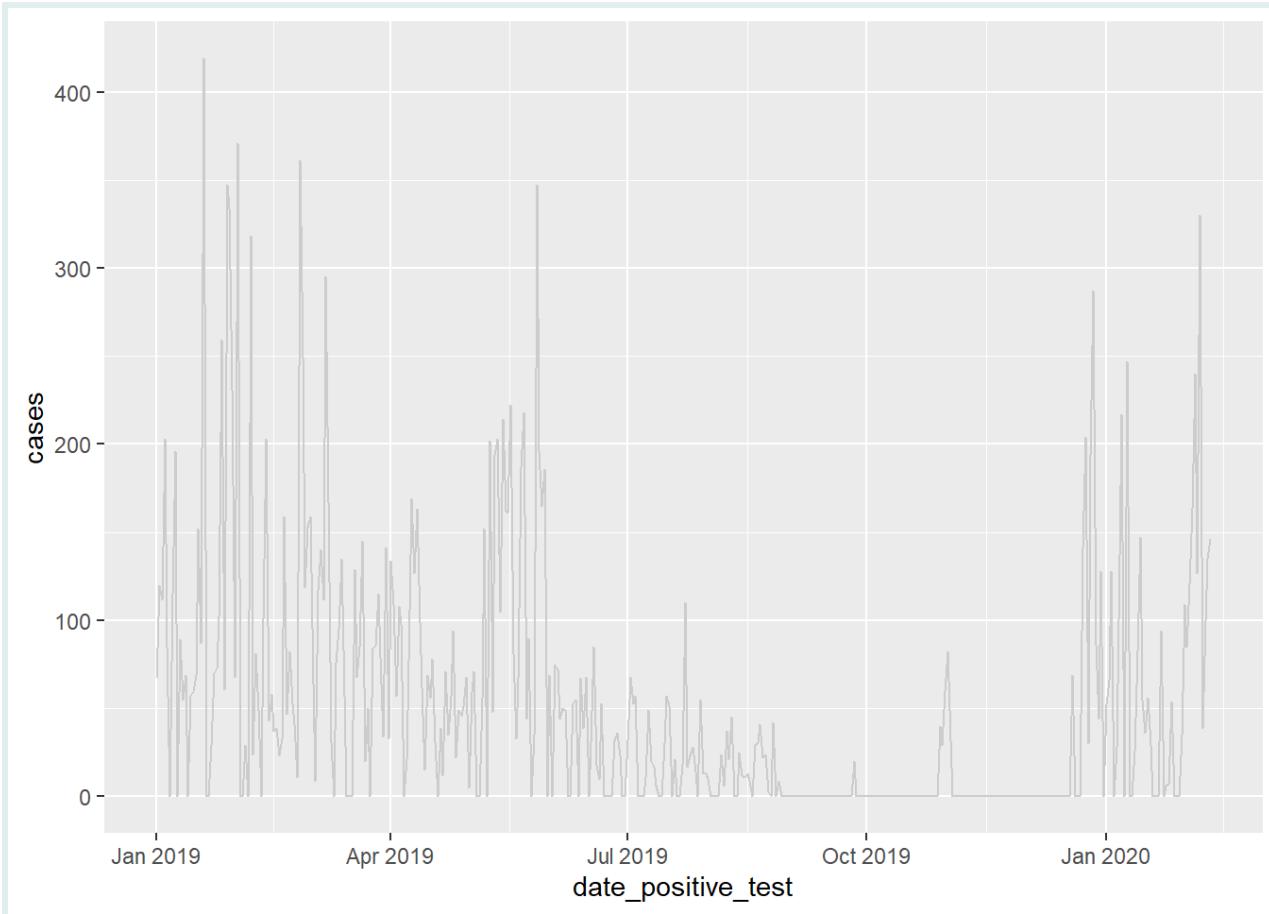
Les arguments clés sont :

- `x` : La série temporelle à lisser
- `k` : Le nombre de points avant et après pour faire la moyenne
- `fill` : Comment gérer les données manquantes dans chaque fenêtre

Cela calcule la moyenne mobile sur 14 jours, laissant les données manquantes comme NA. Notez que les 6 premiers jours sont NA, car il n'y a pas assez de points pour faire une moyenne (avec un `k` de 14, nous avons besoin de 7 jours avant et 7 jours après chaque point pour calculer la moyenne mobile).

Tracer le graphique :

```
mal_notif_count %>%
  ggplot(aes(x = date_positive_test, y = cases)) +
  geom_line(color = "gray80")
```



Souvent, on vous demandera de tracer une moyenne mobile des *dernières 1 ou 2 semaines*. Pour cela, vous devez régler l'argument `align` sur "right" :

```
mal_notif_count_right <-
  mal_notif_count %>%
    mutate(roll_cases = rollmean(cases, k = 14, fill = NA, align = "right"))

head(mal_notif_count_right, 15)
```

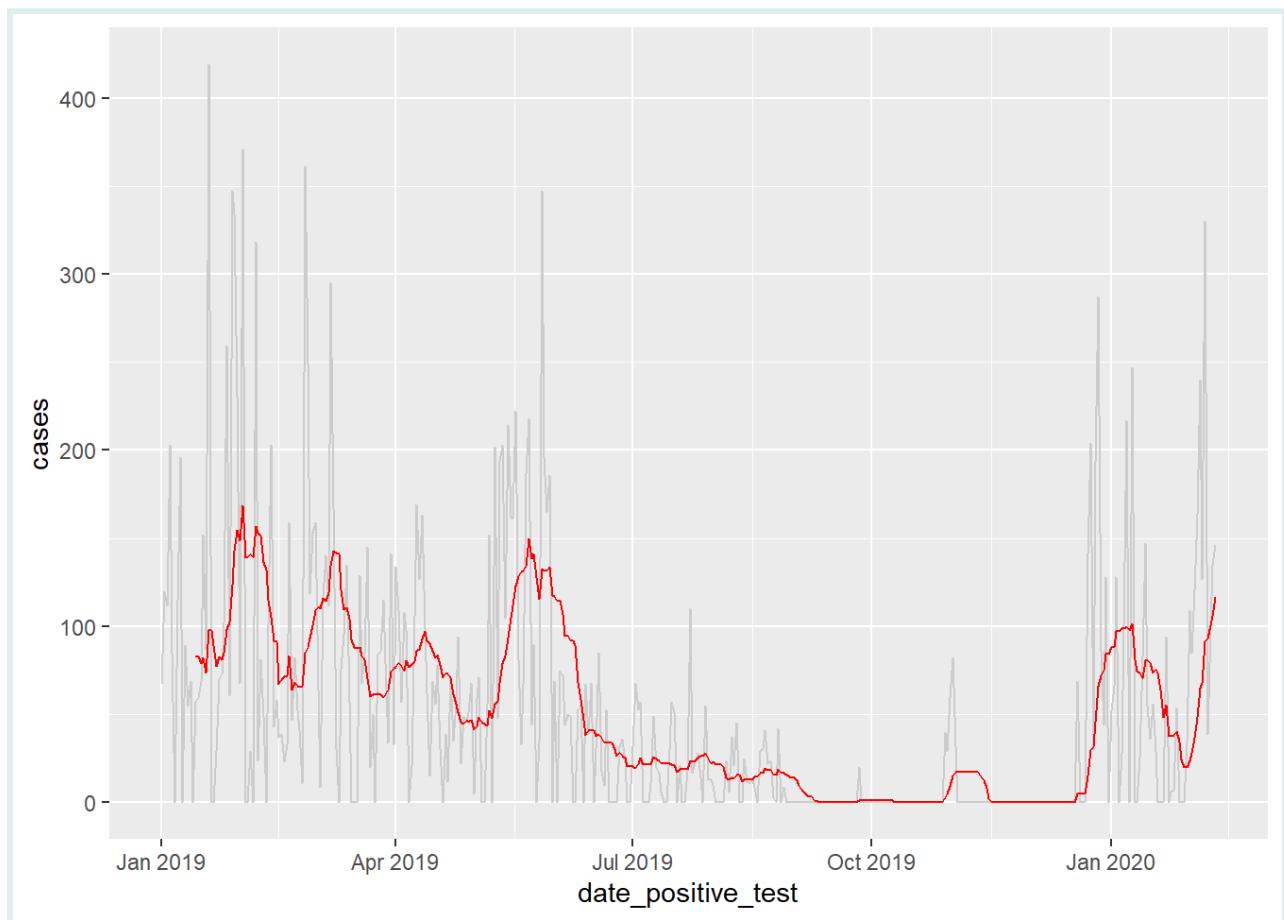
```
## # A tibble: 15 × 3
##   date_positive_test cases roll_cases
##   <date>           <int>     <dbl>
## 1 2019-01-01          67      NA
## 2 2019-01-02         120      NA
## 3 2019-01-03         112      NA
## 4 2019-01-04         203      NA
## 5 2019-01-05          85      NA
## 6 2019-01-06           0      NA
## 7 2019-01-07         115      NA
## 8 2019-01-08         196      NA
## 9 2019-01-09           0      NA
## 10 2019-01-10          89      NA
## 11 2019-01-11          55      NA
## 12 2019-01-12          69      NA
## 13 2019-01-13           0      NA
```

```
## 14 2019-01-14      57      83.4
## 15 2019-01-15      60      82.9
```

Notez que maintenant les 13 premiers jours sont NA, car il n'y a pas assez de points pour faire une moyenne. C'est parce que nous calculons la moyenne des *14 derniers jours*, et les 13 premiers jours n'ont pas 14 jours avant eux.

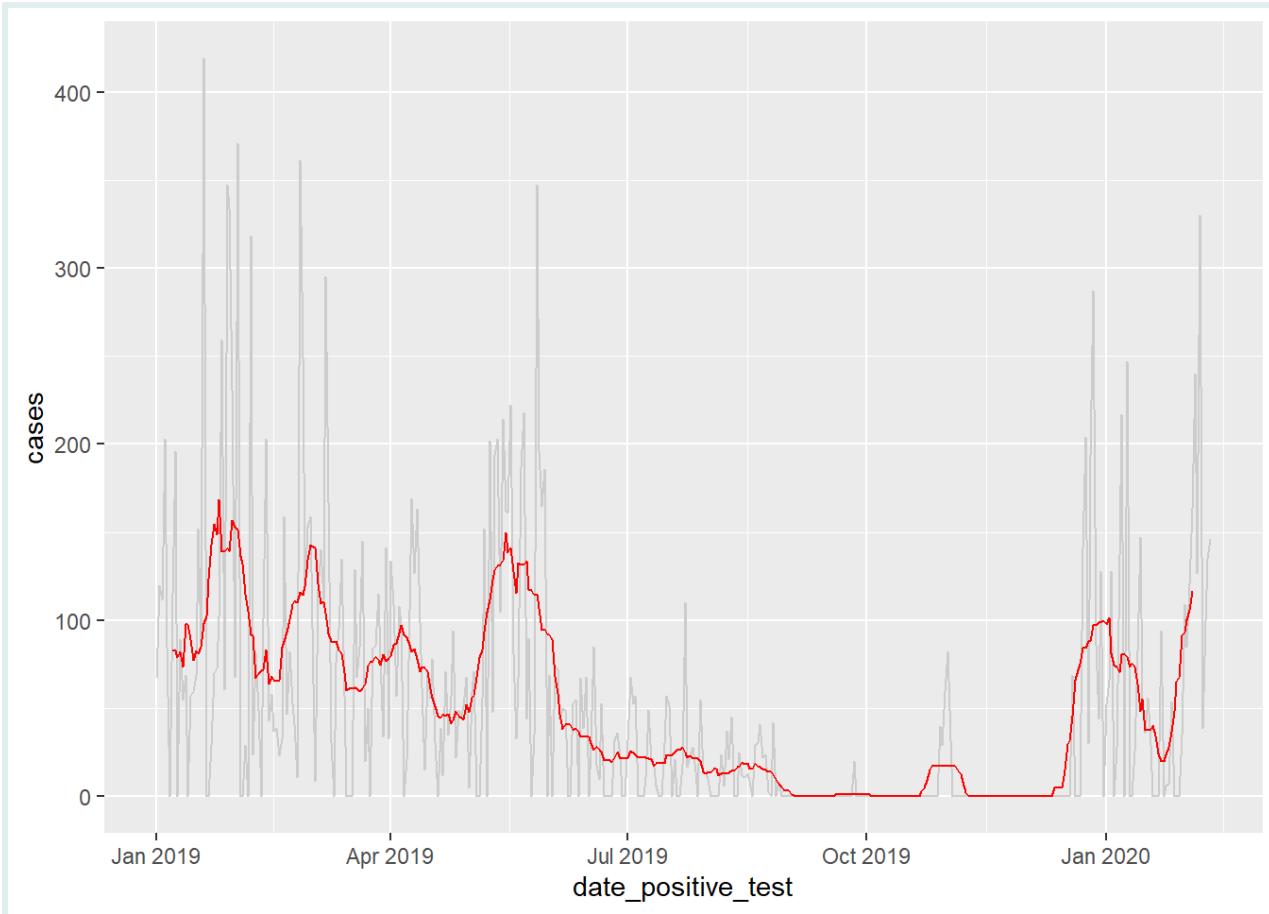
La sortie ne change pas beaucoup dans ce cas :

```
ggplot(mal_notif_count_right, aes(x = date_positive_test, y = cases)) +
  geom_line(color = "gray80") +
  geom_line(aes(y = roll_cases), color = "red")
```



Finalement, traçons à la fois les données originales et lissées :

```
mal_notif_count %>%
  ggplot(aes(x = date_positive_test, y = cases)) +
  geom_line(color = "gray80") +
  geom_line(aes(y = roll_cases), color = "red")
```



En résumé, la fonction `rollmean()` nous permet de calculer facilement une moyenne mobile sur une fenêtre fixe pour lisser et mettre en évidence les motifs dans les données bruyantes de séries temporelles.

Q : Lissage avec des moyennes mobiles

Considérez à nouveau le jeu de données sur les décès de patients VIH en Colombie :

PRACTICE



(in RMD)

```
colom_hiv_deaths

## # A tibble: 445 × 26
##   municipality_type death_location birth_date birth_year
##   <chr>              <chr>          <date>        <dbl>
## 1 Municipal head    Hospital/clinic 1956-05-26    1956
## 2 Municipal head    Hospital/clinic 1983-10-10    1983
## 3 Municipal head    Hospital/clinic 1967-11-22    1967
## 4 Municipal head    Home/address    1964-03-14    1964
```

```

## 7 Municipal head Hospital/clinic 1964-12-09 1964
## 8 Municipal head Hospital/clinic 1975-01-15 1975
## 9 Municipal head Hospital/clinic 1988-02-15 1988
## 10 Municipal head Hospital/clinic NA NA
##   birth_month birth_day death_year death_month death_day
##   <chr>        <dbl>    <dbl> <chr>      <dbl>
## 1 May           26     2012 Sep       14
## 2 Oct            10     2012 Mar       17
## 3 Nov            22     2011 Oct       19
## 4 Mar            14     2012 Nov       19
## 5 Jun            27     2012 Jan       13
## 6 Mar            23     2013 Dec       25
## 7 Dec             9     2013 Feb       21
## 8 Jan            15     2013 Dec        6
## 9 Feb            15     2013 Nov        2
## 10 <NA>          NA     2013 Oct       6
## # i 435 more rows
## # i 17 more variables: age_at_death <dbl>, gender <chr>, ...

```

Le code suivant calcule le nombre de décès par jour :



```

colom_hiv_deaths_per_day <-
  colom_hiv_deaths %>%
  group_by(date_death) %>%
  summarize(deaths = n()) %>%
  complete(date_death = seq.Date(min(date_death),
                                 max(date_death),
                                 by = "day"),
           fill = list(deaths = 0))

colom_hiv_deaths_per_day

```

```

## # A tibble: 2,543 × 2
##   date_death deaths
##   <date>      <int>
## 1 2010-01-05     1
## 2 2010-01-06     0
## 3 2010-01-07     0
## 4 2010-01-08     0
## 5 2010-01-09     1
## 6 2010-01-10     0
## 7 2010-01-11     1
## 8 2010-01-12     0
## 9 2010-01-13     0
## 10 2010-01-14    0
## # i 2,533 more rows

```



Votre tâche est de créer une nouvelle colonne qui calcule la moyenne mobile des décès par jour sur une période de 14 jours. Ensuite, tracez cette moyenne mobile aux côtés des données brutes.

Axes Secondaires

Comprendre le Concept d'un Axe Y Secondaire

Un axe y secondaire est utile lorsque l'on souhaite visualiser deux mesures différentes avec des échelles distinctes sur le même graphique. Cette approche est utile lorsque les variables ont des unités ou des magnitudes différentes, rendant la comparaison directe sur une seule échelle difficile.

Bien que certains experts en visualisation de données déconseillent l'utilisation d'axes secondaires, les décideurs en santé publique apprécient souvent ces graphiques.

Création d'un Graphique avec un Axe Y Secondaire

Démontrons comment créer un graphique avec un axe y secondaire en utilisant notre jeu de données sur les notifications de malaria :

Étape 1 : Créer des Comptes Cumulatifs de Cas

D'abord, nous allons agréger nos données sur la malaria pour calculer les comptes cumulatifs de cas.

```
mal_notif_count_cumul <-
  mal_notif_count %>%
  mutate(cumul_cases = cumsum(cases))

mal_notif_count_cumul
```

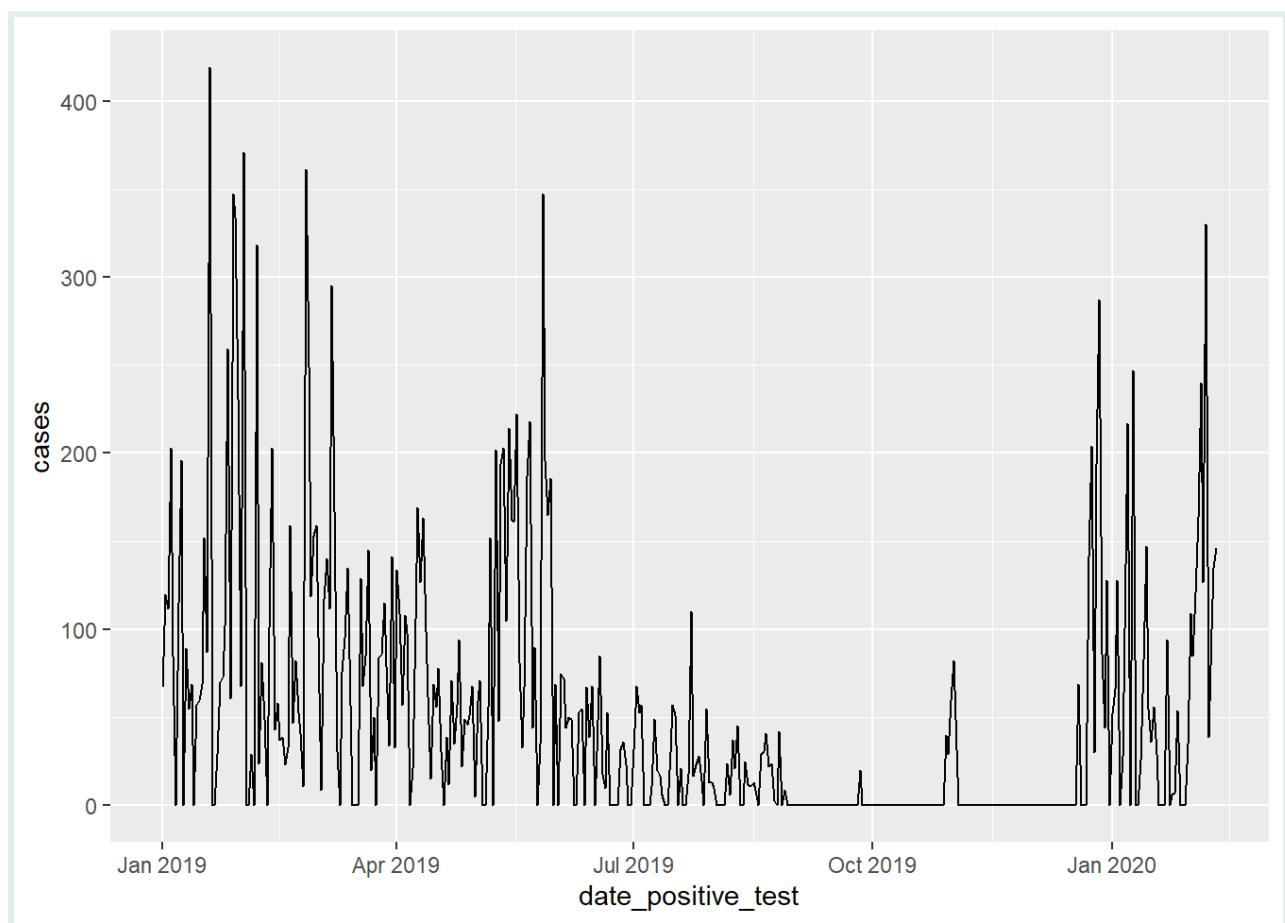
```
## # A tibble: 406 x 4
##   date_positive_test cases roll_cases cumul_cases
##   <date>           <int>    <dbl>      <int>
## 1 2019-01-01          67     NA        67
## 2 2019-01-02         120     NA       187
## 3 2019-01-03         112     NA       299
## 4 2019-01-04         203     NA       502
## 5 2019-01-05          85     NA       587
## 6 2019-01-06           0     NA       587
## 7 2019-01-07         115    83.4      702
## 8 2019-01-08         196    82.9      898
## 9 2019-01-09           0    79.3      898
```

```
## 10 2019-01-10          89        82.1       987
## # i 396 more rows
```

Étape 2 : Identifier le Besoin d'un Axe Y Secondaire

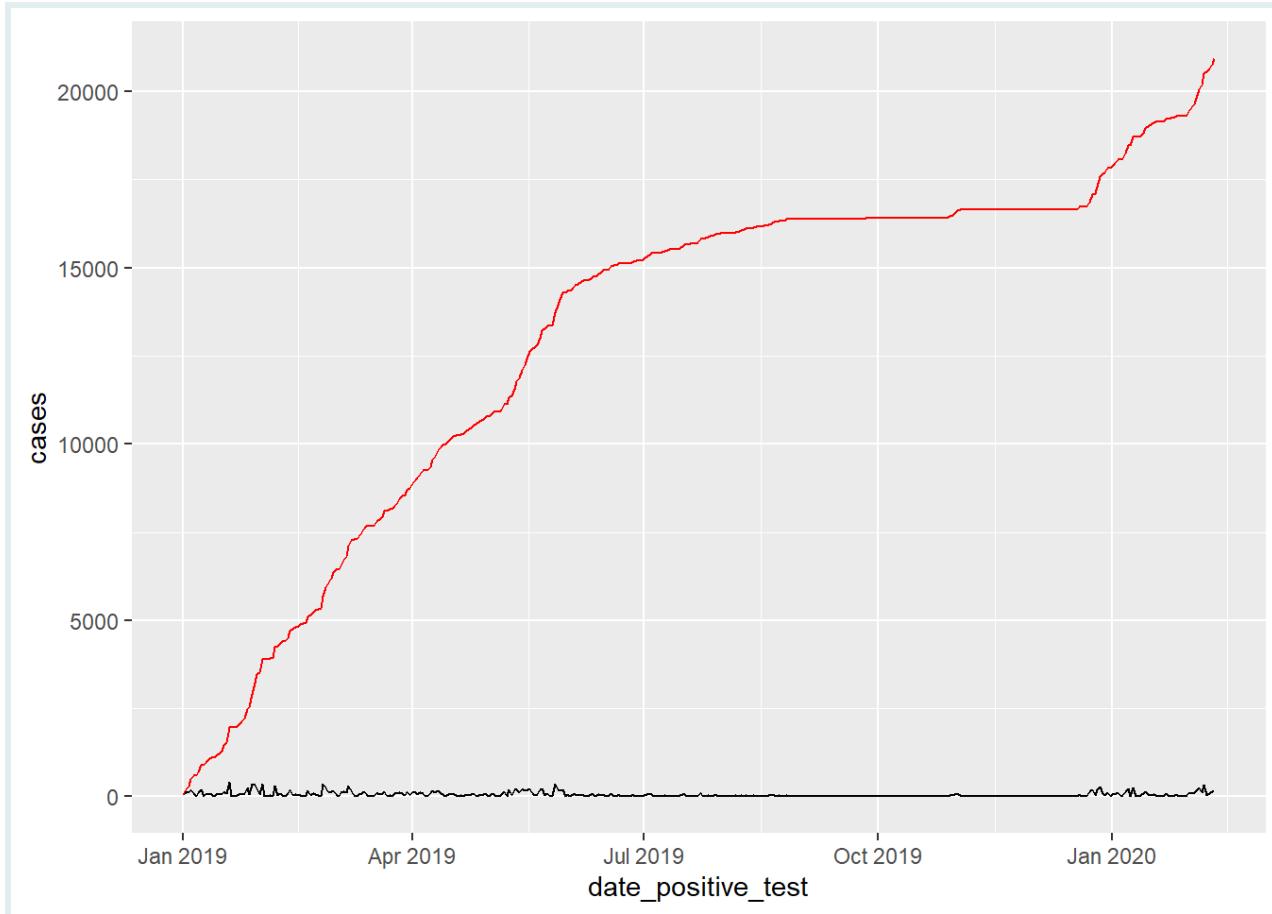
Maintenant, nous pouvons commencer à tracer. D'abord, nous traçons seulement les cas quotidiens :

```
# Tracer les cas totaux de malaria
ggplot(mal_notif_count_cumul, aes(x = date_positive_test)) +
  geom_line(aes(y = cases))
```



Si nous essayons d'ajouter des cas cumulatifs sur le même axe y, les cas quotidiens seront éclipsés et leur magnitude sera difficile à lire en raison de l'échelle beaucoup plus grande des données cumulatives :

```
# Ajouter des cas cumulatifs de malaria au graphique
ggplot(mal_notif_count_cumul, aes(x = date_positive_test)) +
  geom_line(aes(y = cases)) +
  geom_line(aes(y = cumul_cases), color = "red")
```



Pour afficher efficacement les deux ensembles de données, nous devons introduire un axe y secondaire.

Étape 3 : Calcul et Application du Facteur d’Échelle

Avant d’ajouter un axe y secondaire, nous devons déterminer un *facteur d’échelle* en comparant les gammes de cas et de cas cumulatifs.

Le facteur d’échelle est généralement le rapport des valeurs maximales des deux ensembles de données. Voyons quelles sont les valeurs maximales pour chaque variable :

```
max(mal_notif_count_cumul$cases)
```

```
## [1] 419
```

```
max(mal_notif_count_cumul$cumul_cases)
```

```
## [1] 20939
```

Avec un maximum d'environ 20000 pour les cas cumulatifs, et environ 400 pour les cas quotidiens, nous pouvons voir que les cas cumulatifs sont environ 50 fois plus importants que les cas quotidiens, donc notre facteur d'échelle sera d'environ 50.

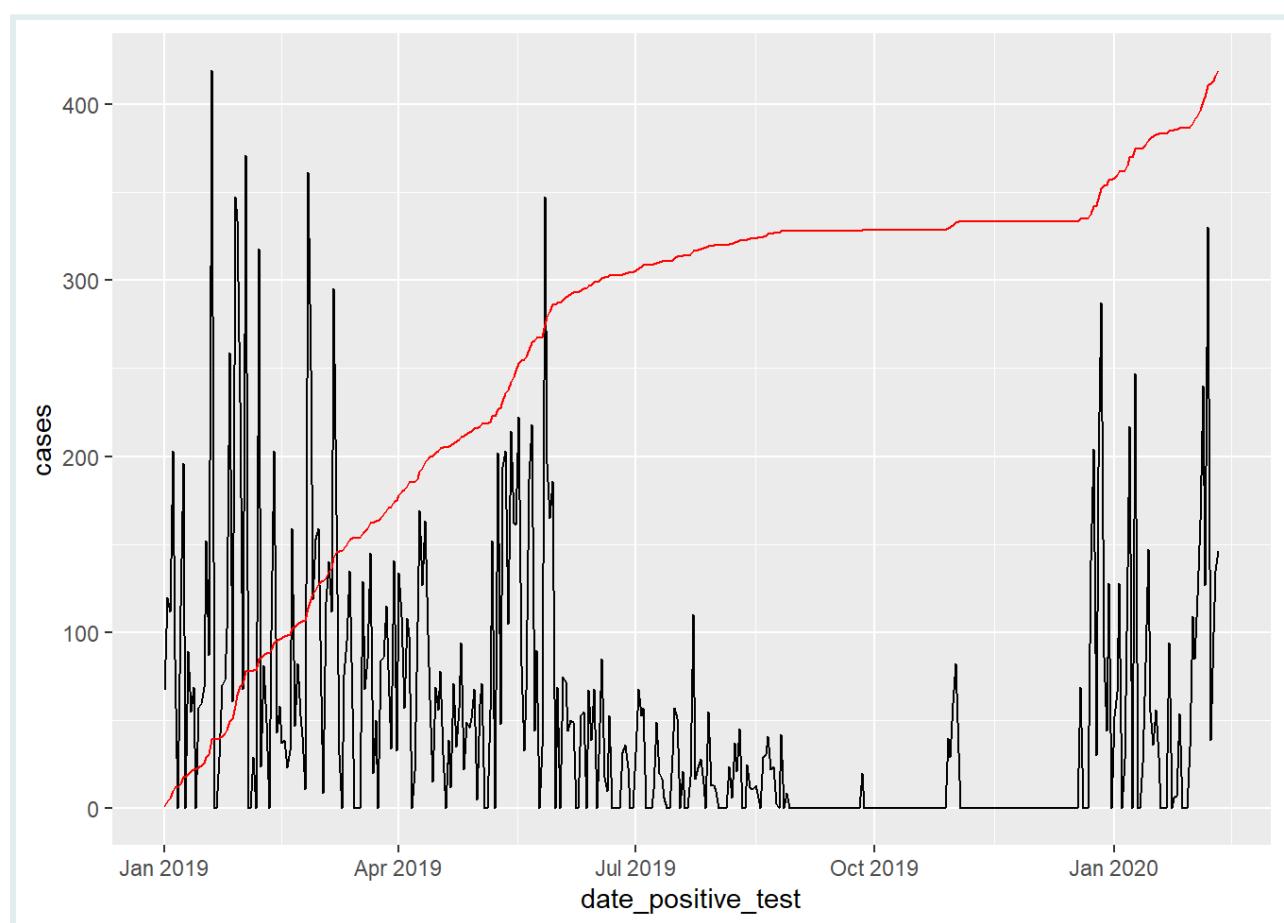
Plus précisément, le facteur d'échelle sera :

```
max(mal_notif_count_cumul$cumul_cases) / max(mal_notif_count_cumul$cases)
```

```
## [1] 49.97375
```

Nous devrons diviser les cas cumulatifs par ce ratio pour forcer les deux variables à être sur une échelle similaire :

```
ggplot(mal_notif_count_cumul, aes(x = date_positive_test)) +  
  geom_line(aes(y = cases)) +  
  geom_line(aes(y = cumul_cases / 49.97), color = "red") # diviser par le  
  facteur d'échelle
```



Super ! Maintenant, nous pouvons clairement voir les deux ensembles de données sur le même graphique, et leurs points maximaux sont alignés. Cependant, l'axe y n'est plus pertinent pour la ligne des cas cumulatifs en rouge. Nous devons ajouter un axe y secondaire pour cela.

SIDE NOTE

Normalement, vous assigneriez le facteur d'échelle à une variable, puis utiliseriez cette variable dans la fonction `geom_line()`. Dans ce cas, nous l'écrivons directement dans la fonction pour une meilleure compréhension.

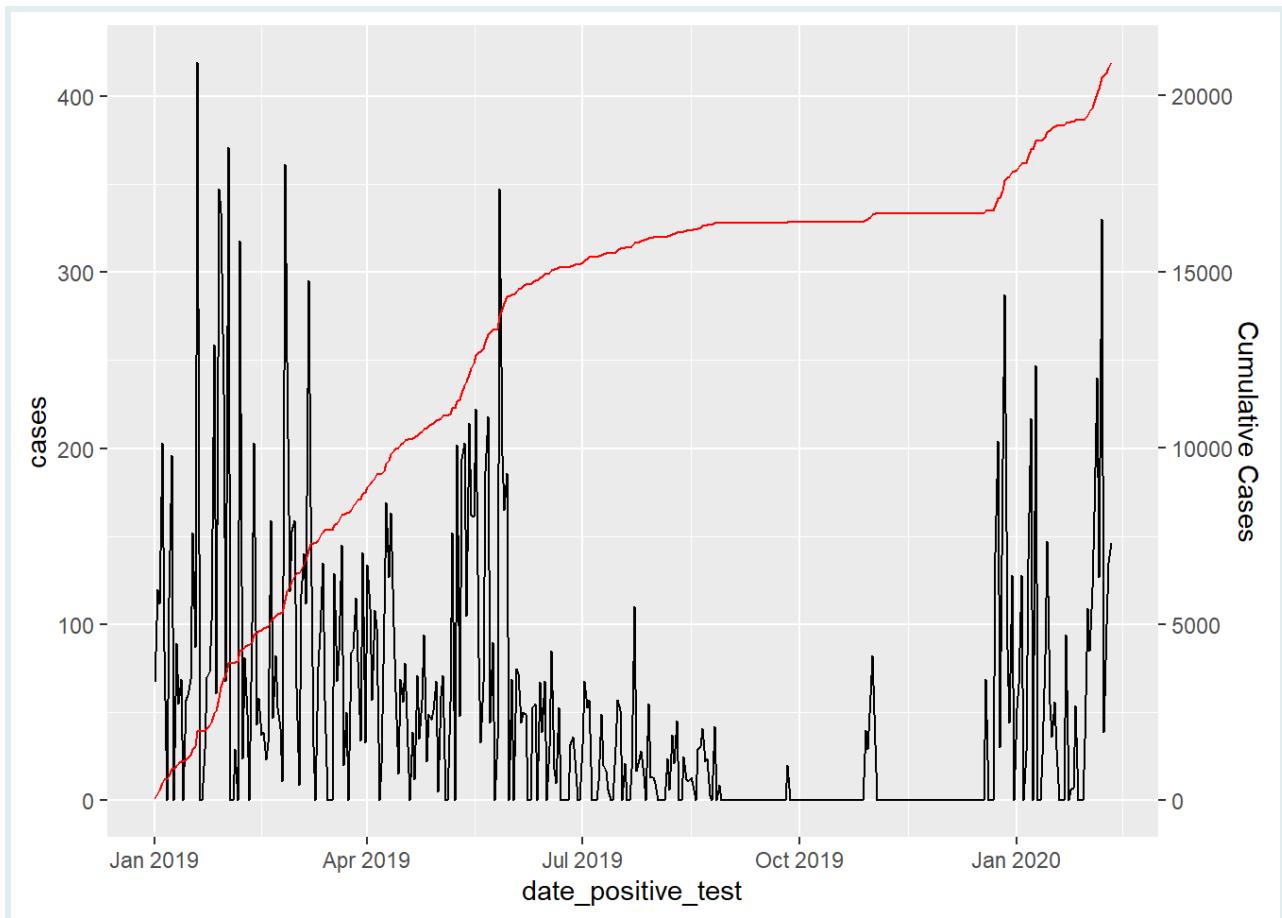
Étape 4 : Ajout de l'Axe Y Secondaire

Nous utiliserons la fonction `sec_axis()` de `{ggplot2}`. Les arguments clés sont `trans`, qui indique combien multiplier ou diviser l'axe y original, et `name`, qui spécifie le nom de l'axe secondaire.

Dans notre cas, nous voulons que l'axe secondaire soit environ 49.97 fois plus grand que l'axe original, donc nous utiliserons `trans = ~ .x * 49.97`. (Le symbole `~` est un opérateur spécial qui indique à R de traiter l'expression qui suit comme une fonction, dont l'entrée est indiquée par le symbole `.x`.)

Mettre cela en œuvre :

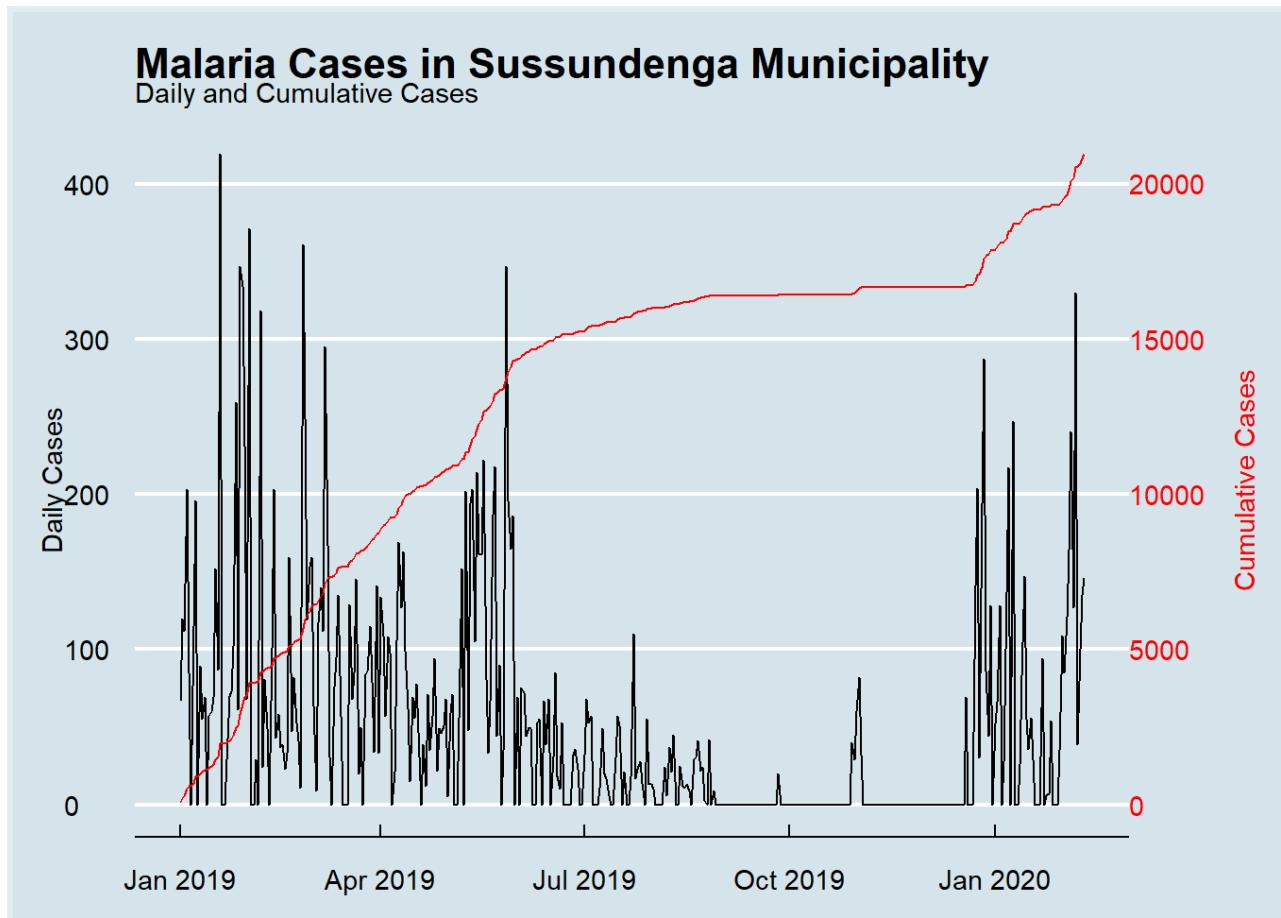
```
# Ajouter un axe y secondaire
ggplot(mal_notif_count_cumul, aes(x = date_positive_test)) +
  geom_line(aes(y = cases)) +
  geom_line(aes(y = cumul_cases / 49.97), color = "red") +
  scale_y_continuous(sec.axis = sec_axis(trans = ~ .x * 49.97,
                                         name = "Cumulative Cases"))
```



Étape 5 : Améliorer la Lisibilité du Graphique

Pour améliorer la lisibilité, nous allons faire correspondre les étiquettes de l'axe secondaire en rouge, assorties à la couleur de la ligne des cas cumulatifs, et nous ajouterons un peu de formatage supplémentaire au graphique :

```
# Finaliser le graphique avec des axes coordonnés en couleur
ggplot(mal_notif_count_cumul, aes(x = date_positive_test)) +
  geom_line(aes(y = cases)) +
  geom_line(aes(y = cumul_cases / 49.97), color = "red") +
  scale_y_continuous(
    name = "Daily Cases",
    sec.axis = sec_axis(~ . * 49.97, name = "Cumulative Cases")
  ) +
  labs(title = "Malaria Cases in Sussundenga Municipality",
       subtitle = "Daily and Cumulative Cases",
       x = NULL) +
  theme_economist() +
  theme(axis.text.y.right = element_text(color = "red"),
        axis.title.y.right = element_text(color = "red"))
```



Voilà ! Nous avons avec succès ajouté un axe y secondaire à un graphique, permettant la comparaison de deux ensembles de données avec des échelles différentes dans une seule visualisation.

Q : Axes secondaires

Revenez sur le jeu de données `colom_hiv_deaths_per_day`.

```
colom_hiv_deaths_per_day
```

```
## # A tibble: 2,543 x 2
##   date_death deaths
##   <date>      <int>
## 1 2010-01-05     1
## 2 2010-01-06     0
## 3 2010-01-07     0
## 4 2010-01-08     0
## 5 2010-01-09     1
## 6 2010-01-10     0
## 7 2010-01-11     1
## 8 2010-01-12     0
## 9 2010-01-13     0
```

```
## 10 2010-01-14      0  
## # i 2,533 more rows
```

Votre tâche est de créer un graphique avec deux axes y : un pour les décès quotidiens et un autre pour les décès cumulatifs en Colombie.

Conclusion !

- Ce cours a fourni une introduction complète à la visualisation des données de séries temporelles à l'aide de graphiques en ligne, en mettant l'accent sur les applications pratiques dans les contextes de santé publique. En maîtrisant ces techniques, vous êtes maintenant équipé pour communiquer efficacement les tendances des données complexes et informer la prise de décision basée sur les données dans les domaines de la santé et connexes.*
- Visualiser les notifications de TB au fil du temps.
- Surmonter les défis de la conversion des données trimestrielles en format annuel.
- Créer des Graphiques en Ligne Basiques et Groupés en employant des méthodes de remodelage des données comme `pivot_longer()`.
- Améliorer les tracés avec des techniques de gestion des étiquettes et de personnalisation des couleurs comme `ggrepel::geom_text_repel()` pour une clarté dans le placement des étiquettes.
- Apprendre l'importance de représenter la variabilité à travers des intervalles de confiance dans les données avec `geom_ribbon()`.
- Lisser les données bruyantes avec `geom_smooth()` et des moyennes mobiles pour une visualisation des tendances plus claire.
- Maîtriser l'utilisation d'un axe y secondaire pour comparer des mesures distinctes.

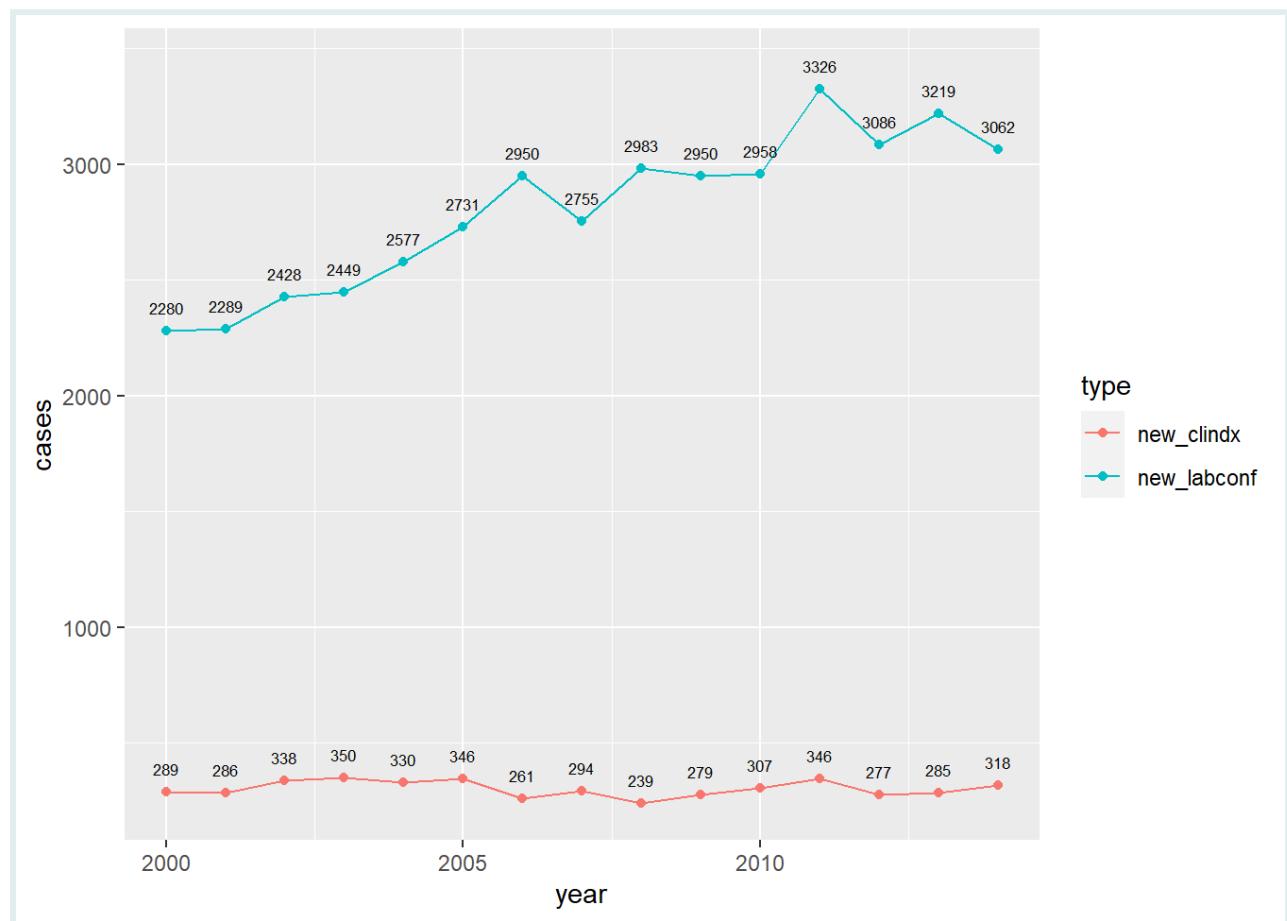
Prochaines Étapes : Continuez à explorer des techniques de visualisation avancées et appliquez ces compétences à divers ensembles de données pour approfondir votre compréhension et votre maîtrise de l'analyse et de la visualisation des données.

Solutions

Q: Mise en forme et représentation des données sur la tuberculose

```
tb_benin_long <- tb_data_benin %>%
  pivot_longer(cols = c("new_clindx", "new_labconf")) %>%
  rename(type = name, cases = value)

ggplot(tb_benin_long, aes(x = year, y = cases, colour = type, group = type)) +
  geom_line() +
  geom_point() +
  geom_text(aes(label = cases), size = 2.2, nudge_y = 100, color = "black")
```



Q: Améliorations esthétiques

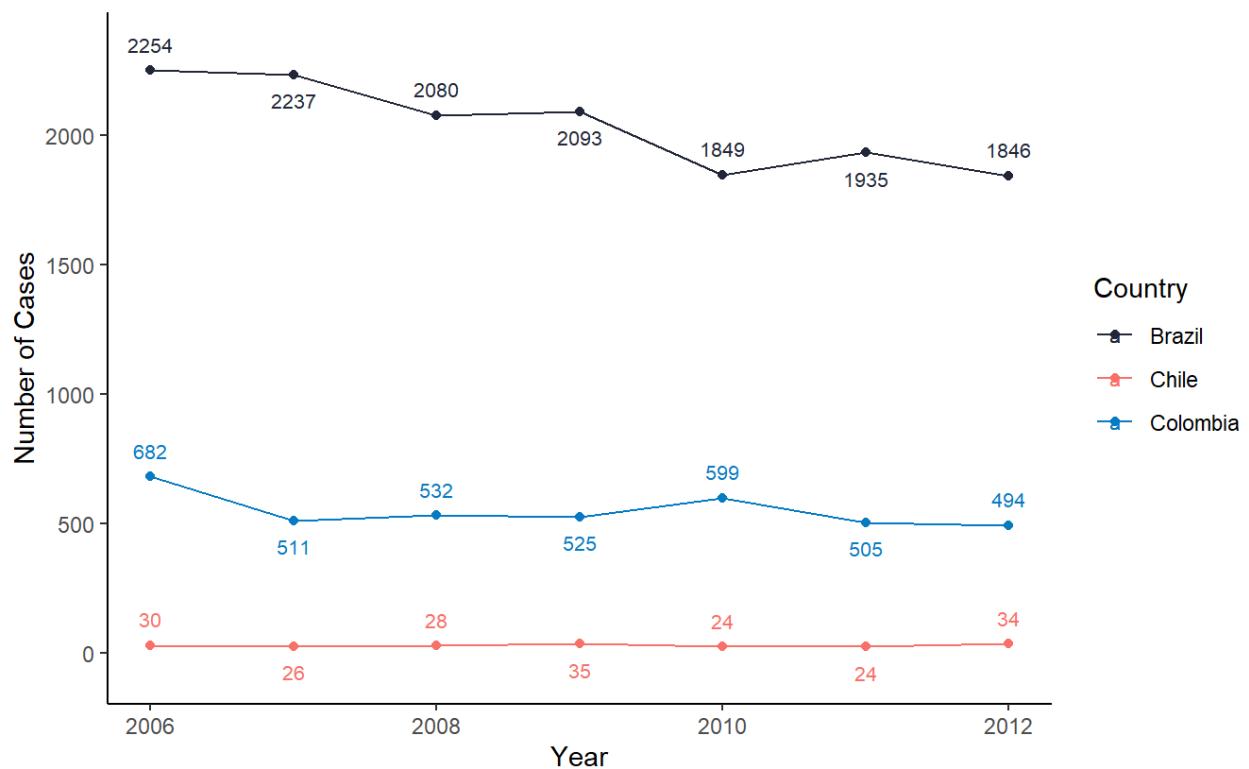
```
even_years_southam <- tb_child_cases_southam %>%
  filter(year %% 2 == 0) # Keep only years that are multiples of 2

odd_years_southam <- tb_child_cases_southam %>%
  filter(year %% 2 == 1) # Keep only years that are not multiples of 2

tb_child_cases_southam %>%
  ggplot(aes(x = year, y = tb_cases_children, color = country)) +
  geom_line() +
  geom_point() +
  geom_text(data = even_years_southam, aes(label = tb_cases_children),
            nudge_y = 100, size = 2.8) +
  geom_text(data = odd_years_southam, aes(label = tb_cases_children),
            nudge_y = -100, size = 2.8) +
  scale_color_manual(values = c("#212738", "#F97068", "#067BC2")) +
  labs(title = "Tuberculosis Notifications in Three South American Countries",
       subtitle = "Child Cases, 1993-2022",
       caption = "Source: World Health Organization",
       x = "Year",
       y = "Number of Cases",
       color = "Country") +
  theme_classic()
```

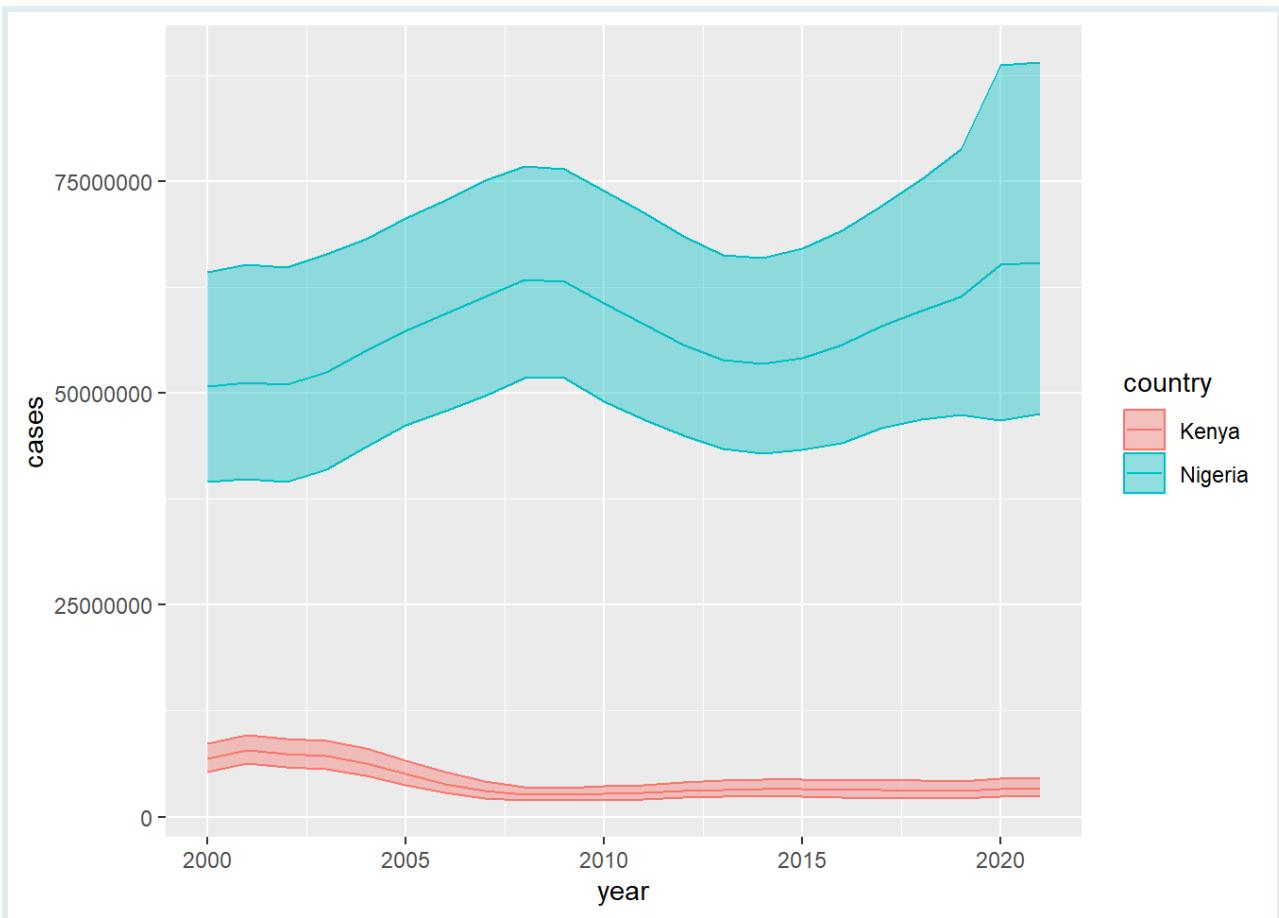
Tuberculosis Notifications in Three South American Countries

Child Cases, 1993-2022



Q: Représentation des intervalles de confiance

```
nig_ken_mal %>%
  separate(malaria_cases,
           into = c("cases", "cases_lower", "cases_upper"),
           sep = "\\\\(|to") %>%
  mutate(across(c("cases", "cases_lower", "cases_upper"),
               ~ str_replace_all(.x, "[^0-9]", "") %>%
                 as.numeric())
        )) %>%
  ggplot(aes(x = year, y = cases, color = country, fill = country)) +
  geom_line() +
  geom_ribbon(aes(ymin = cases_lower, ymax = cases_upper), alpha = 0.4)
```



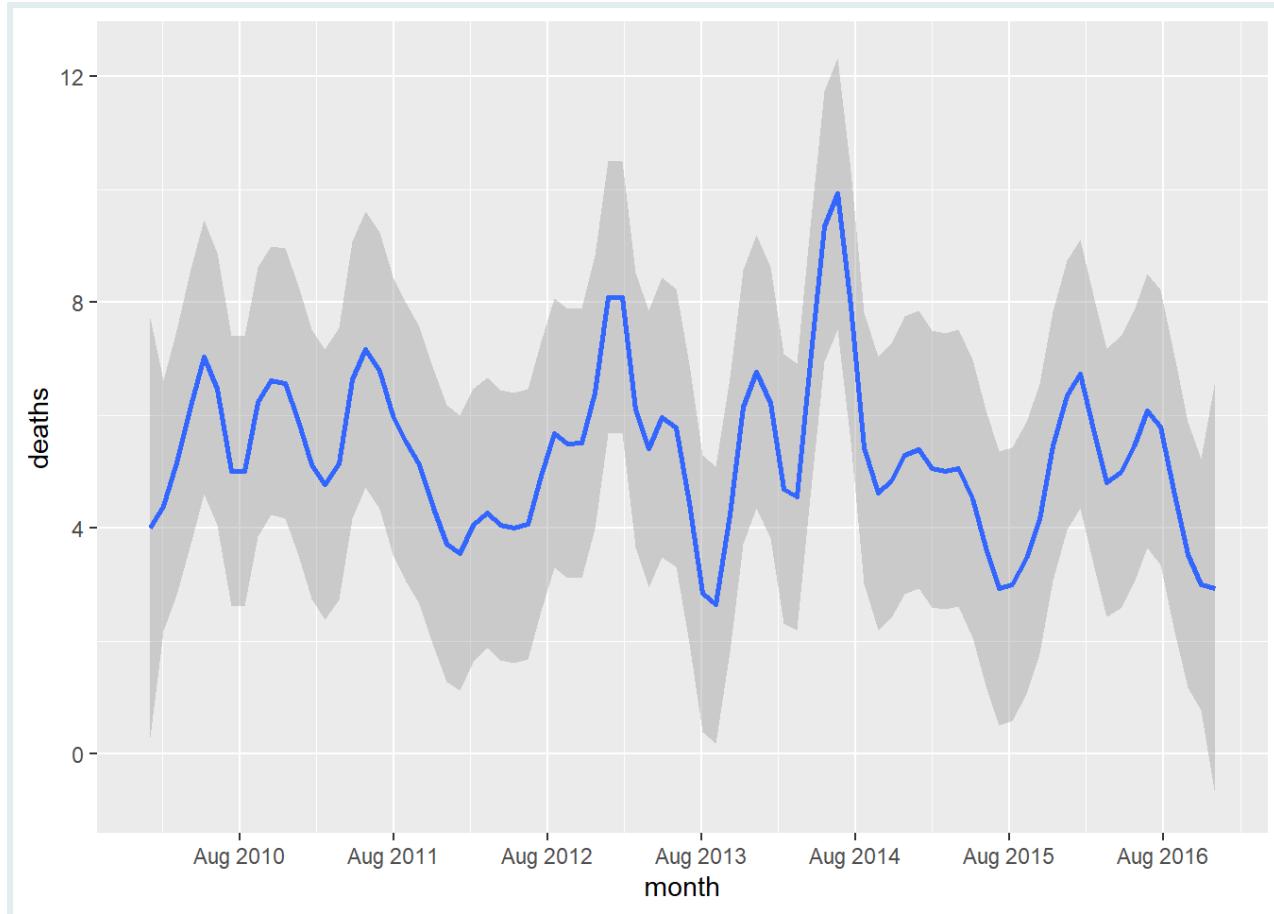
Q: Lissage des données sur les décès liés au VIH en Colombie

```

hiv_monthly_deaths_table <-
  colom_hiv_deaths %>%
  # Aggregate data to count deaths per month
  mutate(month = floor_date(date_death, unit = "month")) %>%
  group_by(month) %>%
  summarize(deaths = n())

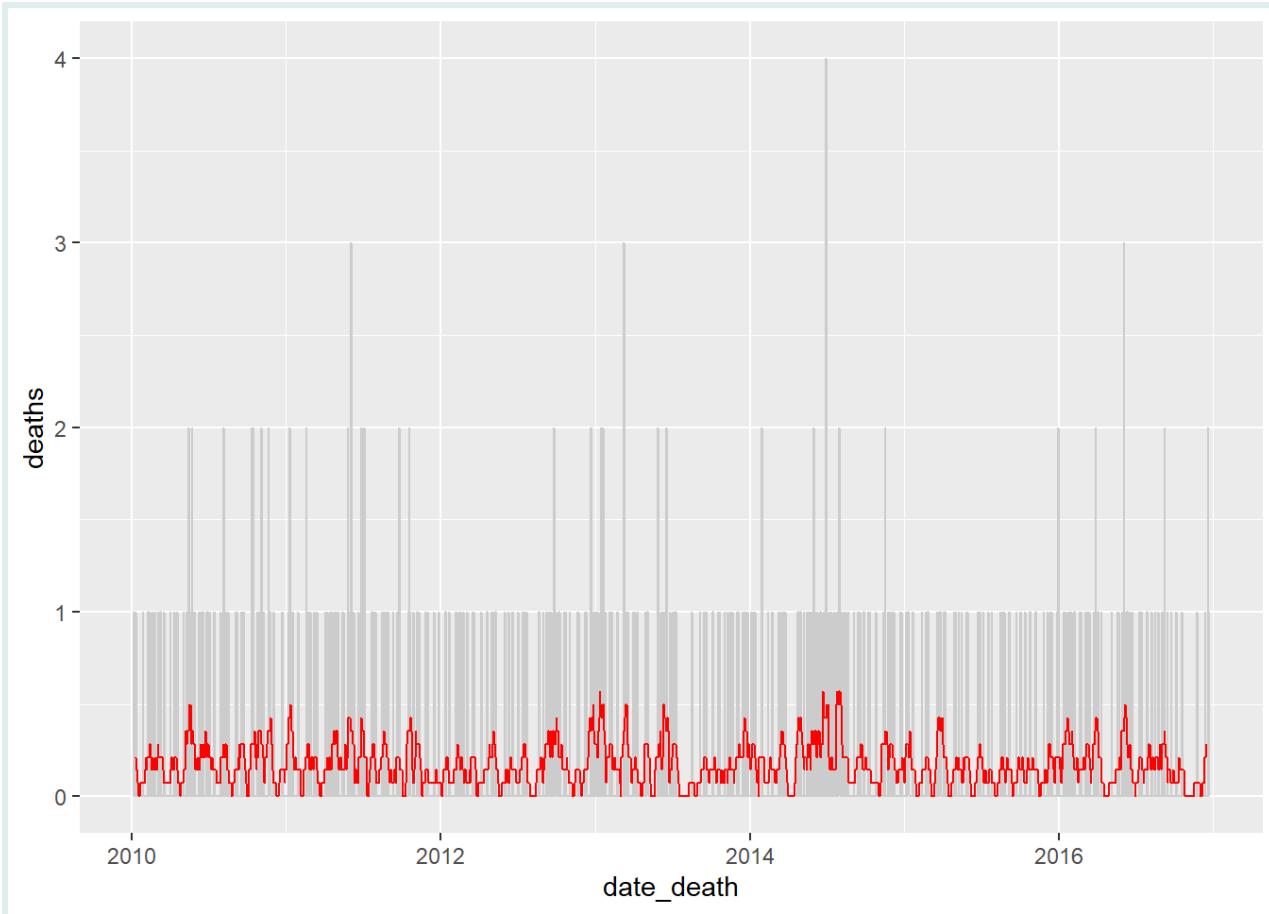
# Create the epicurve
ggplot(hiv_monthly_deaths_table, aes(x = month, y = deaths)) +
  # Apply smoothing to the curve
  geom_smooth(method = "loess", span = 0.1) +
  scale_x_date(date_breaks = "12 months", date_labels = "%b %Y")

```



Q: Lissage avec des moyennes mobiles

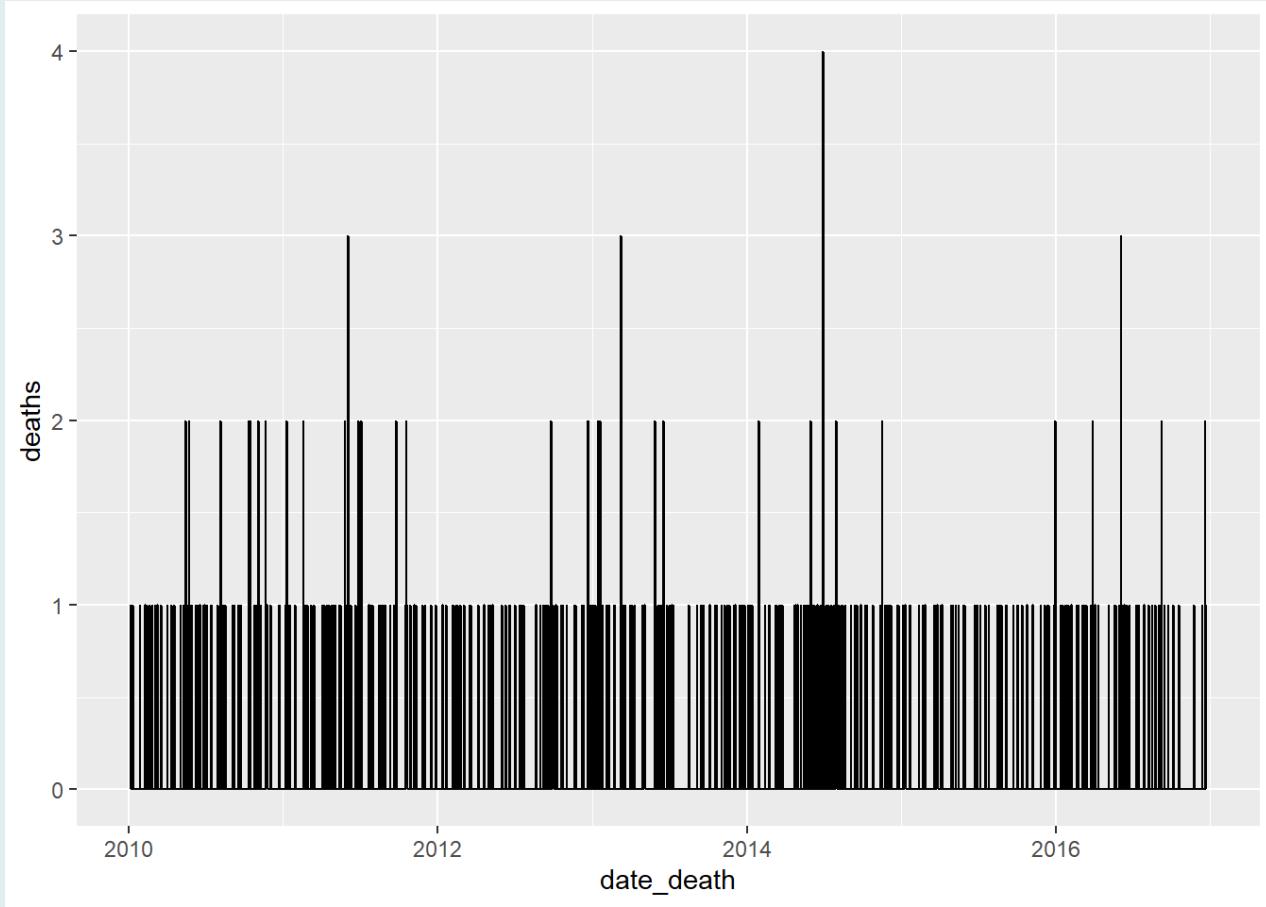
```
colom_hiv_deaths_per_day %>%
  mutate(roll_deaths = rollmean(deaths, k = 14, fill = NA)) %>%
  ggplot(aes(x = date_death, y = deaths)) +
  geom_line(color = "gray80") +
  geom_line(aes(y = roll_deaths), color = "red")
```



Q: Axes secondaires

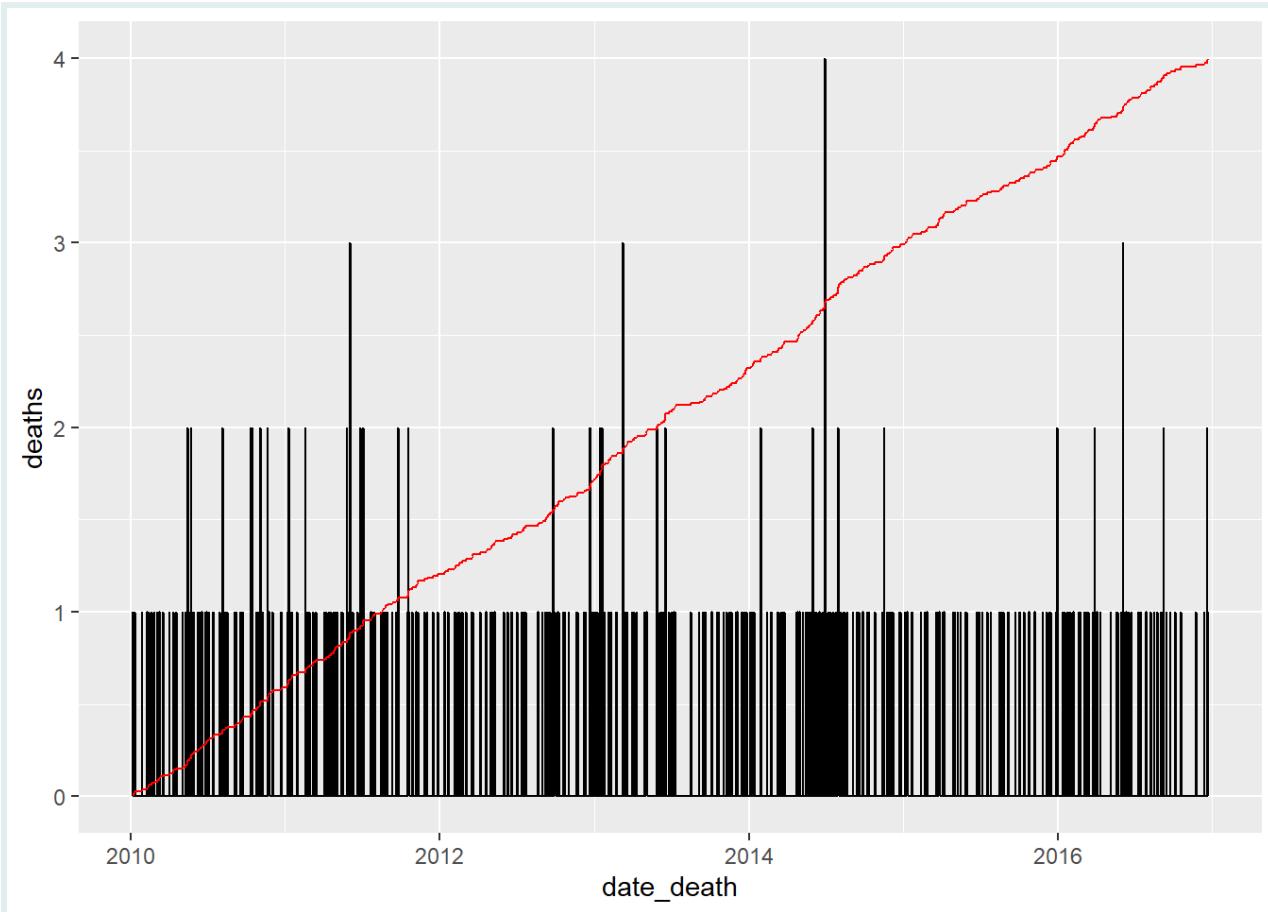
```
# Step 1: Calculate cumulative deaths
colom_hiv_deaths_cumul <- colom_hiv_deaths_per_day %>%
  mutate(cum_deaths = cumsum(deaths))

# Step 2: Plot daily deaths
ggplot(colom_hiv_deaths_cumul, aes(x = date_death)) +
  geom_line(aes(y = deaths))
```

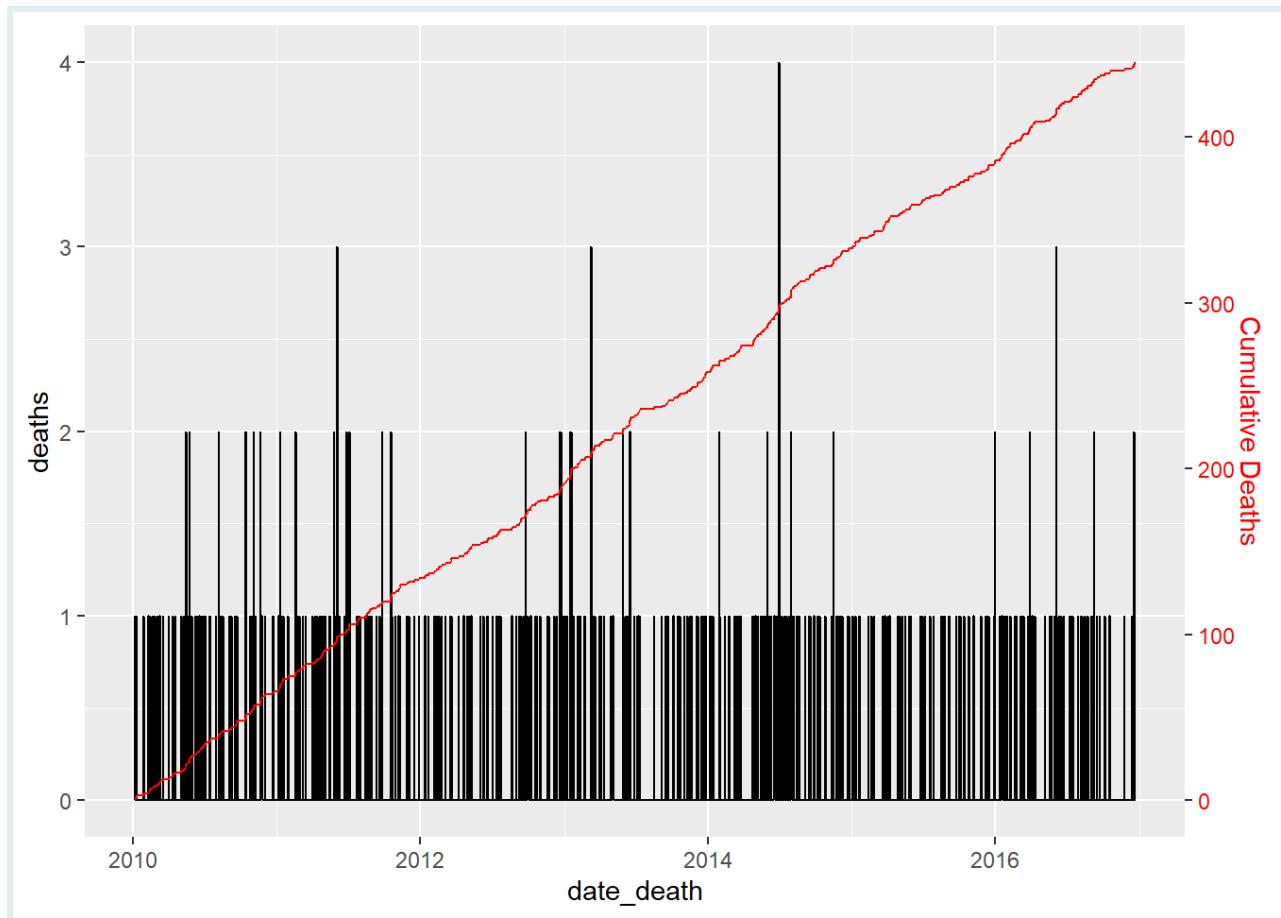


```
# Step 3: Calculate scale factor
scale_factor <- max(colom_hiv_deaths_cumul$cum_deaths) /
  max(colom_hiv_deaths_cumul$deaths)

# Step 4: Add cumulative deaths to the plot
ggplot(colom_hiv_deaths_cumul, aes(x = date_death)) +
  geom_line(aes(y = deaths)) +
  geom_line(aes(y = cum_deaths / scale_factor), color = "red")
```



```
# Step 5: Add secondary y-axis
ggplot(colom_hiv_deaths_cumul, aes(x = date_death)) +
  geom_line(aes(y = deaths)) +
  geom_line(aes(y = cum_deaths / scale_factor), color = "red") +
  scale_y_continuous(sec.axis = sec_axis(trans = ~ .x * scale_factor, name =
    "Cumulative Deaths")) +
  theme(axis.text.y.right = element_text(color = "red"),
    axis.title.y.right = element_text(color = "red"))
```

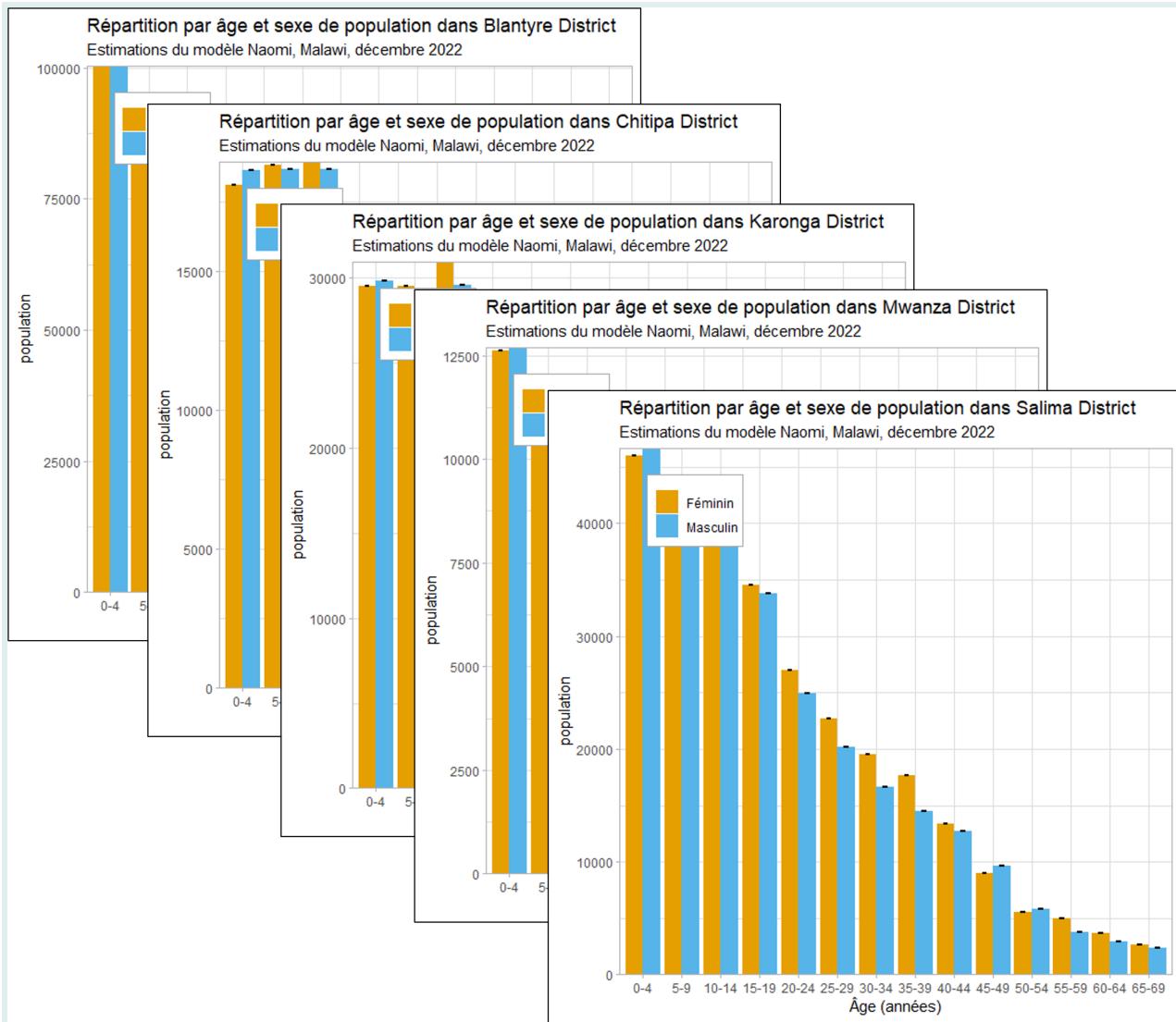


Automatisation de la visualisation de données
Introduction
Packages
Introduction aux données : le VIH au Malawi
Le défi de la création de graphiques répétitifs
Création de fonctions de création de graphiques personnalisées
Bouclage à travers un vecteur de variables
Finalisation et enregistrement
EN RÉSUMÉ !

Automatisation de la visualisation de données

Introduction

Il arrive souvent de devoir effectuer des tâches de création de graphiques de manière répétitive. Par exemple, vous souhaitez tracer le même type de données (par exemple le même indicateur d'épidémie) pour plusieurs États, provinces ou villes. Dans cette leçon, nous allons explorer comment automatiser les visualisations de données à l'aide de la puissante combinaison de `{ggplot2}` et `{purrr}` en R. Tout d'abord, nous plongerons dans le côté programmatique de `{ggplot2}`, en créant des fonctions de création de graphiques personnalisées pour rationaliser vos tâches de graphisme. Ensuite, nous utiliserons `{purrr}` pour itérer à travers différentes variables, ce qui nous permettra de générer et d'enregistrer une multitude de graphiques d'un seul coup! Apprendre à automatiser la création de graphiques améliorera grandement vos flux de travail d'analyse de données.



Objectifs d'apprentissage

- 1. Reconnaître les problèmes du filtrage et de la création de graphiques répétitifs :** Identifier quand la création de graphiques répétitive est nécessaire et créer un workflow impliquant la création de sous-ensemble de données, la création de graphiques, et l'enregistrement.
- 2. Crée des fonctions de création de graphiques personnalisées:** Développer des fonctions personnalisées pour les tâches de création de graphiques répétitives, y compris le sous-ensemble de lignes et de variables. Apprendre à ajouter plusieurs entrées pour une personnalisation dynamique des graphiques.
- 3. Itérer les tâches de création de graphiques:** Appliquer les fonctions de création de graphiques sur un vecteur de variables, avec l'aide de `purrr::map()`.

4. Utiliser des boucles imbriquées pour l'automatisation : Utiliser `map()` à l'intérieur d'une boucle `for` pour itérer sur une combinaison de sous-ensembles et de variables de réponse. À la fin de cette leçon, vous maîtriserez l'automatisation des graphiques `{ggplot2}`, gagnant du temps et améliorant la reproductibilité de vos récits basés sur les données.

À la fin de cette leçon, vous aurez les compétences pour automatiser les graphiques `{ggplot2}`, ce qui va vous gagner du temps et améliorer la reproductibilité de vos récits basés sur les données.

Packages

Dans cette leçon, nous utiliserons les packages suivants :

- `{tidyverse}`, métapackage
 - `{ggplot2}` pour créer des graphiques
 - `{purrr}` pour itérer les fonctions à travers un vecteur
- `{here}` pour les chemins de fichiers relatifs au projet
- `{glue}` pour la concaténation de chaînes et l'automatisation de l'annotation des graphiques

```
# Chargement des packages
pacman::p_load(tidyverse, here, glue)
```

Introduction aux données : le VIH au Malawi

Aujourd'hui, nous allons examiner un jeu de données d'indicateurs épidémiologiques du VIH à l'échelon infranational du Ministère de la Santé du Malawi, Département du VIH et de l'Hépatite virale, pour décembre 2022. Ces estimations ont été dérivées d'un modèle d'estimation de petite zone, appelé Naomi, pour estimer des mesures clés stratifiées par unités administratives infranationales, sexe et groupes d'âge par tranche de 5 ans. Le jeu de données original peut être consulté [ici](#).

Nous avons préparé un sous-ensemble de ces données à analyser dans cette leçon :

```
# Importation des données depuis CSV
hiv_mwi_agesex <- read_csv(here("data/clean/hiv_mwi_agesex.csv"))

# Affichage du dataframe de données
hiv_mwi_agesex
```

- **Zone géographique :**
 - `area_level` - unité administrative (pays, région ou district)

- area_name - nom de la zone géographique
- **Informations démographiques**
 - age_group et sex
 - **Indicateurs du VIH** : population totale, personnes vivant avec le VIH (PVVIH), prévalence du VIH, incidence, couverture du TAR, PVVIH connaissant leur statut.
 - indicator - code court
 - indicator_label - nom complet
- **Mesures statistiques** : estimations du modèle avec incertitude probabiliste
 - mean, lower, upper

KEY POINT



Le modèle Naomi synthétise des données de multiples sources pour fournir des estimations en petite zone des principaux indicateurs du VIH pour l'Afrique subsaharienne. Ces estimations sont essentielles à la planification des programmes du VIH, à l'allocation des ressources et à la définition des objectifs. Vous pouvez en savoir plus sur le modèle Naomi [ici](#).

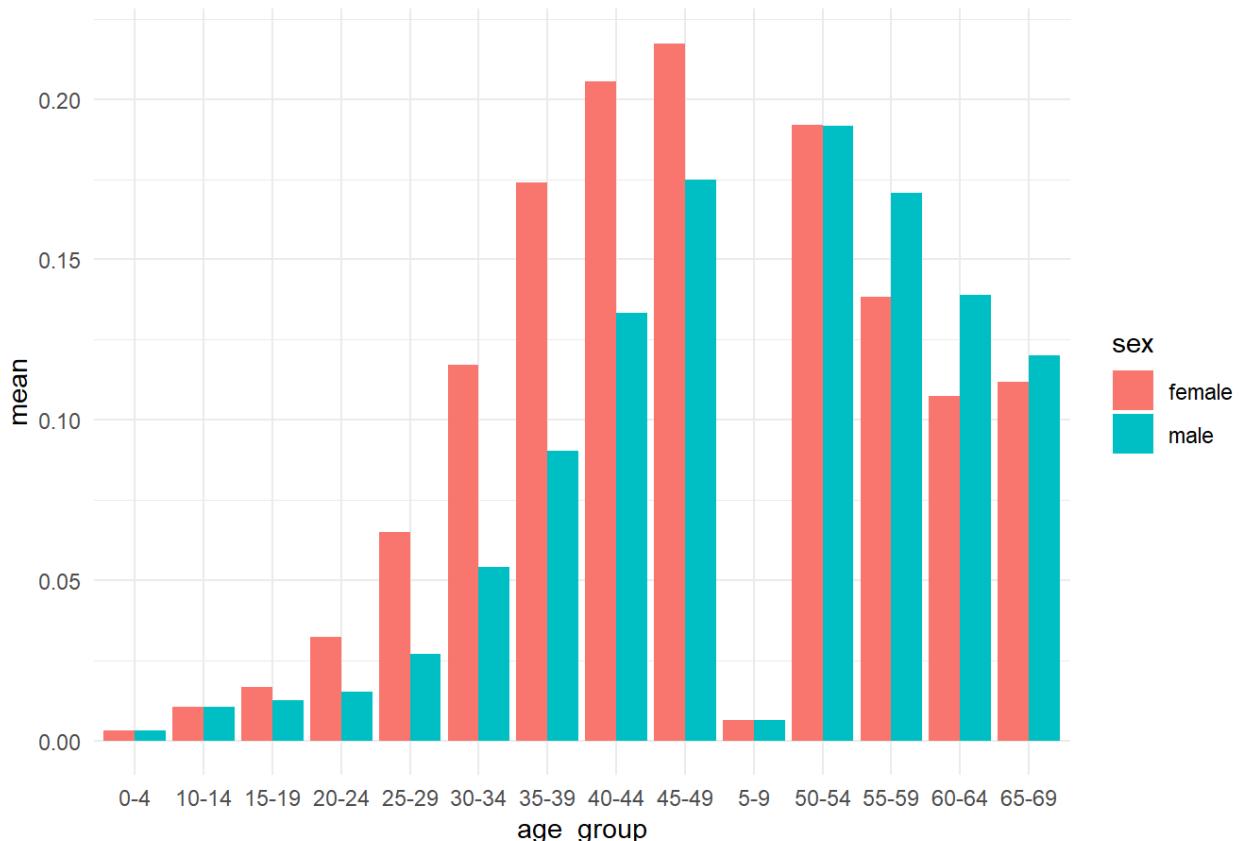
Visualisation de la répartition par âge et sexe

Les disparités d'âge et de sexe en termes de prévalence du VIH sont observées mondialement et sont influencées par de nombreux facteurs qui se chevauchent, notamment la discrimination de genre, les comportements sexuels et l'accès aux soins de santé et à l'éducation. Ces facteurs peuvent influencer à la fois la probabilité d'infection initiale et les issues pour les personnes vivant avec le VIH.

Dans cette leçon, nous nous concentrerons sur la visualisation de la répartition par âge et sexe de divers indicateurs aux niveaux national, régional et de district. Tout d'abord, utilisons `ggplot()` pour créer un graphique en barres national de la prévalence du VIH, regroupé par âge et départagé par sexe.

```
# Graphique en barres de la répartition par âge et sexe
hiv_mwi_agesex %>%
  filter(area_level == "Country",
        indicator == "prevalence") %>%
  ggplot(aes(x = age_group,
             y = mean,
             fill = sex)) +
  geom_col(position = "dodge") +
  theme_minimal() +
  labs(title = "Estimations nationales de la prévalence du VIH, Malawi (déc. 2022)")
```

Estimations nationales de la prévalence du VIH, Malawi (déc. 2022)



Oops! Il semble y avoir un problème avec l'ordre des groupes d'âge. Il est important de corriger cela car nous allons créer des graphiques regroupés par âge et sexe pour le reste de cette leçon.

Réorganisation de l'axe des x

La variable `age_group` est un vecteur de type **caractère**, qui n'est pas intrinsèquement ordonné de la même manière qu'un **facteur**.

```
# Afficher les valeurs uniques de la variable age_group
hiv_mwi_agesex %>% pull(age_group) %>% unique()
```

Error?



```
## [1] "0-4"    "5-9"    "10-14"   "15-19"   "20-24"   "25-29"
## [7] "30-34"  "35-39"  "40-44"  "45-49"  "50-54"  "55-59"
## [13] "60-64"  "65-69"
```

“alphabétiquement”, ce qui signifie que notre groupe d’âge “5-9” est placé dans le mauvais ordre.

Pour ordonner correctement notre graphique en barres selon l’âge, nous pouvons convertir `age_group` en facteur et spécifier l’ordre des niveaux avec `forcats::fct_relevel()`:



```
# Créer un vecteur des valeurs de groupes d'âge ordonnées
ordered_age_groups <- hiv_mwi_agesex %>% pull(age_group) %>%
    unique()

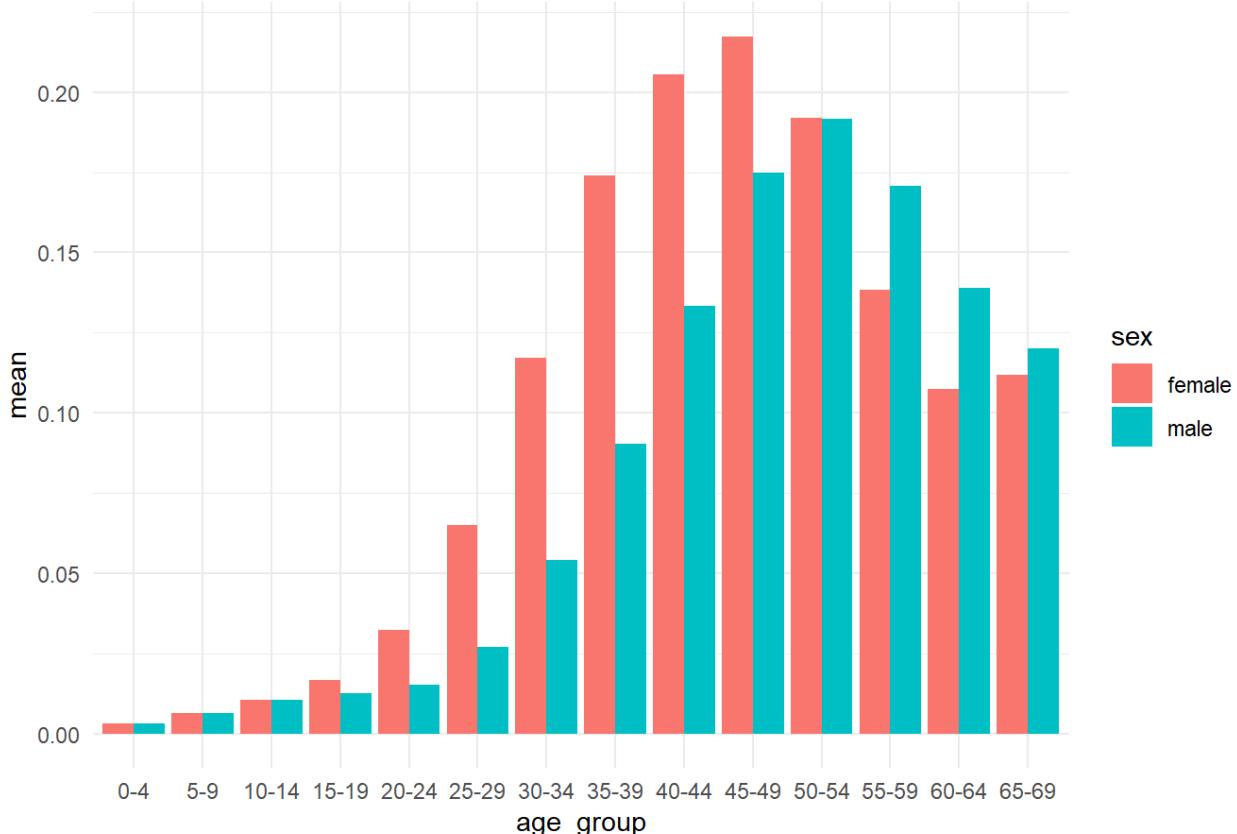
# Réordonner les niveaux de age_group et sauver dans un nouveau
# dataframe
hiv_malawi <- hiv_mwi_agesex %>%
    mutate(age_group = forcats::fct_relevel(age_group,
        ordered_age_groups))
```

Nous sommes maintenant prêts à tracer les distributions d’âge avec notre nouveau dataframe `hiv_malawi`.

Essayons à nouveau le même code `ggplot()` avec `hiv_malawi` :

```
# Graphique avec groupes d'âge réordonnés
hiv_malawi %>%
    filter(area_level == "Country",
        indicator == "prevalence") %>%
    ggplot(aes(x = age_group,
        y = mean,
        fill = sex)) +
    geom_col(position = "dodge") +
    theme_minimal() +
    labs(title = "Estimations nationales de la prévalence du VIH, Malawi (déc.
        2022)")
```

Estimations nationales de la prévalence du VIH, Malawi (déc. 2022)



Bien mieux !

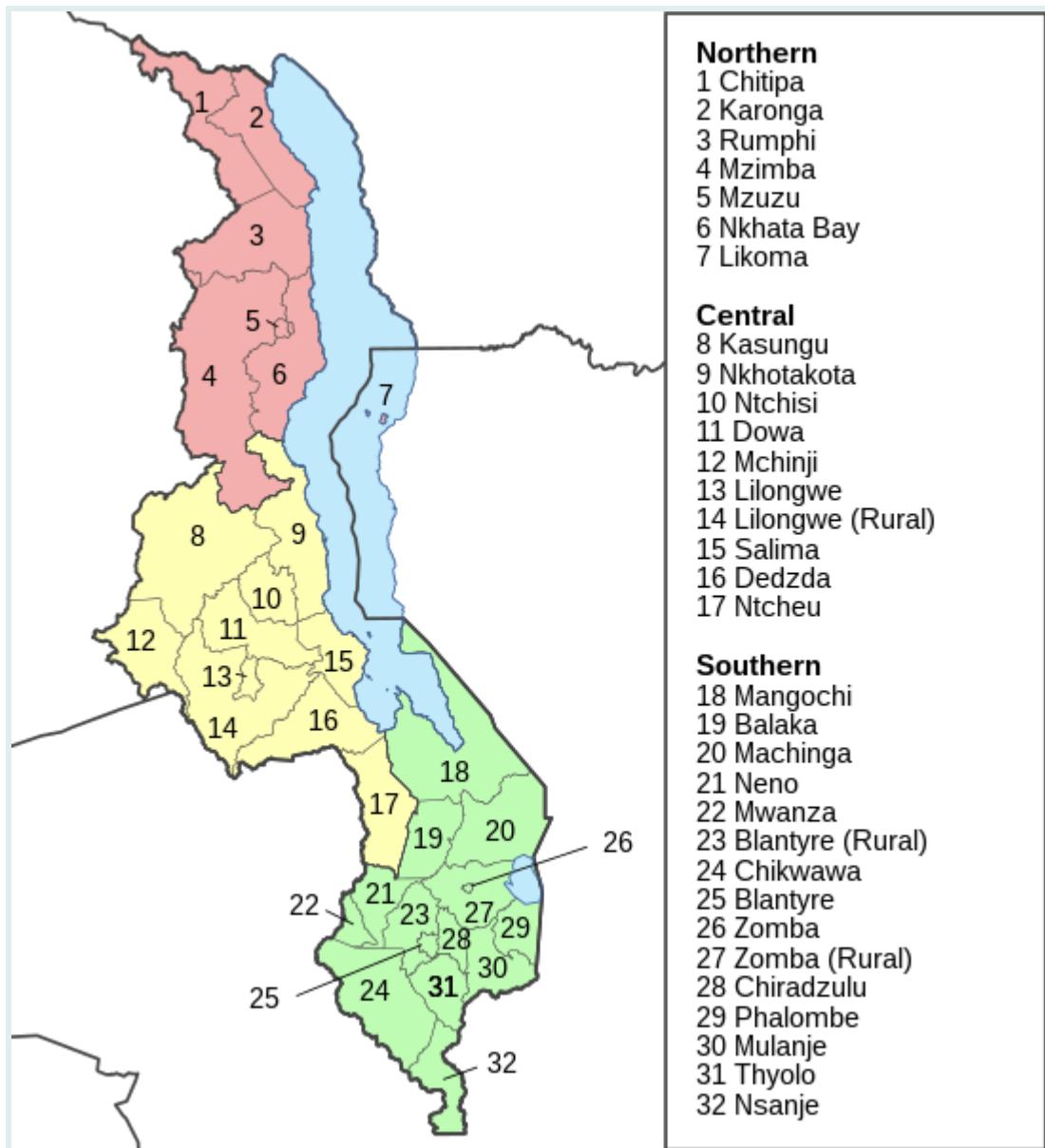
Enseignements sur la prévalence du VIH

SIDE NOTE



Le graphique révèle une disparité de genre discernable dans la prévalence du VIH à partir de 15 ans, probablement liée au début de l'activité sexuelle. Les femmes ont une prévalence nettement plus élevée que les hommes entre 20 et 40 ans, ce qui pourrait refléter des facteurs tels que la vulnérabilité biologique et la dynamique sociale. De manière intéressante, cette tendance s'inverse après 50 ans, où les hommes montrent des taux plus élevés. Ce changement pourrait être influencé par le comportement sexuel des hommes, les taux de mortalité et l'accès ou la recherche de traitement.

Examinons maintenant plus en profondeur si les mêmes tendances sont observées lorsque nous nous concentrons sur des zones plus localisées. Nous pouvons filtrer les données pour tracer la répartition par âge et sexe aux différentes zones géographiques. Notre jeu de données comprend des estimations agrégées pour les 3 principales régions et les 28 districts du Malawi. Nous allons nous intéresser en premier lieu au premier niveau administratif du Malawi — **les trois régions**



Malawi map of regions and districts

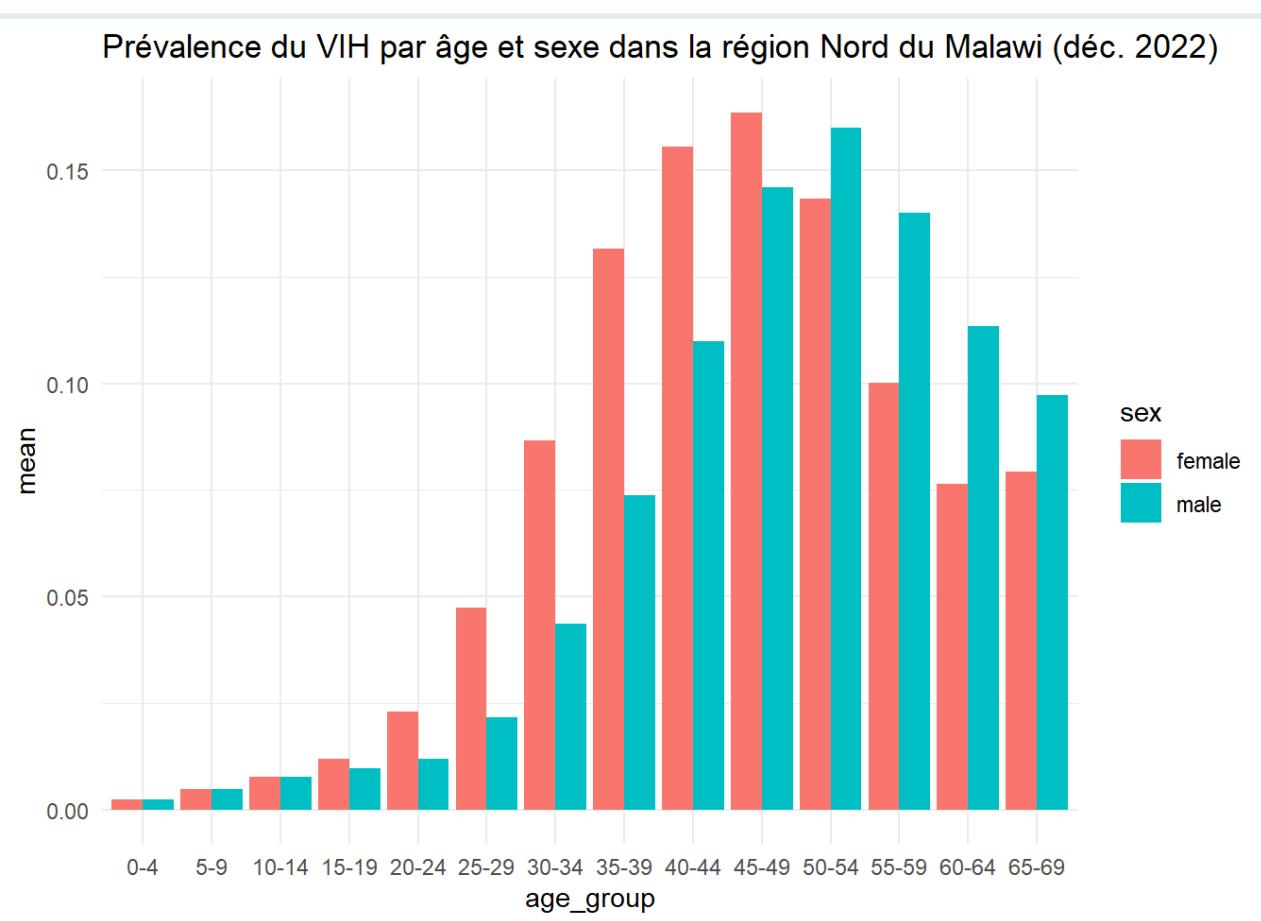
Le défi de la création de graphiques répétitifs

Dans cette section, nous mettrons en évidence un défi commun en matière de visualisation de données : le filtrage répétitif de sous-ensembles de données.

Commençons par créer un graphique de la prévalence du VIH pour la région du "Nord" du Malawi. Nous pouvons copier le code du graphique national que nous avons créé précédemment et remplacer "Country" par "Region", et "Malawi" par le nom de la région que nous souhaitons représenter.

```
# Exemple de filtrage et de création de graphique répétitifs - Région 1

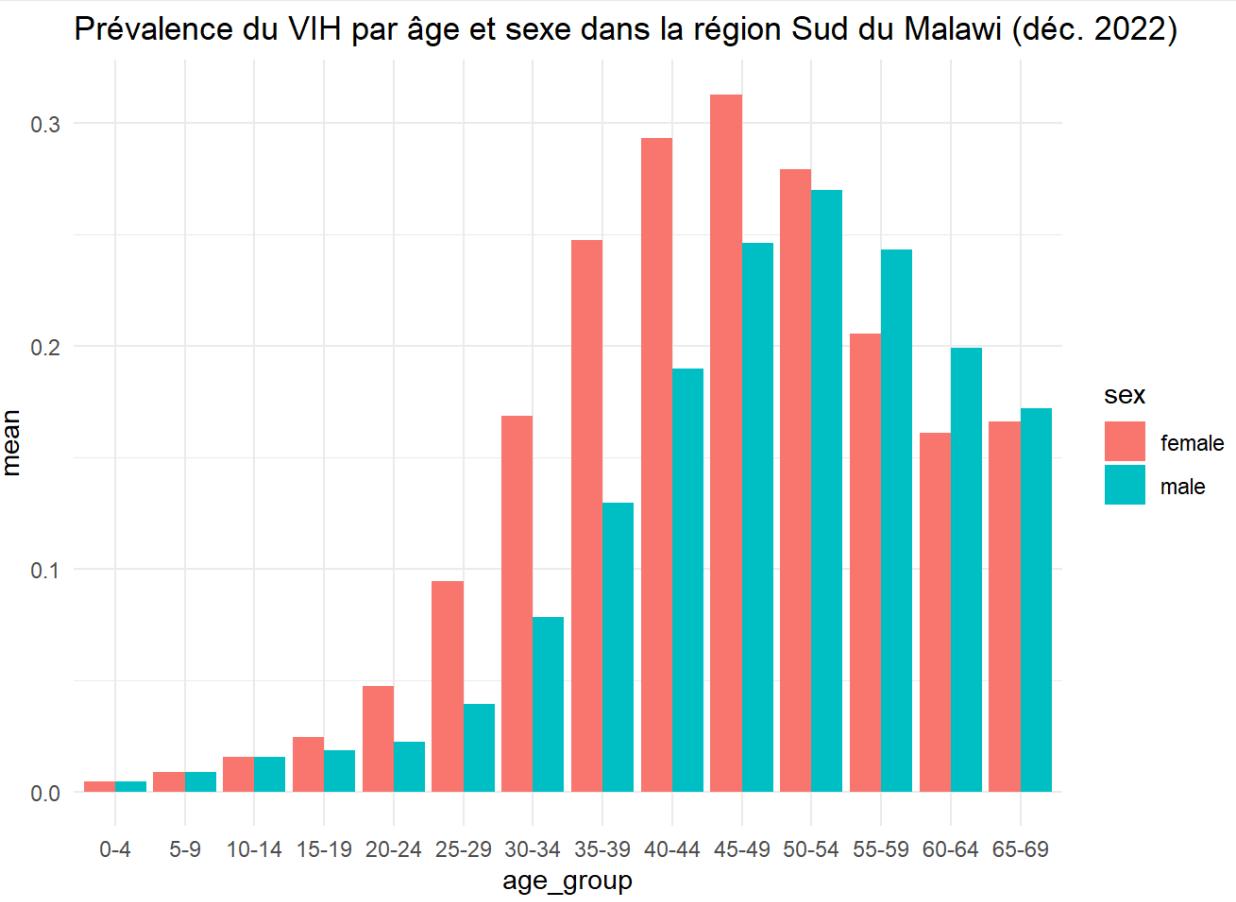
hiv_malawi %>%
  # Filtrer pour la région Nord
  filter(area_level == "Region",
         area_name == 'Northern',
         indicator == "prevalence") %>%
  ggplot(aes(x = age_group,
             y = mean,
             fill = sex)) +
  geom_col(position = "dodge") +
  theme_minimal() +
  # Changer le titre
  labs(title = "Prévalence du VIH par âge et sexe dans la région Nord du Malawi (déc. 2022)")
```



Répétons maintenant cela pour les deux autres régions :

```
# Exemple de filtrage et de création de graphique répétitifs - Région 2

hiv_malawi %>%
  # Filtrer pour la région Sud
  filter(area_level == "Region",
         area_name == 'Southern',
         indicator == "prevalence") %>%
  
  ggplot(aes(x = age_group,
             y = mean,
             fill = sex)) +
  geom_col(position = "dodge") +
  theme_minimal() +
  # Changer le titre
  labs(title = "Prévalence du VIH par âge et sexe dans la région Sud du Malawi
(déc. 2022)")
```

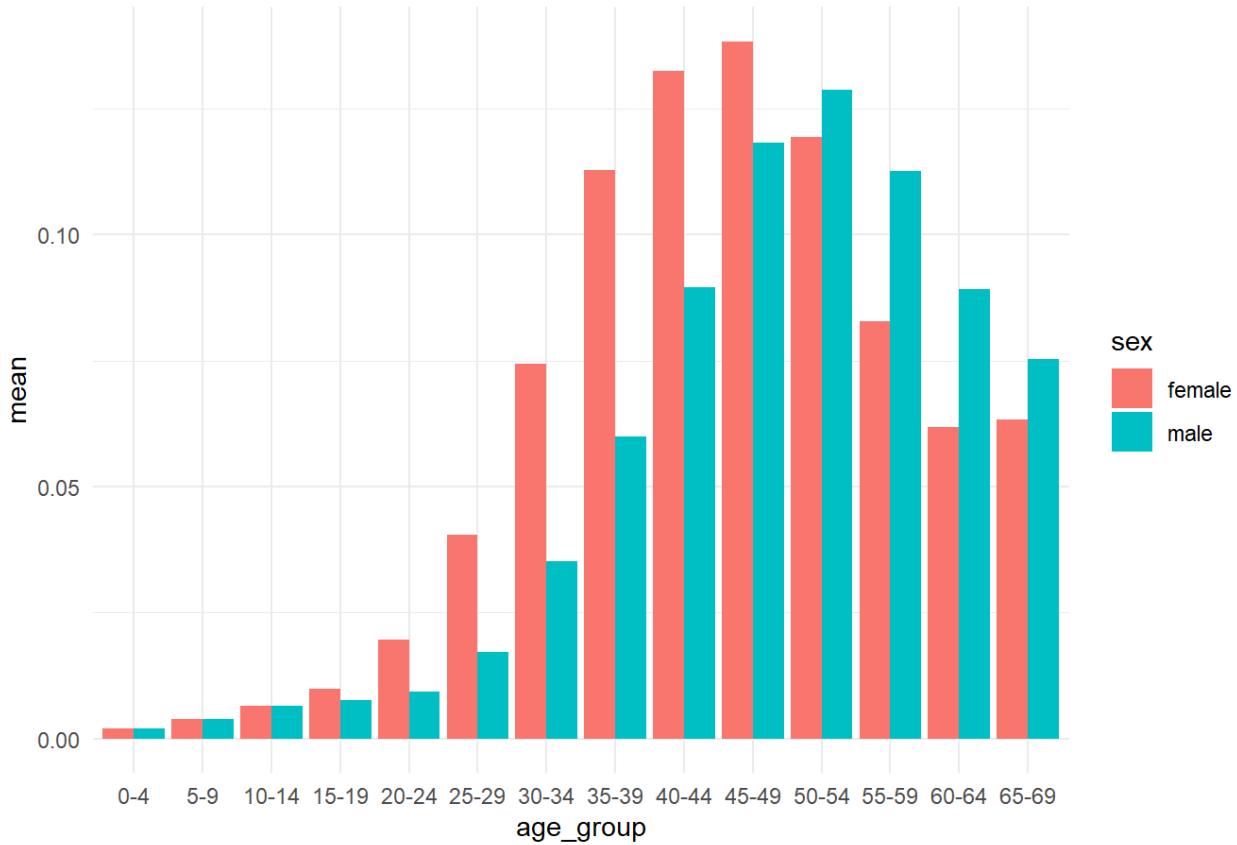


```
# Exemple de filtrage et de création de graphique répétitifs - Région 3

hiv_malawi %>%
  # Filtrer pour la région Centre
  filter(area_level == "Region",
         area_name == 'Central',
         indicator == "prevalence") %>%

  ggplot(aes(x = age_group,
             y = mean,
             fill = sex)) +
  geom_col(position = "dodge") +
  theme_minimal() +
  # Changer le titre
  labs(title = "Prévalence du VIH par âge et sexe dans la région Centre du
        Malawi (déc. 2022)")
```

Prévalence du VIH par âge et sexe dans la région Centre du Malawi (déc. 2022)



Bien que la méthode de copier-coller et de remplacement des noms fonctionne pour un petit nombre de sous-groupes, les limites du filtrage manuel deviennent évidentes lorsque le nombre de sous-groupes augmente. Imaginez faire cela pour chacun des 28 districts dans les données - ce serait extrêmement inefficace et sujet aux erreurs !

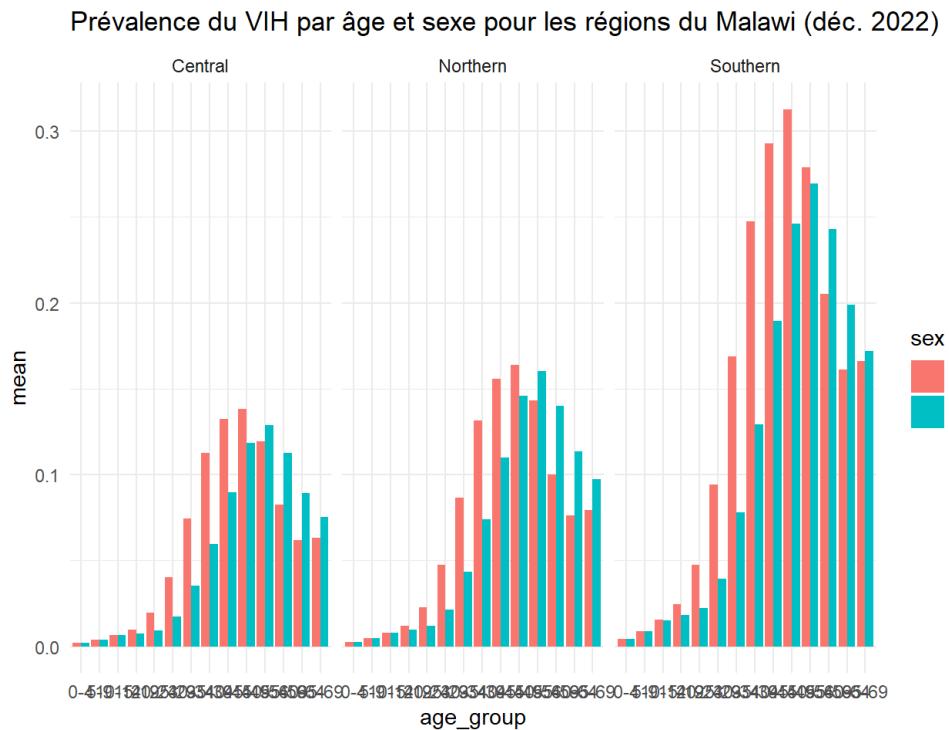
Au lieu de copier-coller le même code et de modifier les variables, nous allons démontrer dans les prochaines sections comment gérer ce défi à l'aide d'une combinaison de `{ggplot2}` et de techniques de programmation fonctionnelle.

Facétages pour petits multiples

Une autre option consiste à créer un graphique facetté, décomposé par région ou district. La limite est qu'il peut y avoir trop de niveaux dans votre variable de regroupement, ou trop de données pour tenir dans chaque sous-ensemble. Un graphique composé de 28 facettes serait encombré et pratiquement illisible.

```
# Exemple de sous-graphiques facettés par région
hiv_malawi %>%
  filter(area_level == "Region",
        indicator == "prevalence") %>%
  ggplot(aes(x = age_group,
             y = mean,
             fill = sex)) +
  geom_col(position = "dodge") +
  theme_minimal() +
  labs(title = "Prévalence du VIH par âge et sexe pour les
        régions du Malawi (déc. 2022)") +
  facet_wrap(~area_name)
```

SIDE NOTE



Parfois, nous pouvons générer des graphiques individuels pour des documents ou des diapositives distincts. Au lieu de nous en tenir uniquement au facétages, nous pouvons créer des fonctions qui nous

SIDE NOTE

permettent de créer une série de graphiques de manière systématique.

PRACTICE**(in RMD)**

Question 1: Filtrage et création de graphiques

Créez un graphique en barres de la répartition par âge et sexe de la **couverture par le TAR** dans le district de **Lilongwe**. Vous pouvez partir du code utilisé pour créer les graphiques régionaux. Cette fois, vous devrez filtrer les données au niveau "**District**" et ajuster le titre du graphique en conséquence.

Maintenant, adaptez votre code pour créer le même graphique pour le district de **Mzimba**.

Création de fonctions de création de graphiques personnalisées

Fonction à un argument

La première étape pour automatiser nos graphiques consiste à créer une petite fonction qui filtrera les données pour nous et créera le sous-ensemble de données représenté.

Par exemple, pour visualiser la prévalence moyenne d'une région, nous pouvons **définir une fonction** prenant comme entrée une condition de sous-ensemble et créant le graphique régional avec les données filtrées.

```
# Simple fonction pour filtrer selon la région et créer le graphique en barres groupées par âge et sexe

plot_region<- function(nom_region) {
  # copier le code au-dessus et remplacer le nom de région par un indicateur

  hiv_malawi %>%
    filter(area_level == "Region",
          area_name == {{nom_region}},
          indicator == "prevalence") %>%
  ggplot(aes(x = age_group,
             y = mean,
             fill = sex)) +
  geom_col(position = "dodge") +
  theme_minimal() +
  labs(title = {{nom_region}})}
```

Le code à l'intérieur de la fonction définie par l'utilisateur ci-dessus est essentiellement le même que celui utilisé pour créer le graphique précédent. La seule différence est que nous ne spécifions pas de nom de région précis, mais créons un “**indicateur**” appelé `{}{nom_region}{{}}` pour contrôler la condition de filtrage et le titre. Dans la fonction `filter()`, nous sous-ensemble nos données selon `nom_region` ; dans la fonction `labs()`, nous utilisons cette chaîne de caractères comme titre du graphique.

Accolades courbes

WATCH OUT



Remarquez l'utilisation d'accolades `{}{}` à l'intérieur de la fonction `plot_region()`. Cette pratique est recommandée lors de l'utilisation de fonctions `{tidyverse}` à l'intérieur d'une autre fonction personnalisée, pour éviter les erreurs. Consultez ici pour plus d'exemples.

Maintenant, exécutons la fonction pour chacune des régions présentées dans le jeu de données, et voyons le résultat !

```
# Créer des graphiques individuels pour les trois régions
nord <- plot_region('Nord')
sud <- plot_region('Sud')
centre <- plot_region('Centre')

# Afficher les graphiques
nord
```

Nord

mean

age_group

sud

Sud

mean

age_group

centre

```
Centre
```

```
mean
```

```
age_group
```

Vous pouvez voir qu'utiliser une fonction personnalisée est bien plus efficace que de répéter et modifier le même bloc de code. Si des modifications sont nécessaires, nous n'avons pas besoin de modifier le code pour chaque graphique individuel. Au lieu de cela, nous pouvons apporter une petite modification à notre fonction `plot_region()`.

Ces graphiques montrent que les tendances de la répartition par âge et sexe de la prévalence du VIH sont les mêmes aux niveaux national et régional. Mais on peut voir que la prévalence globale est beaucoup plus élevée dans la région Sud que dans les autres.

Personnalisation des titres avec `glue()`

Les graphiques générés par notre fonction personnalisée ressemblent *presque* exactement aux précédents - arrivez-vous à repérer une différence ? C'est ça, le titre ! Au lieu de simplement "Centre", nous voulons qu'il dise « Estimations du VIH par âge et sexe dans la région Centre du Malawi ».

Nous pouvons corriger cela avec la fonction `glue()` à l'intérieur de notre fonction personnalisée :

PRO TIP



```
# Adapter la fonction pour inclure un titre personnalisé
plot_region2 <- function(nom_region) {
  hiv_malawi %>%
    filter(area_level == "Region",
          area_name == {{nom_region}},
          indicator == "prevalence") %>%
    ggplot(aes(x = age_group,
               y = mean,
               fill = sex)) +
    geom_col(position = "dodge") +
    theme_minimal() +
    labs(title = glue("Prévalence du VIH par âge et sexe dans la
                      région {nom_region} du Malawi (déc. 2022)"))
}

# Tester la fonction
plot_region2("Centre")
```

PRO TIP



Prévalence du VIH par âge et sexe dans la région Centre du Malawi (déc. 2022)

mean

age_group

PRACTICE



Question 2: Fonction de création de graphique personnalisée pour les districts

- Créer une fonction personnalisée appelée `plot_district()` prenant comme entrée le `nom_district`, et créant un graphique de la répartition par âge et sexe des **personnes vivant avec le VIH** (l'indicateur “`plhiv`”), au niveau du district. Utiliser `glue()` pour créer un titre personnalisé.

Tester la fonction pour créer un graphique pour les districts de **Chitipa et Neno**.

Fonction à entrées multiples

Dans la section précédente, notre fonction `plot_region()` acceptait une seule entrée : `nom_region`, et filtrait les données au niveau “Région” uniquement.

Nous pouvons rendre notre fonction encore plus polyvalente en permettant de personnaliser l’indicateur du VIH à représenter sur l’axe des y, et de filtrer par le niveau “Région” ou “District”.

```
# Créer une fonction personnalisée avec entrées multiples
plot_malawi <- function(nom_zone, niv_zone, indicateur_vih) {
  hiv_malawi %>%
    # filtrer selon 3 conditions
    filter(
      niv_zone == {{niv_zone}},
      nom_zone == {{nom_zone}},
      indicateur == {{indicateur_vih}}) %>%
    ggplot(aes(x = age_group,
               y = mean,
               fill = sex)) +
    geom_col(position = "dodge") +
    theme_minimal() +
    # titre personnalisé
    labs(title = glue("Moyenne de {indicateur_vih} par groupe d'âge dans
                      {nom_zone} {niv_zone}, Malawi (déc. 2022)"))
}
```

Maintenant nous pouvons appliquer la nouvelle fonction personnalisée `plot_malawi()` à n'importe quel indicateur, à n'importe quel niveau géographique de notre jeu de données en spécifiant les 3 entrées requises.

```
# Couverture du TAR pour un district
plot_malawi("Chitipa", "District", "art_coverage")
```

```
## Error in `filter()`:
## i In argument: `indicateur == "art_coverage"` .
## Caused by error:
## ! object 'indicateur' not found
```

```
# Population pour une région
plot_malawi("Sud", "Région", "population")
```

```
## Error in `filter()`:
## i In argument: `indicateur == "population"` .
## Caused by error:
## ! object 'indicateur' not found
```

Filtrage du niveau de zone

SIDE NOTE



La raison pour laquelle nous avons ajouté `niv_zone` est pour éviter une situation où un district et une région partagent le même nom. Ce cas ne se présente pas dans ce jeu de données, mais il n'est pas rare que des États/provinces aient le même nom qu'une ville ou un district important à l'intérieur de leurs frontières (par ex. New York se situe dans l'État de New York). De plus, cela nous permet de

SIDE NOTE

personnaliser le titre de notre graphique pour mentionner le niveau de zone.

En utilisant des fonctions personnalisées, nous pouvons créer des graphiques pour différentes régions, districts et indicateurs sans devoir recopier-coller le code `{ggplot2}` et apporter de multiples ajustements manuellement.

Mais cela reste répétitif ! Il faut toujours recopier et remplacer les noms, même si c'est une seule ligne. Ce n'est toujours pas automatisé !

Par exemple, si nous voulions utiliser notre fonction personnalisée pour créer un graphique des PVVIH pour chacun des 28 districts, il faudrait faire :

```
# Appliquer la fonction personnalisée à chaque district
chitipa_pvvih <- plot_malawi("Chitipa", "District", "pvvih")
karonga_pvvih <- plot_malawi("Karonga", "District", "pvvih")
# etc
```

Quelle corvée ! Heureusement, R nous fournit un moyen d'**itérer** notre fonction personnalisée à travers toutes les régions ou districts, sans aucune recopie-coller.

Bouclage à travers un vecteur de variables

Présentation de `purrr::map()`

Nous pouvons créer un vecteur de noms et faire fonctionner la fonction sur tous les noms de ce vecteur à l'aide de la fonction `map()` du package `{purrr}`.

`map()` prendra deux arguments : un **vecteur** et une **fonction**.

`map()` appliquera alors la fonction à chaque élément du vecteur d'entrée.

Appliquer la fonction à chaque élément du vecteur d'entrée résulte en un élément de sortie par élément d'entrée.

`map()` combine alors tous ces éléments de sortie dans une liste.

Par exemple, voici une fonction personnalisée qui prend en entrée un nom et ajoute le préfixe "Dr." au début de la chaîne de caractères.

Appliquer la fonction à un nom unique

```
# Exemple de fonction à un seul argument
add_dr <- function(nom_complet) {
  return(paste("Dr.", nom_complet))
}
# Appliquer la fonction à un nom unique
add_dr("Mohamed Hsin Bennour")
```

```
## [1] "Dr. Mohamed Hsin Bennour"
```

Imaginons maintenant que nous ayons un vecteur de noms pour lesquels nous souhaitons ajouter le préfixe “Dr.”.

```
# Liste de personnes
etudiants_phd <- c("Mohamed Hsin Bennour", "Imad El Badisy", "Kenechukwu David Nwosu")
```

Nous passons le vecteur de noms à `purrr::map()`, en insérant notre fonction personnalisée `add_dr()` comme argument. Cela nous permettra d’appliquer la fonction personnalisée à tous les éléments du vecteur, en itérant le processus.

```
# Boucler la fonction sur le vecteur de variables
purrr::map(etudiants_phd, add_dr)
```

```
## [[1]]
## [1] "Dr. Mohamed Hsin Bennour"
##
## [[2]]
## [1] "Dr. Imad El Badisy"
##
## [[3]]
## [1] "Dr. Kenechukwu David Nwosu"
```

Vous remarquerez que la sortie de `purrr::map()` est une liste. Pour récupérer les éléments de la liste, nous pouvons d’abord l’assigner à un objet et ensuite utiliser l’opérateur `[[]]`:

```
# Transmettre le vecteur à map() et sauvegarder la sortie dans une liste
etudiants_diplomes <- etudiants_phd %>% purrr::map(add_dr)

# Afficher la liste
etudiants_diplomes
```

```
## [[1]]
## [1] "Dr. Mohamed Hsin Bennour"
##
## [[2]]
## [1] "Dr. Imad El Badisy"
##
```

```
## [3]
## [1] "Dr. Kenechukwu David Nwosu"
```

```
# Rappeler un élément spécifique de la liste
etudiants_diplomes[[2]]
```

```
## [1] "Dr. Imad El Badisy"
```

En essence, `map()` effectue le même travail qu'une boucle `for`, mais de manière fonctionnelle.

Automatisation des graphiques ggplot2

Nous pouvons utiliser le même workflow pour créer une liste de graphiques, en appliquant notre fonction personnalisée `plot_malawi()` à un vecteur de noms de régions.

```
# Créer un vecteur des 3 régions du Malawi
noms_regions <- c("Nord", "Centre", "Sud")

# Appliquer plot_region() à noms_regions
noms_regions %>% map(plot_malawi, "Région", "prevalence")
```

```
## Error in `map()`:
## i In index: 1.
## Caused by error in `filter()`:
## i In argument: `indicateur == "prevalence"` .
## Caused by error:
## ! object 'indicateur' not found
```

Nous avons maintenant créé 3 graphiques avec seulement 2 lignes de code.



Question 3: Itération à travers un vecteur de districts

Créez un vecteur de 5 noms de districts à partir de `hiv_malawi`.

Appliquez la fonction `plot_malawi()` au vecteur de noms de districts pour créer cinq graphiques des PVVIH d'un coup.

Fonction d'aide pour le niveau de zone

L'analyse des tendances épidémiologiques à différentes échelles géographiques (pays, état, département, ville, etc.) est cruciale. Nous pouvons vouloir créer un

graphique pour chaque état, ou chaque département.

Nous pouvons créer un vecteur de tous les noms de districts à partir de `hiv_malawi` avec ce modèle de code :

```
# Création d'un vecteur des noms de district uniques
noms_district <- hiv_malawi %>%
  filter(niv_zone == "District") %>%
  pull(nom_zone) %>%
  unique()
```

```
## Error in `filter()`:
## i In argument: `niv_zone == "District"` .
## Caused by error:
## ! object 'niv_zone' not found
```

```
# Afficher
noms_district
```

```
## Error in eval(expr, envir, enclos): object 'noms_district' not found
```

Ce code identifie les noms de zone uniques au niveau “District”. Cependant, répéter manuellement cela pour différents niveaux est inefficace. Pour optimiser, nous introduisons une fonction d'aide appelée `area_lvl()`:

```
# Écrire une fonction d'aide pour obtenir les noms de zone uniques pour un
# niveau donné
area_lvl <- function(niveau) {
  hiv_malawi %>%
    filter(niv_zone == {{niveau}}) %>%
    pull(nom_zone) %>%
    unique() %>%
    return()
}

# Tester la fonction d'aide
area_lvl("Région")
```

```
## Error in `filter()`:
## i In argument: `niv_zone == "Région"` .
## Caused by error:
## ! object 'niv_zone' not found
```

```
area_lvl("District")
```

```
## Error in `filter()`:
## i In argument: `niv_zone == "District"` .
```

```
## Caused by error:  
## ! object 'niv_zone' not found
```

Cette fonction simplifie l'obtention des noms uniques pour n'importe quel niveau de zone. Nous pouvons utiliser `area_lvl()` avec `map()` pour itérer à travers toutes les régions ou tous les districts, sans avoir à créer un vecteur personnalisé au préalable.

```
# Tracer l'incidence pour toutes les régions  
area_lvl("Région") %>% map(plot_malawi, "Région", "incidence")
```

```
## Error in `filter()`:  
## i In argument: `niv_zone == "Région"`.  
## Caused by error:  
## ! object 'niv_zone' not found
```

```
# Tracer la sensibilisation au VIH pour tous les districts  
area_lvl("District") %>% map(plot_malawi, "District",  
  "prop_pvvih_connnaissance_statut")
```

Bouclage à travers deux vecteurs

Pour seulement quelques variables de réponse, nous pourrions facilement copier-coller le code ci-dessus en changeant à chaque fois la variable codée en dur sur l'axe des y (`indicateur`). Ce processus peut devenir lourd si nous souhaitons le faire pour de nombreux indicateurs.

Bien que nous puissions utiliser une boucle imbriquée, avec une seconde fonction `map()` imbriquée dans la précédente, cette méthode n'est pas aussi facile à interpréter qu'une boucle `for`.

Ici, nous allons créer un vecteur de deux indicateurs, et les transmettre à une boucle `for`. La boucle `for` transmettra chaque indicateur à `map()`, et nous obtiendrons 6 graphiques.

```
# Choisir les indicateurs: PVVIH et Prévalence  
indicateurs <- c("pvvih", "prevalence")  
  
# Boucle imbriquée pour représenter 3 régions x 2 indicateurs  
for (i in 1:length(indicateurs)) {  
  area_lvl("Région") %>%  
    map(plot_malawi, "Région", indicateurs[i]) %>%  
    print()  
}  
  
## Error in `filter()`:  
## i In argument: `niv_zone == "Région"`.
```

```
## Caused by error:  
## ! object 'niv_zone' not found
```

Nous pouvons changer “Région” en “District”, et le code ci-dessus nous donnerait 56 graphiques, 2 indicateurs pour chacun des 28 districts.

Finalisation et enregistrement

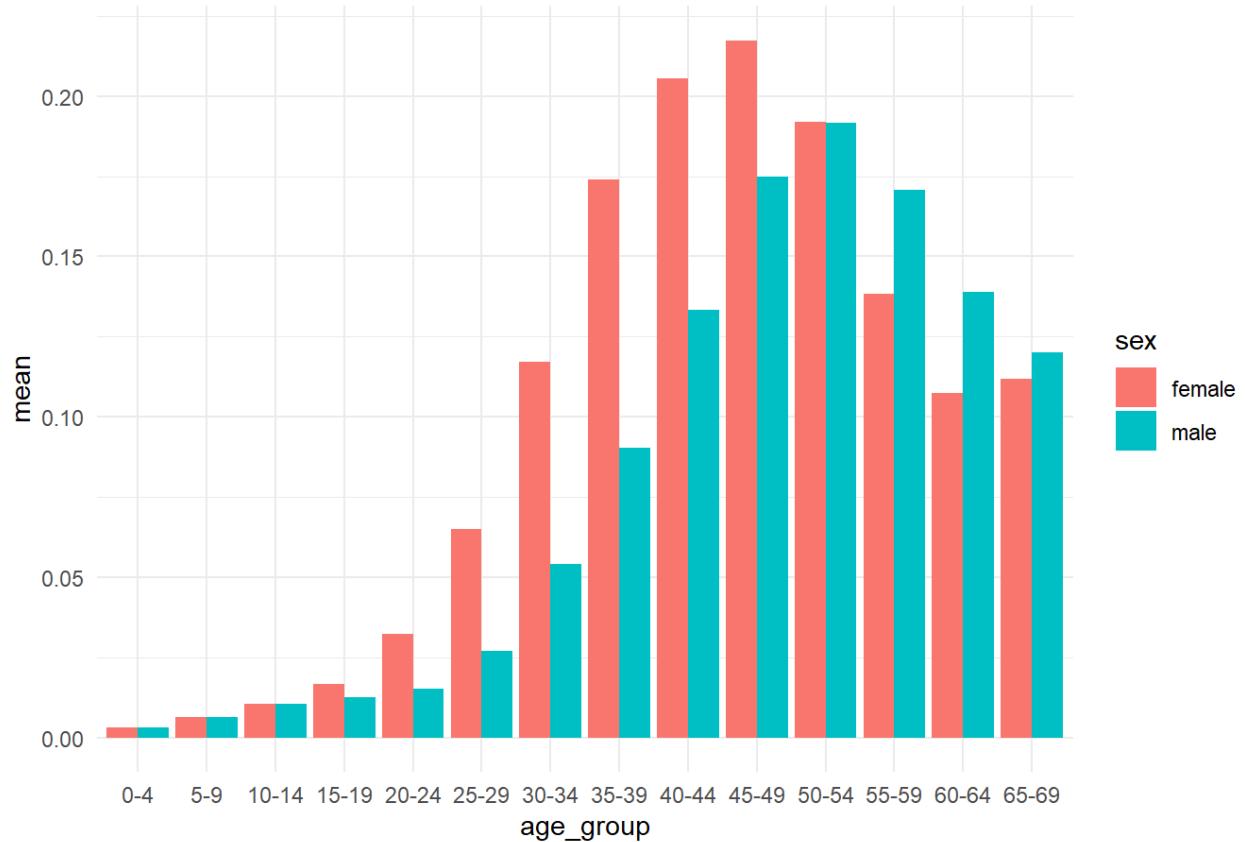
Maintenant que notre processus de création de graphiques est entièrement automatisé, nous pouvons nous préparer à les finaliser et à enregistrer les images pour une utilisation ultérieure.

Examinons d’abord les graphiques et décidons si des réglages sont nécessaires. Si des modifications s’imposent, nous n’avons pas besoin de modifier le code pour chaque graphique individuel. Il suffit d’apporter un petit réglage à notre fonction `plot_malawi()` et de la relancer avec la fonction `map()`. C’est un moyen puissant de gérer de multiples graphiques de façon efficace.

Revisitons notre code `ggplot()` d’origine, que nous avons filtré et réutilisé pour tous les graphiques :

```
# Graphique en barres des prévalences nationales par âge et sexe  
hiv_malawi %>%  
  filter(area_level == "Country",  
         indicator == "prevalence") %>%  
  ggplot(aes(x = age_group,  
             y = mean,  
             fill = sex)) +  
  geom_col(position = "dodge") +  
  theme_minimal() +  
  labs(title = "Estimations nationales de la prévalence du VIH, Malawi (déc.  
2022)")
```

Estimations nationales de la prévalence du VIH, Malawi (déc. 2022)



Ce graphique n'est pas mauvais en soi, mais de nombreuses caractéristiques peuvent encore être ajustées et affinées pour obtenir un graphique prêt pour publication. Rappelez-vous que les graphiques `ggplot2` sont infiniment personnalisables !

Le code ci-dessous apporte plusieurs modifications pour créer un graphique final plus affiné.

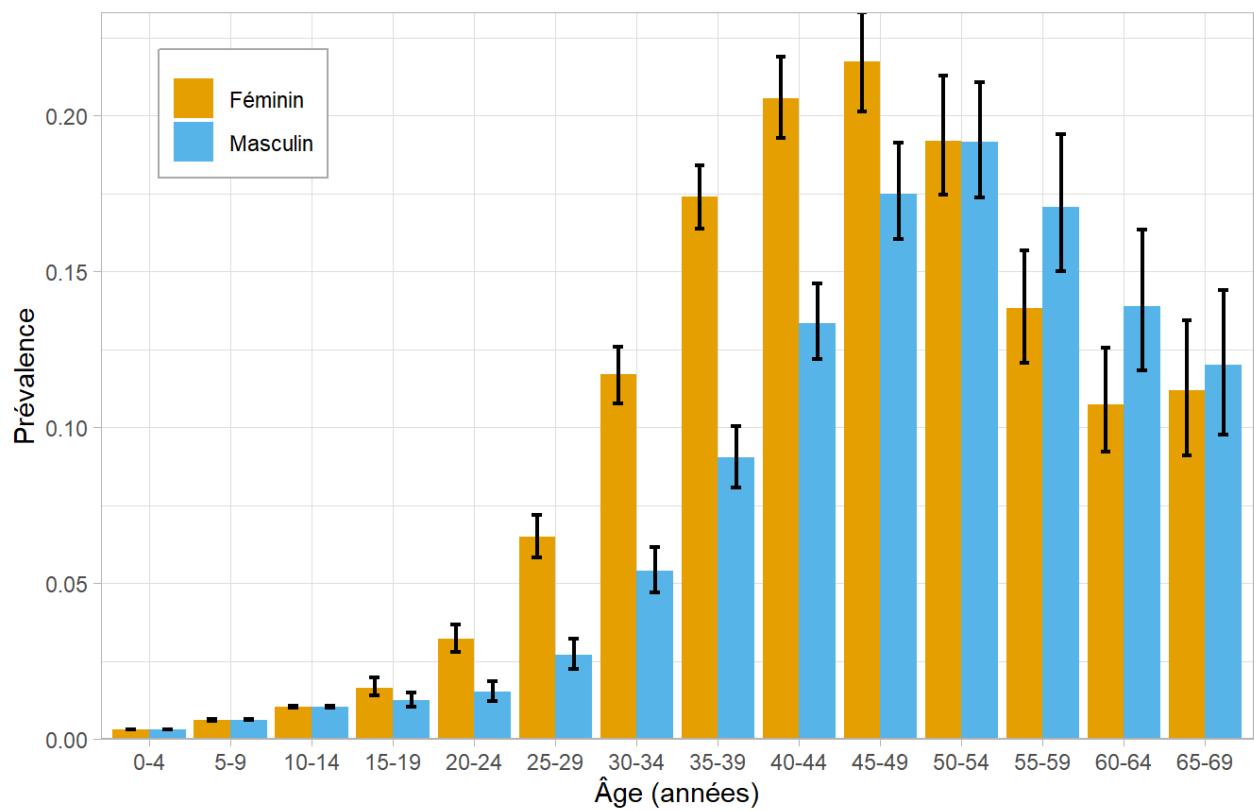
```

# Graphique national des prévalences avec modifications supplémentaires
ggplot(data = filter(hiv_malawi,
                     indicator == "prevalence",
                     area_name=="Malawi"),
        aes(x = age_group, y = mean, fill = sex,
            ymin = lower, ymax = upper)) +
  geom_col(position = position_dodge(width = 0.9)) +
  geom_errorbar(aes(ymin = lower, ymax = upper),
                position = position_dodge(width = 0.9),
                width = 0.25, linewidth = 0.8) +
  scale_y_continuous(expand = c(0, 0)) +
  scale_fill_manual(values = c("female" = "#E69F00", "male" = "#56B4E9"),
                    labels = c("Féminin", "Masculin")) +
  theme_light() +
  theme(legend.position = c(0.05, 0.95),
        legend.justification = c(0, 1),
        legend.box.just = "left",
        legend.box.background = element_rect(color = "white", linewidth = 0.5),
        legend.background = element_rect(color = "darkgray", linewidth = 0.5),
        legend.title = element_blank()) +
  labs(x = "Âge (années)",
       y = "Prévalence",
       title = "Répartition par âge et sexe de la prévalence du VIH au Malawi",
       subtitle = "Estimations du modèle Naomi, décembre 2022")

```

Répartition par âge et sexe de la prévalence du VIH au Malawi

Estimations du modèle Naomi, décembre 2022



Une fois satisfaits du rendu du graphique, nous pouvons utiliser ce nouveau code `ggplot()` dans une fonction personnalisée :

```

# Créer une fonction personnalisée avec entrées multiples
plot_malawi_final <- function(nom_zone, niv_zone, indicateur_vih) {
  filter(hiv_malawi,
    area_name == {{nom_zone}},
    area_level == {{niv_zone}},
    indicator == {{indicateur_vih}}) %>%
  
  # FILTRER SELON LES ENTREES
  # AJOUTER DES INDICATEURS POUR LE FILTRAGE

  ggplot(aes(x = age_group, y = mean, fill = sex)) +
  geom_col(position = position_dodge(width = 0.9)) +
  geom_errorbar(aes(ymax = upper, ymin = lower),
    position = position_dodge(width = 0.9),
    width = 0.25, linewidth = 0.8) +
  scale_y_continuous(expand = c(0, 0)) +
  scale_fill_manual(values = c("female" = "#E69F00", "male" = "#56B4E9"),
    labels = c("Féminin", "Masculin")) +
  theme_light() +
  theme(legend.position = c(0.05, 0.95),
    legend.justification = c(0, 1),
    legend.box.just = "left",
    legend.box.background = element_rect(color = "white", linewidth = 0.5),
    legend.background = element_rect(color = "darkgray", linewidth = 0.5),
    legend.title = element_blank()) +
  
  # AJOUTER DES INDICATEURS POUR LE TITRE ET L'AXE Y

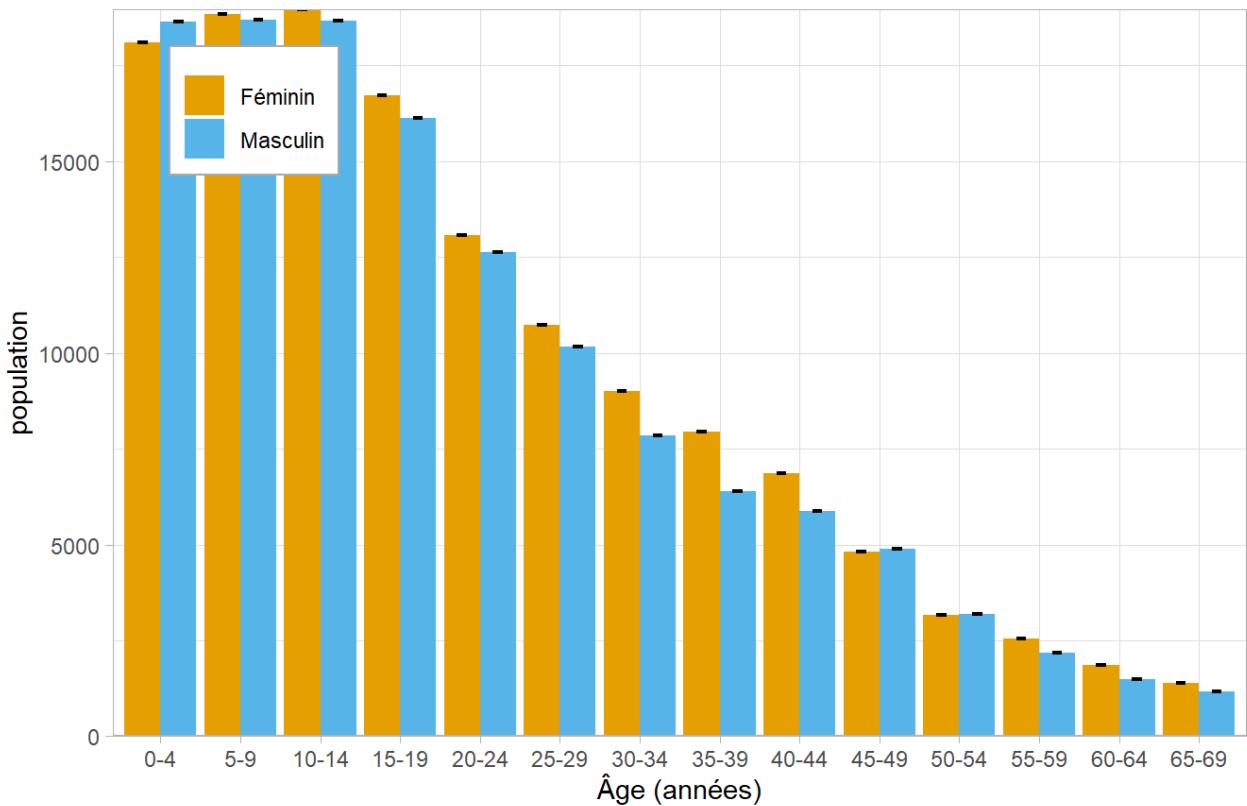
  labs(x = "Âge (années)",
    y = indicateur_vih,
    title = glue("Répartition par âge et sexe de {indicateur_vih} dans
      {nom_zone} {niv_zone}"),
    subtitle = "Estimations du modèle Naomi, Malawi, décembre 2022")
}

# Tester la fonction
plot_malawi_final("Chitipa", "District", "population")

```

Répartition par âge et sexe de population dans Chitipa District

Estimations du modèle Naomi, Malawi, décembre 2022



Nous pouvons utiliser cette nouvelle fonction pour obtenir des graphiques pour un ensemble de régions ou de districts, tout comme nous l'avons fait avec la fonction `plot_malawi()` précédente.

```
# Itérer sur les régions
area_lvl("Région") %>% map(plot_malawi_final, "Région", "prevalence")
```

```
## Error in `filter()`:
## i In argument: `niv_zone == "Région"` .
## Caused by error:
## ! object 'niv_zone' not found
```

Maintenant que nous savons que la fonction fonctionne correctement et génère les graphiques souhaités, il est temps de les sauver localement. Cela nous permettra d'accéder aux graphiques sans avoir à rerunning le code à chaque fois pour les générer.

Nous allons faire cela en apportant un changement final à notre fonction de création de graphiques. Cette fois, nous ajouterons la fonction `ggsave()` à la fin, pour sauver nos graphiques sous forme de fichiers image avec des noms uniques et descriptifs.

```

# Créer une fonction personnalisée pour tracer ET enregistrer dans un dossier
# spécifique
plot_save_final <- function(nom_zone, niv_zone, indicateur_vih){

  hiv_malawi %>%
    ggplot(...) +
    geom_col(...) +
    # Code plot
    labs(...)

  # NOUVEAU CODE COMMENCE ICI: Sauver le graphique avec des noms de fichier
  # personnalisés
  ggsave(filename =
    here(glue("outputs/{indicateur_vih}_{niv_zone}_{nom_zone}.jpg")))
}

}

```

Maintenant que nous avons finalisé notre fonction personnalisée appelée `plot_save_final()`, essayons-la pour le district de Chitipa !

```
plot_save_final("Chitipa", "District", "pvvih")
```

Vous devriez maintenant voir un nouveau fichier nommé “pvvih_District_Chitipa.jpg” dans votre dossier outputs.

Comme précédemment, créons une boucle for pour nos deux indicateurs `pvvih` et `prevalence`, mais cette fois nous utiliserons notre fonction `plot_save_final()` pour créer les graphiques et sauver les images dans notre dossier `outputs` !

```

# Itérer sur les régions et SAUVER
area_lvl("Région") %>% map(plot_save_final, "Région", "prevalence")

```

Ensuite, bouclons sur **deux** vecteurs et sauvons les graphiques au niveau régional pour deux autres indicateurs du VIH.

```

# Choisir de nouveaux indicateurs
indicateurs2 <- c("pvvih", "couverture_tar")

# Boucler à travers la fonction de sauvegarde de graphiques
for (i in 1:length(indicateurs2)) {
  area_lvl("Région") %>%
    map(plot_save_final, "Région", indicateurs2[i])
}

```

Si vous accédez au dossier `outputs`, vous devriez maintenant trouver **6** nouveaux graphiques créés et sauvegardés. C'est la magie de l'automatisation sous R !



Question 4: Sauvegarder une série de graphiques

En utilisant votre vecteur `districts5` de la dernière question, écrire une boucle `for` pour créer et sauver les graphiques de répartition par âge et sexe pour:

- La prévalence
- La couverture du TAR
- Les PVVIH

EN RÉSUMÉ !

Dans cette leçon, nous avons appris à développer des fonctions personnalisées de filtrage et de tracé avec `{dplyr}` et `{ggplot2}`, et à les itérer à travers des vecteurs en deux dimensions avec `purrr::map()` et les boucles `for`.

De cette façon, nous pouvons générer des graphiques personnalisés de manière efficace et les sauvegarder pour une utilisation ultérieure, sans avoir à les créer individuellement à chaque fois. Cette approche offre une démonstration puissante de la façon dont les principes de programmation fonctionnelle peuvent être utilisés pour écrire un code plus propre, plus modulaire et plus facile à maintenir.

References

Some material in this lesson was adapted from the following sources:

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.



Création de rapports paramétrés avec {R Markdown}

Introduction
Objectifs d'apprentissage
Packages
Jeu de données
Construire un seul rapport pour un pays (par exemple, "Angola")
Sortie 1 : Graphique en ligne pour "Angola"
Sortie 2 : Tableau Statistique pour "Angola"
Principe DRY : Ne vous répétez pas !
Approche 1 : Produire une sortie grâce à une fonction personnalisée
Construction d'un Rapport Paramétrisé
Configuration de l'En-tête YAML
Exemple avec un Jeu de Données sur le VIH
Utilisation du Paramètre dans le Document
Mise en Œuvre des Paramètres dans l'Analyse
Explication de la Cartographie des Paramètres
Tricotage avec Différents Paramètres
Tricotage avec Différents Paramètres
Le Processus Complet
Étape 0 : Créer un Projet R
Étape 1 : En-tête YAML
Étape 2 : Configuration de l'Environnement du Document
Étape 3 : Le Cœur de Votre Rapport
Étape 4 : Tricoter le Rapport
Étape 5 : Créer un Script R pour Paramétriser Tout le Processus
CONCLUSION !

Introduction

- **Importance de la paramétrisation des rapports :**
 - Représente une tâche cruciale dans le rapportage épidémiologique.
 - Offre une flexibilité pour générer dynamiquement du contenu basé sur des paramètres spécifiques.
- **Polyvalence et application inestimable :**
 - Inestimable dans des scénarios nécessitant la génération de rapports en fonction de divers facteurs (pays, périodes, incidences de maladies).
 - Permet l'utilisation d'un modèle unique et polyvalent pour produire une multitude de rapports.
- **Consistance, précision et efficacité :**
 - Assure la cohérence et la précision dans le rapportage des données épidémiologiques.
 - Améliore considérablement l'efficacité de la communication des données épidémiologiques.
- **Amélioration de la communication des données :**
 - Permet de présenter des données complexes dans un format plus accessible et interprétable.
 - Aide à la diffusion efficace d'informations sanitaires critiques.

Objectifs d'apprentissage

- 1. Comprendre l'importance de la paramétrisation dans R Markdown :** Comprendre le concept fondamental de la paramétrisation, son importance dans le rapportage épidémiologique, et comment elle transforme la manière dont les données sont présentées et analysées.
- 2. Apprendre à créer des rapports dynamiques basés sur des paramètres spécifiés :** Développer la compétence pour rédiger des rapports dans R Markdown qui ajustent automatiquement leur contenu en fonction des paramètres donnés, tels que les régions géographiques ou les cadres temporels.
- 3. Développer des compétences dans la création de fonctions pour la paramétrisation de rapports :** Acquérir une maîtrise dans l'écriture de fonctions en R qui permettent la paramétrisation de rapports, simplifiant ainsi le processus de génération de rapports.
- 4. Explorer la programmation fonctionnelle avec `map()` et `pwalk()` :** Se plonger dans les aspects de la programmation fonctionnelle de R, en se concentrant spécifiquement sur l'utilisation des fonctions `map()` et `pwalk()` du package `{purrr}`. Comprendre comment ces fonctions peuvent être utilisées pour gérer efficacement de multiples ensembles d'entrées de données pour la génération de rapports.

En atteignant ces objectifs, vous serez bien équipé pour gérer des ensembles de données épidémiologiques et les présenter de manière informative, organisée et percutante. La capacité à paramétriser des rapports simplifie non seulement le processus d'analyse des données, mais augmente également considérablement la polyvalence et l'applicabilité des résultats présentés.

Packages

Ce fragment de code ci-dessous montre comment charger les paquets nécessaires en utilisant `p_load()` du package `{pacman}`. S'il n'est pas installé, il installe le paquet avant de le charger.

```
pacman::p_load(readr, ggplot2, dplyr, knitr, purrr, gt, kableExtra)
```

```
## package 'kableExtra' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
```

```
## C:\Users\Sabina  
Rodriguez\AppData\Local\Temp\RtmpE54HPD\downloaded_packages
```

Jeu de données

```
hiv_data <- read_csv(here::here("data/clean/hiv_incidence.csv"))
```

Le jeu de données `hiv_incidence.csv` contient trois colonnes :

1. `country` : Le nom du pays.
2. `year` : L'année de l'enregistrement.
3. `new_cases` : Le nombre de nouveaux cas de VIH signalés dans ce pays pour l'année donnée.

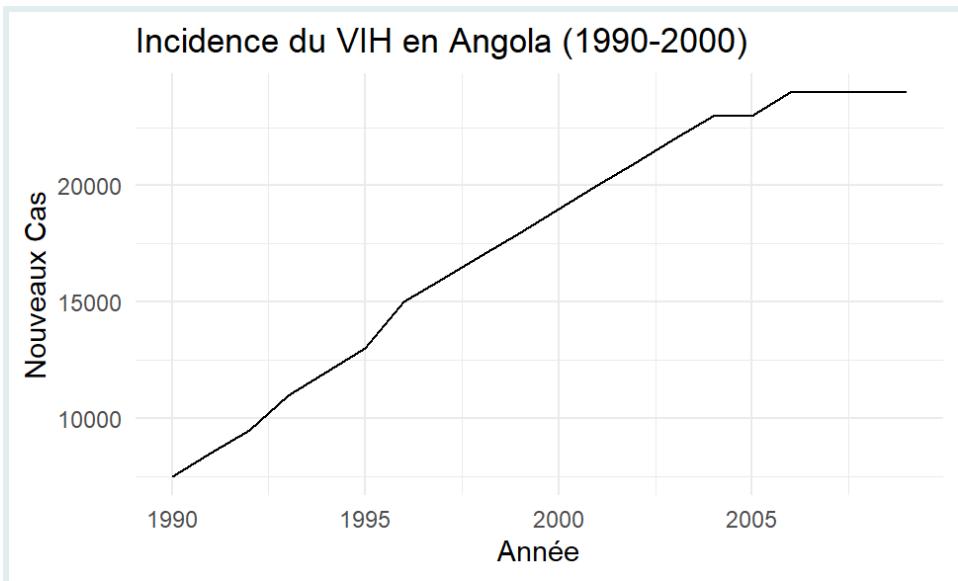
Avec cette compréhension, procédons à la création des rapports comme décrit précédemment. Nous adapterons les étapes à ce jeu de données spécifique. Le tutoriel démontrera comment construire des rapports se concentrant sur les données de prévalence du VIH pour différents pays au cours de diverses années.

Construire un seul rapport pour un pays (par exemple, “Angola”)

- 1. Concentrez-vous sur la création de rapports visuels :**
 - Créer un rapport visuel pour l’Angola en utilisant le langage R.
- 2. Analyser et visualiser les tendances :**
 - L’objectif est d’analyser et de visualiser les nouveaux cas de VIH en Angola.
- 3. Analyse temporelle sur plusieurs années :**
 - Se concentrer sur la tendance des nouveaux cas de VIH sur une série d’années.

Sortie 1 : Graphique en ligne pour “Angola”

```
angola_data <- subset(hiv_data, country == "Angola")  
ggplot(angola_data, aes(x = year, y = new_cases)) +  
  geom_line() +  
  theme_minimal() +  
  labs(title = "Incidence du VIH en Angola (1990-2000)",  
       x = "Année",  
       y = "Nouveaux Cas")
```



Sortie 2 : Tableau Statistique pour “Angola”

Ce fragment de code ci-dessous est conçu pour générer un tableau récapitulatif pour l’Angola, en se concentrant sur les cas de VIH par année. Le tableau mettra en évidence les données de l’année la plus récente pour une identification plus facile. Décomposons et expliquons chaque partie du code.

```
# Filtrer pour les données de l'Angola
angola_data <- hiv_data %>% filter(country == "Angola")

# Résumer les données par année
angola_summary <- angola_data %>%
  group_by(year) %>%
  summarise(Total_Cases = sum(new_cases))

# Identifier l'année la plus récente
most_recent_year <- max(angola_summary$year)

# Afficher le tableau en utilisant kable et mettre en évidence l'année la plus récente
angola_summary %>%
  kable("html", caption = "Résumé des cas de VIH en Angola par année") %>%
  kable_styling(bootstrap_options = c("striped", "hover")) %>%
  row_spec(which(angola_summary$year == most_recent_year), background =
    "lightblue")

## Error in row_spec(., which(angola_summary$year == most_recent_year), background = "lightblue"): could not find function "row_spec"
```

```
angola_summary %>%
  gt() %>%
  tab_style(
    style = cell_fill(color = "lightblue"),
    locations = cells_body(
      columns = vars(year),
      rows = angola_summary$year == most_recent_year
    )
  ) %>%
  tab_header(title = "Résumé des cas de VIH en Angola par année")
```

Résumé des cas de VIH en Angola par année

year	Total_Cases
1990	7500
1991	8500
1992	9500
1993	11000
1994	12000
1995	13000
1996	15000
1997	16000
1998	17000
1999	18000
2000	19000
2001	20000
2002	21000
2003	22000
2004	23000
2005	23000
2006	24000
2007	24000
2008	24000
2009	24000

PRACTICE





Construire un rapport simple en utilisant un fichier R markdown

Maintenant que vous avez deux sorties bien préparées – un graphique en ligne et un tableau statistique pour les données sur le VIH en Angola – vous pouvez les compiler dans un rapport simple et cohérent en utilisant R Markdown. Ce rapport mettra non seulement en valeur vos compétences analytiques, mais aussi votre capacité à communiquer efficacement les résultats.

Étapes à suivre :

1. Créer un nouveau fichier RMarkdown
2. Rédiger l'en-tête YAML
3. Ajouter une section de préparation des données
4. Ajouter un graphique en ligne et un tableau statistique
5. Tricoter le fichier RMarkdown



Construire un rapport pour plusieurs pays (par exemple, “Angola”, “Nigeria”, “Mali”)

En plus de l’Angola, nous souhaitons créer le même rapport pour le Nigeria et le Mali. Ce défi vous demande d’étendre l’approche analytique utilisée pour l’Angola à ces pays supplémentaires.

Y a-t-il une méthode plus rationalisée pour faire cela sans dupliquer nos lignes de code ?

Principe DRY : Ne vous répétez pas !

Approche 1 : Produire une sortie grâce à une fonction personnalisée

- La fonction `generate_country_report()` est créée pour automatiser la génération de rapports pour un pays spécifié à l'aide d'un ensemble de données.
- Elle incarne le principe DRY (Don't Repeat Yourself).
- Cela est réalisé en centralisant les tâches répétitives au sein d'une seule fonction, évitant ainsi la redondance.

- **DRY (Don't Repeat Yourself)** : Un principe fondamental de développement logiciel axé sur la minimisation de la répétition.
- Encourage le remplacement des schémas logiciels répétés par des abstractions ou une normalisation des données pour éliminer la redondance.
- Prône l'utilisation de fonctions ou de méthodes pour remplacer les blocs de code répétitifs.
- Le principe DRY est largement considéré comme essentiel dans de nombreux langages de programmation et méthodologies de conception.

```
generate_country_report <- function(data, country_name) {

  # Tracer les données
  country_data_plot <- subset(data, country == country_name)
  p <- ggplot(country_data_plot, aes(x = year, y = new_cases)) +
    geom_line() +
    theme_minimal() +
    labs(title = paste("Incidence du VIH en", country_name, "(1990-2000)"),
         x = "Année",
         y = "Nouveaux Cas")
  print(p)

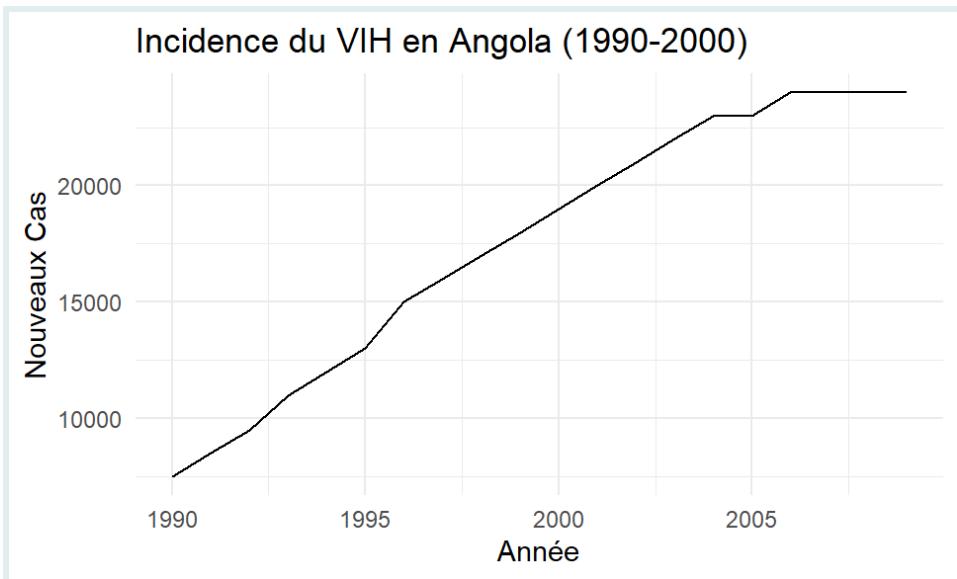
  # Créer le tableau récapitulatif
  country_data_table <- data %>%
    filter(country == country_name) %>%
    group_by(year) %>%
    summarise(Total_Cases = sum(new_cases))

  # Identifier l'année la plus récente
  most_recent_year <- max(country_data_table$year)

  # Afficher le tableau en utilisant kable et mettre en évidence l'année la plus récente
  table_output <- country_data_table %>%
    gt() %>%
    tab_style(
      style = cell_fill(color = "lightblue"),
      locations = cells_body(
        columns = vars(year),
        rows = angola_summary$year == most_recent_year
      )
    ) %>%
    tab_header(title = "Résumé des cas de VIH en Angola par année")

  print(table_output)
}

# Exemple d'utilisation de la fonction
generate_country_report(hiv_data, "Angola")
```



Expliquons ce code :

1. **Déclaration de la fonction** : La fonction `generate_country_report()` prend deux paramètres : `data` (l'ensemble de données) et `country_name` (le nom du pays pour lequel le rapport doit être généré).

```
generate_country_report <- function(data, country_name) {
  .....
}
```

2. **Tracé des Données** : La fonction commence par isoler les données pour le pays spécifié. Elle crée ensuite un graphique en ligne des nouveaux cas de VIH au fil des années en utilisant `{ggplot}`.

```
# Tracé des données
country_data_plot <- subset(data, country == country_name)
p <- ggplot(country_data_plot, aes(x = year, y = new_cases)) +
  geom_line() +
  theme_minimal() +
  labs(title = paste("Incidence du VIH en", country_name, "(1990-2000)"),
       x = "Année",
       y = "Nouveaux Cas")
print(p)
```

3. **Création de Tableau Récapitulatif** : Ensuite, elle filtre les données pour le pays spécifié, les regroupe par année et calcule le nombre total de nouveaux cas pour chaque année.

```
# Création du tableau récapitulatif
country_data_table <- data %>%
  filter(country == country_name) %>%
  group_by(year) %>%
  summarise(Total_Cases = sum(new_cases))
```

- 4. Mise en Évidence des Données Récentes :** La fonction identifie l'année la plus récente dans les données et la met en évidence dans le tableau récapitulatif.

```
# Identifier l'année la plus récente
most_recent_year <- max(country_data_table$year)
```

- 5. Affichage du Tableau :** Le tableau récapitulatif est affiché en utilisant `kable()` et `kable_styling()` pour une sortie HTML propre et interactive. La ligne correspondant à l'année la plus récente est mise en évidence.

```
# Afficher le tableau en utilisant kable et mettre en évidence l'année la
# plus récente
table_output <- country_data_table %>%
  gt() %>%
  tab_style(
    style = cell_fill(color = "lightblue"),
    locations = cells_body(
      columns = vars(year),
      rows = angola_summary$year == most_recent_year
    )
  ) %>%
  tab_header(title = "Résumé des cas de VIH en Angola par année")
```

- 6. Impression des Sorties :** Le graphique (`p`) et le tableau (`table_output`) sont imprimés dans la sortie.

```
print(table_output)
```



- **Programmation Fonctionnelle Utilisant {purrr} :**
 - Étend le processus de génération de rapports pour gérer plusieurs pays.
 - Utilise les paradigmes de programmation fonctionnelle en R, en particulier avec le package `{purrr}`.
 - Facilite l'itération sur une liste de pays.
 - Applique la fonction `generate_country_report()` à chaque pays de la liste.
 - Rationalise le processus tout en respectant le principe DRY.

Cette approche basée sur les fonctions rend le code plus organisé et lisible et améliore également sa réutilisabilité et sa scalabilité, des aspects cruciaux des pratiques de programmation efficaces.

Utilisation de `map()`

- Vous avez une liste de pays pour lesquels des rapports doivent être générés.
- Utilisez la fonction `map()` du package `{purrr}` comme outil pour la tâche.
- La fonction `map()` permet d'appliquer la fonction `generate_country_report()` à chaque pays de la liste.
- Un exemple est fourni pour illustrer l'utilisation de `map()` en conjonction avec `generate_country_report()`.

```
library(purrr)
# Vecteur de pays
countries <- c("Angola", "Nigeria", "Mali")

# En supposant que 'hiv_data' soit votre jeu de données
# Appliquer 'generate_country_report' à chaque pays
map(countries, ~generate_country_report(hiv_data, .))
```

Explication

- `countries` est une liste de noms de pays pour la génération de rapports.
- `map()` applique `generate_country_report()` à chaque pays, en utilisant `~` pour créer une formule dans laquelle `.` est le pays actuel.
- Les rapports pour tous les pays de `countries` sont générés en appelant `generate_country_report(hiv_data, .)`.

Cette méthode rationalise la création de rapports cohérents pour plusieurs pays, économisant du temps grâce à l'automatisation.

Construction d'un Rapport Paramétrisé

Configuration de l'En-tête YAML

- **Définition des Paramètres :** Les paramètres sont définis dans l'en-tête YAML. C'est la section au début de votre fichier R Markdown où vous spécifiez divers paramètres pour votre document, tels que son titre, son format de sortie et ses paramètres.

Exemple avec un Jeu de Données sur le VIH

Considérez un fichier R Markdown conçu pour générer un rapport sur les données du VIH pour différents pays. Vous pouvez définir un paramètre dans l'en-tête YAML pour changer dynamiquement le pays pour lequel le rapport est généré.

YAML dans le fichier R Markdown :

```
---
```

```
title: "Rapport VIH par Pays"
output: html_document
params:
  country: "Angola"
---
```

Explication

1. Titre du Document et Sortie : Dans l'en-tête YAML, le document est intitulé "Rapport VIH par Pays", et le format de sortie est spécifié comme un document HTML.

2. Définition du Paramètre :

- `params` : Ce champ est utilisé pour définir des paramètres.
- `country` : Sous `params`, un paramètre nommé `country` est déclaré. Ce paramètre contrôlera les données du pays utilisées dans le rapport.
- "Angola" : La valeur par défaut pour le paramètre `country` est définie à "Angola". Cela signifie que lorsque le rapport est généré sans spécifier un autre pays, il utilisera automatiquement les données pour l'Angola.

Utilisation du Paramètre dans le Document

Dans le document R Markdown, vous pouvez référencer ce paramètre en utilisant `params$country`. Par exemple, si vous avez une fonction comme `generate_country_report()`, vous pouvez l'appeler dans un bloc de code comme `generate_country_report(data, params$country)`. Cela utilisera la valeur du paramètre `country` pour déterminer les données du pays sur lesquelles rapporter.



En utilisant des paramètres, votre document R Markdown devient plus adaptable. Si vous devez générer un rapport pour un autre pays, vous changez simplement la valeur du paramètre lors de la tricotage du document, plutôt que de modifier le code lui-même. Cela rend votre document plus efficace et polyvalent, en particulier lorsqu'il s'agit de rapports nécessitant des mises à jour fréquentes ou des modifications basées sur différents sous-ensembles de données.

Mise en Œuvre des Paramètres dans l'Analyse

- **Filtrage des Données Basé sur les Paramètres** : Utilisez les paramètres pour filtrer dynamiquement le jeu de données sur la tuberculose (TB). Par exemple, si le rapport concerne un pays et une année spécifiques, vous pouvez filtrer les données en conséquence en utilisant `dplyr`.

Exemple de fragment de code

```
filtered_data <- tb_data %>%
  filter(country == params$country, year == params$year)
```

Explication de la Cartographie des Paramètres

- **Accès aux Paramètres** : Les paramètres définis dans l'en-tête YAML de votre document R Markdown peuvent être accessibles dans le code en utilisant `params$nom_du_paramètre`. Cela est similaire à la manière dont vous accédez aux colonnes dans un dataframe en utilisant `dataframe$nom_de_la_colonne`.
- **Rapport Dynamique** : Lorsque vous créez des visualisations ou des résumés, vous utilisez `params$country` et `params$year` pour garantir que la sortie reflète le pays et l'année spécifiques définis en tant que paramètres. Cela signifie que le contenu du rapport s'adaptera en fonction des valeurs fournies pour ces paramètres au moment du tricotage du document.

SIDE NOTE



Dans le contexte de R Markdown avec des paramètres, `params$nom_du_paramètre` accède à la valeur d'un paramètre nommé `nom_du_paramètre` qui a été défini dans l'en-tête YAML. Cela permet de générer des rapports dynamiques et interactifs basés sur ces valeurs de paramètres.

Dans l'ensemble, l'opérateur `$` est une partie clé de la syntaxe R qui facilite la manipulation des éléments dans les objets R.

Tricotage avec Différents Paramètres

- **Tricotage depuis RStudio** : Expliquez comment tricoter le document avec différentes valeurs de paramètres directement à partir du bouton de tricotage de RStudio, où vous pouvez spécifier les valeurs des paramètres dans une boîte de dialogue.
- **Tricotage en Ligne de Commande** : Pour une génération de rapport automatisée, vous pouvez utiliser des commandes R pour tricoter le document avec différents paramètres. Cela est particulièrement utile pour le traitement par lots ou la génération de rapport automatisée.

- Exemple de commande :

```
rmarkdown::render("TB_Report.Rmd", params = list(country = "Angola"))
```

Tricotage avec Différents Paramètres

La création de rapports dynamiques et flexibles dans R Markdown implique souvent de tricoter le document avec des paramètres variables. Il existe deux principales manières de le faire : directement via l'interface de RStudio et via la ligne de commande pour l'automatisation.

Tricotage depuis RStudio

- **Utilisation du Bouton de Tricotage de RStudio** : Lorsque vous tricotez un document R Markdown dans RStudio qui contient des paramètres, une boîte de dialogue apparaîtra. Cette boîte de dialogue vous permet de spécifier les valeurs pour chaque paramètre avant de tricoter.
- **Sélection Interactive des Paramètres** : Cette méthode est conviviale et interactive, idéale pour la génération de rapport ad hoc où vous pouvez manuellement entrer ou modifier les valeurs des paramètres chaque fois que vous tricotez le document.
- **Personnalisation des Paramètres** : Dans la boîte de dialogue, vous pouvez personnaliser les paramètres selon vos besoins. Par exemple, si vous avez un paramètre pour le pays, vous pouvez changer sa valeur pour générer un rapport pour un autre pays.

Tricotage en Ligne de Commande

- **Génération de Rapport Automatisée** : Pour des scénarios où vous devez automatiser le processus de génération de rapport, comme dans le traitement par lots ou les rapports planifiés, vous pouvez tricoter des documents en utilisant des commandes R.
- **Exemple de Commande** :

```
rmarkdown::render("hiv_incidence.Rmd", params = list(country = "Angola"))
```

Cette commande utilise la fonction `render` du package `rmarkdown`. Elle spécifie le fichier à rendre (`"hiv_incidence.Rmd"`) et définit l'argument `params` à une liste de valeurs de paramètres souhaitées. Dans cet exemple, le paramètre `pays` est défini sur "Angola".

- **Flexibilité et Scripting** : Le tricotage en ligne de commande offre plus de flexibilité et peut être intégré dans des scripts pour des flux de travail automatisés. Cela est particulièrement utile lorsqu'on travaille avec de grands

ensembles de données ou qu'on doit générer plusieurs rapports périodiquement.

Les deux méthodes offrent les moyens d'exploiter la puissance des rapports paramétrés dans R Markdown, répondant à différents besoins — de la génération de rapport interactive et pilotée par l'utilisateur à des processus automatisés et dirigés par des scripts. Cette flexibilité est l'une des forces clés de l'utilisation de R Markdown pour l'analyse de données et la création de rapports.

Le Processus Complet

Créer un rapport paramétré en R implique plusieurs étapes clés, de la configuration de votre projet R à l'automatisation du processus de génération de rapport. Passons en revue ces étapes en détail :

Étape 0 : Créer un Projet R

- **Fichier .Rproj** : Commencez par créer un projet R avec un fichier `.Rproj`. Ce fichier aide à gérer les chemins et les paramètres, rendant plus facile de travailler avec des fichiers et des données dans le projet.

Étape 1 : En-tête YAML

Voici un exemple d'en-tête YAML pour votre document R Markdown :

```
---
title: "Rapport - `r stringr::str_to_title(params$state)`"
date: "`r format(Sys.time(), '%d %B %Y')`"
output: html_document
editor_options:
  chunk_output_type: console
params:
  state: "Alabama"
---
```

Cet en-tête configure le titre du document, la date, le format de sortie et définit un paramètre `state` avec une valeur par défaut de “Alabama”.

Étape 2 : Configuration de l'Environnement du Document

Dans le premier bloc de code, configurez votre environnement en chargeant les packages requis et votre jeu de données :

```

# Configuration des options globales
knitr::opts_chunk$set(warning = FALSE, message = FALSE, echo = FALSE)

# Chargement des packages
pacman::p_load(ggplot2, dplyr, knitr, kableExtra)

# Chargement du jeu de données
hiv_data <- read.csv("hiv_incidence.csv")

```

Étape 3 : Le Cœur de Votre Rapport

Maintenant, ajoutez des sections avec des sorties, générées dynamiquement en fonction de l'état sélectionné.

Tendance de l'Incidence du VIH

```

hiv_incidence_data <- subset(hiv_data, country == params$country)
ggplot(hiv_incidence_data, aes(x = year, y = new_cases)) +
  geom_line() +
  theme_minimal() +
  labs(title = paste("Incidence du VIH en", params$country, "(1990-2000)"),
       x = "Année",
       y = "Nouveaux Cas")

```

Tableau Récapitulatif des Cas de VIH

```

## Tableau Récapitulatif des Cas de VIH

# Filtrer pour les données du pays sélectionné
country_data <- hiv_data %>% filter(country == params$country)

# Résumer les données par année
country_summary <- country_data %>%
  group_by(year) %>%
  summarise(Total_Cases = sum(new_cases))

# Identifier l'année la plus récente
most_recent_year <- max(country_summary$year)

# Afficher le tableau en utilisant kable et mettre en évidence l'année la plus récente
country_summary %>%
  kable("html", caption = paste("Résumé des cas de VIH en", params$country,
                                "par année")) %>%
  kable_styling(bootstrap_options = c("striped", "hover")) %>%
  row_spec(which(country_summary$year == most_recent_year), background =
            "lightblue")

```

Étape 4 : Tricoter le Rapport

Le document peut maintenant être tricoté pour générer un rapport pour l'état par défaut ou spécifié.

CHALLENGE



Créez un nouveau R Markdown avec ces composants et tricotez-le pour voir la sortie.

Étape 5 : Créer un Script R pour Paramétriser Tout le Processus

Pour l'automatisation, utilisez un script R pour paramétriser le processus :

```
# Chargement des packages nécessaires
pacman::p_load(rmarkdown, purrr, stringr)

# Définir un vecteur d'états
states <- c("Alabama", "Alaska", "Arizona")

# Générer une tibble pour les rapports
reports <- tibble(
  filename = str_c("state_report_", states, ".html"),
  params = map(states, ~list(state = .))
)

# Utiliser pwalk pour rendre chaque rapport
reports %>%
  select(output_file = filename, params) %>%
  pwalk(rmarkdown::render, input = "state_report.Rmd", output_dir = "output/")
```

Ce script utilise `map()` et `pwalk()` pour itérer sur chaque état, rendant un rapport individualisé pour chacun.

La fonction `map()` crée une liste de paramètres pour chaque état, et `pwalk()` applique la fonction `rmarkdown::render()` à chaque ensemble de paramètres, générant les rapports.

Avec ces étapes, vous pouvez efficacement générer des rapports personnalisés pour plusieurs états ou critères, rationalisant votre flux de travail d'analyse de données et de création de rapports en R.

CONCLUSION !

Pour conclure, nous avons exploré un flux de travail complet pour créer des rapports paramétrés en utilisant R Markdown. Ce processus permet de générer des rapports

dynamiques et efficaces adaptés à des critères spécifiques, tels que des régions géographiques ou des cadres temporels.

Résultats d'apprentissage

À la fin de cette leçon, vous devez être capable de :

- **Gérer Efficacement les Projets R** : Organiser et gérer des projets R pour diverses tâches d'analyse de données.
- **Produire des Documents R Markdown Dynamiques** : Générer des documents R Markdown qui s'adaptent à différents entrées de données.
- **Réaliser des Analyses de Données et des Visualisations** : Analyser des données et créer des visualisations, en se concentrant sur des jeux de données réels comme l'incidence du VIH.
- **Développer des Scripts d'Automatisation pour les Rapports** : Créer des scripts pour automatiser la génération de rapports basés sur des paramètres, améliorant la cohérence et l'efficacité.
- **Adopter les Meilleures Pratiques en Programmation R** : Mettre en œuvre les meilleures pratiques pour une programmation R plus efficace, lisible et maintenable.
- **Améliorer les Compétences de Crédit de Rapport** : Améliorer les capacités à élaborer des rapports complets, informatifs et visuellement attrayants pour divers publics.
- **Utiliser des Techniques de Programmation Fonctionnelle** : Appliquer des méthodes de programmation fonctionnelle en R pour un traitement de données et une création de rapports rationalisés.

Bon rendu !

Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



IMAD EL BADISY

Data Science Education Officer
Deeply interested in health data



KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement



JOY VAZ

R Developer and Instructor, the GRAPH Network
Loves doing science and teaching science

Références

- Johnson, Paul. "R Markdown: Le Guide Définitif : Yihui Xie, JJ Allaire, et Garrett Grolemund. Boca Raton, FL : Chapman & Hall/CRC Press, 2018, xxxiv+ 303 pp., \$38.95 (P), ISBN : 978-1-13-835933-8

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.

