

TP de Biométrie Semestre 4

Benoît Simon-Bouhet

dimanche 17 mars 2024

Table des matières

| | |
|---|-----------|
| Introduction | 4 |
| Objectifs | 4 |
| Pré-requis | 4 |
| Organisation | 5 |
| Volume de travail | 5 |
| Modalités d'enseignement | 6 |
| Utilisation de Slack | 7 |
| Progression conseillée | 9 |
| Évaluation(s) | 11 |
| Licence | 11 |
| | |
| 1 Exploration statistique des données | 13 |
| 1.1 Pré-requis | 13 |
| 1.2 Créer des résumés avec la fonction <code>summarise()</code> | 15 |
| 1.2.1 Principe de la fonction <code>summarise()</code> . | 16 |
| 1.2.2 Intérêt de l'argument <code>.by</code> | 19 |
| 1.2.3 Grouper par plus d'une variable . . . | 22 |
| 1.2.4 Un raccourci pratique pour compter des effectifs | 25 |
| 1.2.5 Exercices | 28 |
| 1.3 Les fonctions <code>pivot_wider()</code> et <code>pivot_longer()</code> | 31 |
| 1.3.1 Du format long au format large : <code>pivot_wider()</code> | 31 |
| 1.3.2 Du format large au format long : <code>pivot_longer()</code> | 33 |
| 1.4 Créer des résumés avec la fonction <code>reframe()</code> | 34 |
| 1.5 Créer des résumés de données avec des fonc- tions spécifiques | 41 |
| 1.5.1 La fonction <code>summary()</code> | 42 |
| 1.5.2 La fonction <code>skim()</code> | 51 |
| 1.5.3 Exercice | 55 |
| | |
| 2 Dispersion et incertitude | 57 |
| 2.1 Pré-requis | 57 |
| 2.2 La notion de dispersion | 57 |
| 2.3 La notion d'incertitude | 58 |

| | | |
|----------|--|-----------|
| 2.4 | Calcul de l'erreur standard de la moyenne . . . | 60 |
| 2.5 | Calculs d'intervalles de confiance à 95% . . . | 66 |
| 3 | Visualiser l'incertitude et la dispersion | 73 |
| 3.1 | Pré-requis | 73 |
| 3.2 | Position et dispersion : les histogrammes . . . | 73 |
| 3.3 | Position et dispersion : les stripcharts | 76 |
| 3.4 | Position et dispersion : les boxplots | 78 |
| 3.5 | Visualiser l'incertitude : les barres d'erreur . . | 81 |
| 3.6 | Visualiser l'incertitude : les boîtes à moustaches | 87 |
| 3.7 | Exercice | 89 |
| | References | 90 |

Introduction

Objectifs

Ce livre contient l'ensemble du matériel (contenus, exemples, exercices...) nécessaire à la réalisation des travaux pratiques sous R pour la **Biométrie** de l'EC '*Outils pour l'étude et la compréhension du vivant 3*' du semestre 4 de la licence Sciences de la Vie de La Rochelle Université.

À l'issue des 4 séances prévues ce semestre (2 TP et 2 TEA), vous devriez être capables de faire les choses suivantes dans le logiciel RStudio :


- Explorer des jeux de données en produisant des résumés statistiques de variables de différentes natures (numériques continues ou catégorielles) et en produisant des graphiques appropriés.
- Être capables de distinguer les notions de dispersion et d'incertitude.
- Calculer des statistiques descriptives permettant de déterminer la position (moyennes, médianes, quartiles...) et la dispersion des données (écart-types, variances, intervalles inter-quartiles...) pour plusieurs sous-groupes de vos jeux de données, et les représenter sur des graphiques adaptés.
- Calculer des indices d'incertitude (erreurs standard, intervalles de confiance...) pour plusieurs sous-groupes de vos jeux de données, et les représenter sur des graphiques adaptés.

Pré-requis

Pour atteindre les objectifs fixés ici, et compte tenu du volume horaire restreint qui est consacré aux TP et TEA de

Biométrie au S4, vous devez obligatoirement posséder un certain nombre de pré-requis. En particulier, vous devriez avoir à ce stade une bonne connaissance de l'interface des logiciels R et RStudio, et vous devriez être capables :

1. de créer un Rproject et un script d'analyse dans RStudio
2. d'importer des jeux de données issus de tableurs dans RStudio
3. d'effectuer des manipulations de données simples (sélectionner des variables, trier des colonnes, filtrer des lignes, créer de nouvelles variables, etc.)
4. de produire des graphiques de qualité, adaptés à la fois aux variables dont vous disposez et aux questions auxquelles vous souhaitez répondre.

 Si ces pré-requis ne sont pas maîtrisés

Mettez-vous à niveau de toute urgence en lisant attentivement [le livre en ligne de Biométrie du semestre 3](#)

Organisation

Volume de travail

Les travaux pratiques et TEA de biométrie auront lieu entre le 19 mars et le 5 avril 2024 :

- Entre le mardi 19 et le lundi 25 mars : 1 séance de TP d'1h30 et 1 séance de TEA d'1h30
- Entre le mardi 2 et le vendredi 5 avril : 1 séance de TP d'1h30 et 1 séance de TEA d'1h30

Tous les TP ont lieu dans les salles du Pôle Communication, Multimédia, Réseaux. Tous les TEA sont à distance.

Au total, chaque groupe aura donc 2 séances de TP et 2 séances de TEA, soit un total de 6 heures prévues dans vos emplois du temps. C'est peu pour atteindre les objectifs fixés et il y aura donc évidemment du travail personnel à fournir en dehors de ces séances. J'estime que vous devrez fournir 3 à 6 heures de travail personnel en plus des séances prévues

dans votre emploi du temps. Attention donc : pensez bien à prévoir du temps dans vos plannings car le travail personnel est essentiel pour progresser dans cette matière. J'insiste sur l'importance de faire l'effort dès maintenant : vous allez en effet avoir des enseignements qui reposent sur l'utilisation de ces logiciels jusqu'à la fin du S6 (y compris pendant vos stages et, très vraisemblablement, dans vos futurs masters également). C'est donc maintenant qu'il faut acquérir des automatismes, cela vous fera gagner énormément de temps ensuite.

Modalités d'enseignement

Pour suivre cet enseignement vous pourrez utiliser les ordinateurs de l'université, mais je ne peux que vous encourager à utiliser vos propres ordinateurs, sous Windows, Linux ou MacOS. Lors de vos futurs stages et pour rédiger vos comptes-rendus de TP, vous utiliserez le plus souvent vos propres ordinateurs, autant prendre dès maintenant de bonnes habitudes en installant les logiciels dont vous aurez besoin tout au long de votre licence. Si vous n'avez pas suivi la biométrie du semestre 3 et que les logiciels R et RStudio ne sont pas encore installés sur vos ordinateurs, suivez [la procédure décrite ici](#). Si vous ne possédez pas d'ordinateur, manifestez vous rapidement auprès de moi car des solutions existent (prêt par l'université, travail sur tablette via [RStudio cloud...](#)).

! Important

L'essentiel du contenu de cet enseignement peut être abordé en autonomie, à distance, grâce à ce livre en ligne, aux ressources mises à disposition sur Moodle et à votre ordinateur personnel. Cela signifie que **la présence physique lors des séances de TP n'est pas obligatoire**.

Plus que des séances de TP classiques, considérez plutôt qu'il s'agit de **permanences non-obligatoires** : si vous pensez avoir besoin d'aide, si vous avez des points de blocage ou des questions sur le contenu de ce document ou sur les exercices demandés, alors venez poser vos questions lors des séances de TP. Vous ne serez d'ailleurs pas tenus de rester pendant

1h30 : si vous obtenez une réponse en 10 minutes et que vous préférez travailler ailleurs, vous serez libres de repartir !

De même, si vous n'avez pas de difficulté de compréhension, que vous n'avez pas de problème avec les exercices de ce livre en ligne, votre présence n'est pas requise : tant que le travail demandé est fait, libre à vous de choisir votre façon de travailler. Bien entendu, si vous souhaitez venir en salle de TP pour travailler au calme et dans un cadre plus formel, même si vous n'avez pas de questions à poser : aucun problème, vous y serez toujours les bienvenus et je serai présent systématiquement, pour toutes les séances et tous les groupes.

Ce fonctionnement très souple a de nombreux avantages :

- vous vous organisez comme vous le souhaitez
- vous ne venez que lorsque vous en avez vraiment besoin
- celles et ceux qui se déplacent reçoivent une aide personnalisée
- vous travaillez sur vos ordinateurs
- les effectifs étant réduits, c'est aussi plus confortable pour moi !

Toutefois, pour que cette organisation fonctionne, cela demande de la rigueur de votre part, en particulier sur la régularité du travail que vous devez fournir. Si la présence en salle de TP n'est pas requise, **le travail demandé est bel et bien obligatoire** ! Si vous venez en salle de TP sans avoir travaillé en amont, vous risquez de perdre votre temps car vous passerez votre séance à lire et suivre ce livre en ligne, choses que vous pouvez très bien faire chez vous. De même, si vous attendez le 13 avril pour vous y mettre, je ne pourrais pas grand chose pour vous. Je le répète, outre les heures de TP/TEA prévus dans vos emplois du temps, vous devez prévoir au moins 3 à 6 heures de travail personnel supplémentaire.

Je vous laisse donc une grande liberté d'organisation. À vous d'en tirer le maximum et de faire preuve du sérieux nécessaire.

Utilisation de Slack

Comme au semestre précédent, nous pourrons échanger sur [l'application Slack](#). Si vous ne l'avez pas encore fait (vous êtes

une vingtaine dans ce cas !), créez-vous un compte en ligne et installez le logiciel sur votre ordinateur (il existe aussi des versions pour tablettes et smartphones). Lorsque vous aurez installé le logiciel, [cliquez sur ce lien](#) pour vous connecter à notre espace de travail commun intitulé L2 SV 23-24 / EC outils (ce lien expire régulièrement : faites moi signe s'il n'est plus valide). C'est le même espace de travail qu'au semestre précédent et si vous vous y êtes déjà connecté cet automne, vous n'avez plus qu'à relancer le logiciel.

Vous verrez que 3 “chaînes” sont disponibles :

- #général : c'est là que les questions liées à l'organisation générale du cours, des TP et TEA, des évaluations, etc. doivent être posées. Si vous ne savez pas si une séance de permanence a lieu, posez la question ici.
- #questions-rstudio : c'est ici que toutes les questions pratiques liées à l'utilisation de R et RStudio devront être posées. Problèmes de syntaxe, problèmes liés à l'interface, à l'installation des packages ou à l'utilisation des fonctions, à la création des graphiques, à la manipulation des tableaux... Tout ce qui concerne directement les logiciels sera traité ici. Vous êtes libres de poser des questions, de poster des captures d'écran, des morceaux de code, des messages d'erreur. Et **vous êtes bien entendus vivement encouragés à vous entraider et à répondre aux questions de vos collègues**. Je n'interviendrai ici que pour répondre aux questions laissées sans réponse ou si les réponses apportées sont inexactes. Le fonctionnement est celui d'un forum de discussion instantané. Vous en tirerez le plus grand bénéfice en participant et en n'ayant pas peur de poser des questions, même si elles vous paraissent idiotes. Rappelez-vous toujours que si vous vous posez une question, d'autres se la posent aussi probablement.
- #questions-stats : C'est ici que vous pourrez poser vos questions liées aux méthodes statistiques ou aux choix des modèles de dynamique des populations. Tout ce qui ne concerne pas directement l'utilisation du logiciel (comme par exemple le choix d'un test ou des hypothèses nulles et alternatives, la démarche d'analyse, la signification de tel paramètre ou estimateur, le principe de telle ou telle méthode...) peut être discuté ici. Comme pour le canal #questions-rstudio, **vous êtes**

encouragés à vous entraider et à répondre aux questions de vos collègues.

Ainsi, quand vous travaillerez à vos TP ou TEA, que vous soyez installés chez vous ou en salle de TP, prenez l'habitude de garder Slack ouvert sur votre ordinateur. Même si vous n'avez pas de question à poser, votre participation active pour répondre à vos collègues est souhaitable et souhaitée. Je vous incite donc fortement à vous **entraider** : c'est très formateur pour celui qui explique, et celui qui rencontre une difficulté a plus de chances de comprendre si c'est quelqu'un d'autre qui lui explique plutôt que la personne qui a rédigé les instructions mal comprises.

Ce document est fait pour vous permettre d'avancer en autonomie et vous ne devriez normalement pas avoir beaucoup besoin de moi si votre lecture est attentive. L'expérience montre en effet que la plupart du temps, il suffit de lire correctement les paragraphes précédents et/ou suivants pour obtenir la réponse à ses questions. J'essaie néanmoins de rester disponible sur Slack pendant les séances de TP et de TEA de tous les groupes. Cela veut donc dire que même si votre groupe n'est pas en TP, vos questions ont des chances d'être lues et de recevoir des réponses dès que d'autres groupes sont en TP ou TEA. Vous êtes d'ailleurs encouragés à échanger sur Slack aussi pendant vos phases de travail personnel.

Progression conseillée

Si vous avez suivi le document de prise en main de R et RStudio du semestre 3, vous savez que pour apprendre à utiliser ces logiciels, il faut faire les choses soi-même, ne pas avoir peur des messages d'erreurs (il faut d'ailleurs apprendre à les déchiffrer pour comprendre d'où viennent les problèmes), essayer maintes fois, se tromper beaucoup, recommencer, et surtout, ne pas se décourager. J'utilise ce logiciel presque quotidiennement depuis plus de 15 ans et à chaque session de travail, je rencontre des messages d'erreur. Avec suffisamment d'habitude, on apprend à les déchiffrer, et on corrige les problèmes en quelques secondes. Ce livre est conçu pour vous faciliter la tâche, mais ne vous y trompez pas, vous rencontrerez des difficultés, et c'est normal. C'est le prix à payer pour profiter de la puissance du meilleur logiciel permettant

d'analyser des données, de produire des graphiques de qualité et de réaliser toutes les statistiques dont vous aurez besoin d'ici la fin de vos études et au-delà.

Pour que cet apprentissage soit le moins problématique possible, il convient de prendre les choses dans l'ordre. C'est la raison pour laquelle les 3 chapitres de ce livre doivent être lus dans l'ordre, et les exercices d'application faits au fur et à mesure de la lecture.

Idéalement, voilà les étapes que vous devriez avoir franchi chaque semaine :

1. À l'issue de la première séance de TP et de la première séance de TEA, vous devriez avoir compris comment calculer et interpréter des résumés statistiques de vos jeux de données (c'est le premier chapitre de ce livre en ligne). Vous devriez en particulier être capable de calculer des estimateurs de position (moyennes, médianes, quartiles...) et de dispersion (variances, écart-types, intervalles inter-quartiles...) sur des variables numériques, et ce, pour plusieurs modalités d'une variable catégorielle ou pour chaque combinaison de modalités de plusieurs variables catégorielles (par exemple, quelles sont les moyennes et variances des longueurs de becs pour chaque espèce de manchots et chaque sexe). Vous devrez donc être capables d'utiliser les fonctions `group_by()` et `summarise()` du package `dplyr`. Cela suppose bien sûr que vous soyez au clair sur les pré-requis évoqués plus haut (**?@sec-prerequis**) avant d'aborder le premier chapitre de ce livre en ligne.
2. À l'issue de la seconde séance de TP, Vous devrez être capables de distinguer la notion de dispersion de celle de précision. Vous devrez être capable d'expliquer clairement la différence entre ces 2 notions, et vous devrez savoir à quoi servent les indices de dispersion et d'incertitude. Vous devrez être capables de calculer des indices d'incertitude, en particulier l'erreur standard de la moyenne (ou erreur type) et l'intervalle de confiance de la moyenne (chapitre 2). Vous devrez en outre être capables de produire des graphiques sur lesquels apparaissent des barres d'incertitude (erreurs standards ou intervalles de confiance, chapitre 3). Là encore, cela suppose que vous soyez au clair avec les représentations

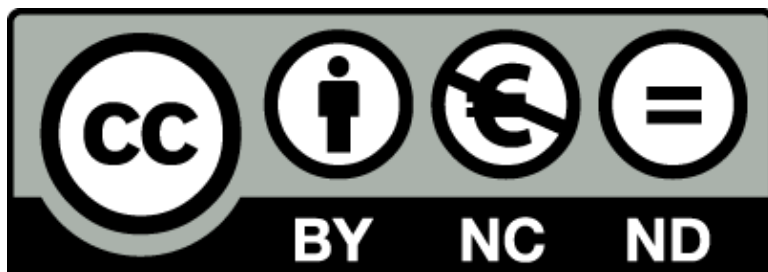
graphiques abordées au semestre 3 (comment produire un graphique avec `ggplot2`, quel graphique choisir pour quelles données et quel objectif ?...)

Évaluation(s)

L'évaluation de la partie "Biométrie" de l'EC "Outils pour l'étude et la compréhension du vivant 3" sera conduite par mes collègues en charge des cours magistraux, travaux dirigés, et travaux pratiques "classiques" (Fanny Cusset et Gérard Blanchard). Il est bien évident toutefois que mes collègues attendent une bonne maîtrise des notions développées ici, et qu'au-delà de la biométrie, les autres collègues intervenant dans l'EC "outils pour l'étude et la compréhension du vivant 3" attendent eux aussi que vous mettiez en pratique ce que vous apprenez ici dans vos futurs compte-rendus de TP.

Licence

Ce livre est ligne est sous licence Creative Commons ([CC BY-NC-ND 4.0](https://creativecommons.org/licenses/by-nc-nd/4.0/))



Vous êtes autorisé à partager, copier, distribuer et communiquer ce matériel par tous moyens et sous tous formats, tant que les conditions suivantes sont respectées :

④ **Attribution** : vous devez créditer ce travail (donc citer son auteur), fournir un lien vers ce livre en ligne, intégrer un lien vers la licence Creative Commons et indiquer si des modifications du contenu original ont été effectuées. Vous devez indiquer ces informations par tous les moyens raisonnables, sans toutefois suggérer que l'auteur vous soutient ou soutient la façon dont vous avez utilisé son travail.

⊗ **Pas d'Utilisation Commerciale** : vous n'êtes pas autorisé à faire un usage commercial de cet ouvrage, ni de tout ou partie du matériel le composant. Cela comprend évidemment la diffusion sur des plateformes de partage telles que studocu.com qui tirent profit d'œuvres dont elles ne sont pas propriétaires, souvent à l'insu des auteurs.

⊖ **Pas de modifications** : dans le cas où vous effectuez un remix, que vous transformez, ou créez à partir du matériel composant l'ouvrage original, vous n'êtes pas autorisé à distribuer ou mettre à disposition l'ouvrage modifié.

🔒 **Pas de restrictions complémentaires** : vous n'êtes pas autorisé à appliquer des conditions légales ou des mesures techniques qui restreindraient légalement autrui à utiliser cet ouvrage dans les conditions décrites par la licence.

1 Exploration statistique des données

1.1 Pré-requis

La première étape de toute analyse de données est l'**exploration**. Avant de se lancer dans des tests statistiques et des procédures complexes, et à supposer que les données dont vous disposez sont déjà dans un format approprié, il est toujours très utile :

1. d'explorer visuellement les données dont on dispose en faisant des graphiques nombreux et variés, afin de comprendre, notamment quelle est la distribution des variables numériques, quelles sont les catégories les plus représentées pour les variables qualitatives (ou facteurs), quelles sont les relations les plus marquantes entre variables numériques et/ou catégorielles, etc. Vous avez déjà appris, au semestre 3, comment produire toutes sortes de graphiques avec le package `ggplot2`. Si vous avez besoin de revoir les bases, [c'est là que ça se passe](#).
2. d'explorer les données en calculant des indices de statistiques descriptives. Ces indices relèvent en général de 2 catégories : les indices de position (e.g. moyennes, médianes, quartiles...) et les indices de dispersion (e.g. variance, écart-type, intervalle inter-quartiles...). Nous allons voir dans ce chapitre comment calculer ces indices dans plusieurs situations, notamment lorsque l'on souhaite les calculer pour plusieurs sous-groupes d'un jeu de données.

Nous verrons dans le chapitre suivant comment calculer des indices d'incertitude (Section 2.4 et Section 2.5). Attention, il ne faudra pas confondre indices de dispersion et indices d'incertitude.

Afin d'explorer ces questions, nous aurons besoin des pa-ckages suivants :

```
library(tidyverse)
library(skimr)
library(palmerpenguins)
library(nycflights13)
```

Les packages du `tidyverse` (Wickham 2022) permettent de manipuler facilement des tableaux de données et de réaliser des graphiques. Charger le `tidyverse` permet d'accéder, entre autres, aux packages `readr` (Wickham, Hester, et Bryan 2022), pour importer facilement des fichiers `.csv` au format `tibble`, `dplyr` (Wickham et al. 2023) pour manipuler des tableaux de données ou encore `ggplot2` (Wickham et al. 2022) pour produire des graphiques. Le package `skimr` (Waring et al. 2022) permet de calculer des résumés de données très informatifs. Les packages `palmerpenguins` (Horst, Hill, et Gorman 2022) et `nycflights13` (Wickham 2021) fournissent des jeux de données qui seront faciles à manipuler pour illustrer ce chapitre (et les suivants).

Important

Même si vous avez déjà installé le `tidyverse` ou `dplyr` au semestre précédent, ré-installez `dplyr` avec `install.packages("dplyr")`. Ce package a en effet été mis à jour tout récemment, et nous aurons besoin de sa toute dernière version (v1.1.0). Chargez-le ensuite en mémoire avec `library(dplyr)`.

Attention

Pensez à installer tous les packages listés ci-dessous avant de les charger en mémoire si vous ne l'avez pas déjà fait. Si vous ne savez plus comment faire, consultez d'urgence [la section dédiée aux packages dans le livre en ligne de Biométrie du semestre 3](#).

Pour travailler dans de bonnes conditions, créez un nouveau dossier sur votre ordinateur, créez un `Rproject` et un script dans ce dossier et travaillez systématiquement **dans votre script**, et surtout pas directement dans la console. Là encore,

consultez [le livre en ligne du semestre 3](#) si vous ne savez plus comment faire.

1.2 Créer des résumés avec la fonction `summarise()`

Comme nous l'avons vu au semestre 3, le package `dplyr` fournit plusieurs fonctions qui portent le nom de verbes et qui permettent d'effectuer des manipulations simples mais qui peuvent devenir très puissantes lorsqu'on les combine. Nous avons ainsi vu les fonctions suivantes :

- `select()` : pour sélectionner ou exclure certaines colonnes (variables) d'un tableau de données
- `filter()` : pour trier des lignes d'un tableau de données selon des critères ou conditions choisis par l'utilisateur
- `mutate()` : pour transformer des variables existantes, ou pour créer de nouvelles colonnes dans un tableau de données
- `arrange()` : pour trier des tableaux de données par ordre croissants ou décroissants

Si vous ne savez plus comment utiliser ces fonctions, relisez [le chapitre 4 du livre en ligne de Biométrie du semestre 3](#).

À ces 4 verbes, nous allons ici ajouter :

- `summarise()` : pour créer des résumés de données simples à partir des colonnes d'un tableau
- `reframe()` : pour créer des résumés de données plus élaborés à partir des colonnes d'un tableau
- `count()` : pour compter le nombre d'observations pour chaque niveau d'un facteur (ou modalité d'une variable catégorielle)
- `group_by()` : pour effectuer des opérations pour chaque niveau d'un facteur (ou modalité d'une variable catégorielle)

Cette dernière fonction `group_by()` a été rendue presque obsolète par une mise à jour récente du package `dplyr` qui introduit un nouvel argument pour plusieurs fonctions, dont `summarise()` : l'argument `.by`. Un peu comme `group-by()`, ce nouvel argument permet d'effectuer des opérations pour

chaque niveau d'un facteur (ou modalité d'une variable catégorielle). À notre niveau, les différences entre la fonction `group_by()` et l'argument `.by` ne sont pas importantes. Nous utiliserons donc de préférence la notation la plus simple, celle de l'argument `.by`.

Voyons comment on utilise ces fonctions pour calculer des indices de statistiques descriptives pour les variables du tableau `penguins` :

```
# affichage du tableau
penguins

# A tibble: 344 x 8
  species island  bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>   <fct>         <dbl>         <dbl>         <int>         <int>
1 Adelie Torgersen     39.1          18.7           181           3750
2 Adelie Torgersen     39.5          17.4           186           3800
3 Adelie Torgersen     40.3           18            195           3250
4 Adelie Torgersen     NA            NA             NA            NA
5 Adelie Torgersen     36.7          19.3           193           3450
6 Adelie Torgersen     39.3          20.6           190           3650
7 Adelie Torgersen     38.9          17.8           181           3625
8 Adelie Torgersen     39.2          19.6           195           4675
9 Adelie Torgersen     34.1          18.1           193           3475
10 Adelie Torgersen     42            20.2           190           4250
# i 334 more rows
# i 2 more variables: sex <fct>, year <int>
```

1.2.1 Principe de la fonction `summarise()`

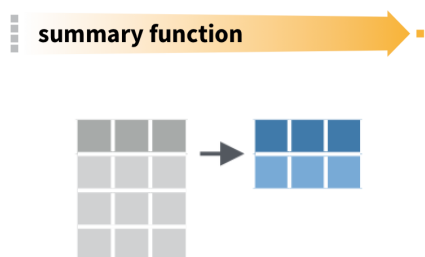


Figure 1.1: Schéma de la fonction `summarise()` tiré de la 'cheatsheet' de `dplyr` et `tidyr`

La Figure 1.1 ci-dessus indique comment travaille la fonction `summarise()` : elle prend plusieurs valeurs (potentiellement, un très grand nombre) et les réduit à **une unique valeur qui les résume**. La valeur qui résume les données est choisie par l'utilisateur. Il peut s'agir par exemple d'un calcul de moyenne, de quartile ou de variance, il peut s'agir de calculer une somme, ou d'extraire la valeur maximale ou minimale, ou encore, il peut tout simplement s'agir de déterminer un nombre d'observations. Mais le fonctionnement est toujours le même : la fonction `summarise()` ne renvoie qu'une unique valeur pour une variable donnée (ou pour chaque modalité d'une variable catégorielle).

Ainsi, pour connaître la moyenne de la longueur du bec des manchots de l'île de Palmer, il suffit d'utiliser le tableau `penguins` du package `palmerpenguins` et sa variable `bill_length_mm` que nous avons déjà utilisée au semestre 3 :

```
penguins |>
  summarise(moyenne = mean(bill_length_mm))

# A tibble: 1 x 1
  moyenne
  <dbl>
1      NA
```

La fonction `mean()` permet de calculer une moyenne. Ici, la valeur retournée est `NA` car 2 individus n'ont pas été mesurés, et le tableau contient donc des valeurs manquantes :

```
penguins |>
  filter(is.na(bill_length_mm))

# A tibble: 2 x 8
  species island  bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>   <fct>          <dbl>          <dbl>          <int>         <int>
1 Adelie  Torgersen         NA              NA              NA             NA
2 Gentoo  Biscoe            NA              NA              NA             NA
# i 2 more variables: sex <fct>, year <int>
```

Pour obtenir la valeur souhaitée, il faut indiquer à R d'exclure les valeurs manquantes lors du calcul de moyenne :

```
penguins |>
  summarise(moyenne = mean(bill_length_mm, na.rm = TRUE))

# A tibble: 1 x 1
  moyenne
  <dbl>
1    43.9
```

La longueur moyenne du bec des manchots (toutes espèces confondues) est donc de 43.9 millimètres.

De la même façon, on peut demander plusieurs calculs d'indices à la fois, par exemple la moyenne et l'écart-type (avec la fonction `sd()`) de la longueur des becs :

```
penguins |>
  summarise(moyenne = mean(bill_length_mm, na.rm = TRUE),
            ecart_type = sd(bill_length_mm, na.rm = TRUE))

# A tibble: 1 x 2
  moyenne ecart_type
  <dbl>     <dbl>
1    43.9         5.46
```

Ici, l'écart-type vaut 5.5 millimètres.

La fonction `summarise()` permet donc de calculer des indices statistiques variés, et permet aussi d'accéder à plusieurs variables à la fois. Par exemple. pour calculer les moyennes, médianes, minima et maxima des longueurs de nageoires et de masses corporelles, on peut procéder ainsi :

```
penguins |>
  summarise(moy_flip = mean(flipper_length_mm, na.rm = TRUE),
            med_flip = median(flipper_length_mm, na.rm = TRUE),
            min_flip = min(flipper_length_mm, na.rm = TRUE),
            max_flip = max(flipper_length_mm, na.rm = TRUE),
            moy_mass = mean(body_mass_g, na.rm = TRUE),
            med_mass = median(body_mass_g, na.rm = TRUE),
```

```

min_mass = min(body_mass_g, na.rm = TRUE),
max_mass = max(body_mass_g, na.rm = TRUE)

```

```
# A tibble: 1 x 8
```

```

  moy_flip med_flip min_flip max_flip moy_mass med_mass min_mass max_mass
  <dbl>   <dbl>   <int>   <int>   <dbl>   <dbl>   <int>   <int>
1    201.    197     172     231    4202.   4050    2700    6300

```

La fonction `summarise()` est donc très utile pour produire des résumés informatifs des données, mais nos exemples ne sont ici pas très pertinents puisque nous avons jusqu'ici calculé des indices sans distinguer les espèces. Si les 3 espèces de manchots ont des caractéristiques très différentes, calculer des moyennes toutes espèces confondues n'a pas de sens. Voyons maintenant comment obtenir ces même indices pour chaque espèce.

1.2.2 Intérêt de l'argument `.by`

La fonction `summarise()` devient particulièrement puissante lorsqu'on y ajoute l'argument `.by` :



Figure 1.2: Fonctionnement de l'argument `.by` travaillant de concert avec `summarise()`, tiré de la 'cheatsheet' de `dplyr` et `tidyr`

Comme son nom l'indique, l'argument `.by` permet de créer des sous-groupes dans un tableau, afin que le résumé des données soit calculé pour chacun des sous-groupes plutôt que sur l'ensemble du tableau. En ce sens, son fonctionnement est analogue à celui des `facets` de `ggplot2`

qui permettent de scinder les données d'un graphique en plusieurs sous-groupes.

Pour revenir à l'exemple de la longueur du bec des manchots, imaginons que nous souhaitions calculer les moyennes et les écart-types pour chacune des trois espèces. Voilà comment procéder :

```
penguins |>
  summarise(moyenne = mean(bill_length_mm, na.rm = TRUE),
            ecart_type = sd(bill_length_mm, na.rm = TRUE),
            .by = species)
```

```
# A tibble: 3 x 3
  species  moyenne ecart_type
  <fct>    <dbl>    <dbl>
1 Adelie   38.8      2.66
2 Gentoo  47.5      3.08
3 Chinstrap 48.8      3.34
```

Ici, les étapes sont les suivantes :

1. On prend le tableau `penguins`, puis
2. On résume les données sous la forme de moyennes et d'écart-types
3. On demande un calcul pour chaque modalité de la variable `species`

Là où nous avons auparavant une seule valeur de moyenne et d'écart-type pour l'ensemble des individus du tableau de données, nous avons maintenant une valeur de moyenne et d'écart-type pour chaque modalité de la variable espèce. Puisque le facteur `species` contient 3 modalités (`Adelie`, `Chinstrap` et `Gentoo`), le résumé des données contient maintenant 3 lignes.

Cette syntaxe très simple est presque équivalente à celle de la fonction `group_by()` :

```
penguins |>
  group_by(species) |>
  summarise(moyenne = mean(bill_length_mm, na.rm = TRUE),
            ecart_type = sd(bill_length_mm, na.rm = TRUE))
```

```
# A tibble: 3 x 3
  species  moyenne ecart_type
  <fct>    <dbl>    <dbl>
1 Adelie   38.8      2.66
2 Chinstrap 48.8      3.34
3 Gentoo   47.5      3.08
```

Les valeurs obtenues sont les mêmes, mais d'une part, les commandes sont fournies avec une syntaxe et dans un ordre différents :

1. On prend le tableau `penguins`, puis
2. On groupe les données par espèce, puis
3. On résume les données sous la forme de moyennes et d'écart-types

Et l'objet obtenu au final n'est pas strictement identique : avec la fonction `group_by()`, et dans certaines situations, la tibble obtenu conserve l'information du regroupement effectué, ce qui peut être utile dans certaines situations, mais pose parfois problème et cause l'affichage de messages d'avertissements dans la console. Ce comportement n'est pas observé avec l'argument `.by` qui ne groupe les données qu'au moment du calcul des indices dans la fonction `summarise()` et n'en conserve pas la trace ensuite. C'est la raison pour laquelle nous privilégierons cette méthode.

Pour aller plus loin, ajoutons à ce résumé 2 variables supplémentaires : le nombre de mesures et l'**erreur standard** (notée *se*), qui peut être calculée de la façon suivante :

$$se \approx \frac{s}{\sqrt{n}}$$

avec *s*, l'écart-type de l'échantillon et *n*, la taille de l'échantillon (plus d'informations sur cette statistique très importante dans la Chapitre 2). Nous allons donc calculer ici ces résumés, et nous donnerons un nom au tableau créé pour pouvoir ré-utiliser ces statistiques descriptives :

```
stats_esp <- penguins |>
  summarise(moyenne = mean(bill_length_mm, na.rm = TRUE),
            ecart_type = sd(bill_length_mm, na.rm = TRUE),
            nb_obs = n(),
```

```

    erreur_std = ecart_type / sqrt(nb_obs),
    .by = species)

stats_esp

```

```

# A tibble: 3 x 5
  species  moyenne ecart_type nb_obs erreur_std
  <fct>    <dbl>    <dbl> <int>    <dbl>
1 Adelie   38.8      2.66   152     0.216
2 Gentoo  47.5      3.08   124     0.277
3 Chinstrap 48.8      3.34    68     0.405

```

Vous constatez ici que nous avons 4 statistiques descriptives pour chaque espèce. Deux choses sont importantes à retenir ici :

1. on peut obtenir le nombre d'observations dans chaque sous-groupe d'un tableau groupé en utilisant la fonction `n()`. Cette fonction n'a besoin d'aucun argument : elle détermine automatiquement la taille des groupes créés par `.by` (ou par la fonction `group_by()`).
2. on peut créer de nouvelles variables en utilisant le nom de variables créées auparavant. Ainsi, nous avons créé la variable `erreur_std` en utilisant deux variables créées au préalable : `ecart_type` et `nb_obs`

1.2.3 Grouper par plus d'une variable

Jusqu'ici, nous avons groupé les données par espèce. Il est tout à fait possible de grouper les données par plus d'une variable, par exemple, par espèce et par sexe :

```

stats_esp_sex <- penguins |>
  summarise(moyenne = mean(bill_length_mm, na.rm = TRUE),
            ecart_type = sd(bill_length_mm, na.rm = TRUE),
            nb_obs = n(),
            erreur_std = ecart_type / sqrt(nb_obs),
            .by = c(species, sex))

stats_esp_sex

```

```
# A tibble: 8 x 6
  species sex      moyenne ecart_type nb_obs erreur_std
  <fct>   <fct>   <dbl>     <dbl> <int>     <dbl>
1 Adelie  male     40.4       2.28    73        0.267
2 Adelie  female   37.3       2.03    73        0.237
3 Adelie  <NA>     37.8       2.80     6         1.14
4 Gentoo  female   45.6       2.05    58        0.269
5 Gentoo  male     49.5       2.72    61        0.348
6 Gentoo  <NA>     45.6       1.37     5         0.615
7 Chinstrap female   46.6       3.11    34        0.533
8 Chinstrap male    51.1       1.56    34        0.268
```

En plus de la variable `species`, la tableau `stats_esp_sex` contient une variable `sex`. Les statistiques que nous avons calculées plus tôt sont maintenant disponibles pour chaque espèce et chaque sexe. D'ailleurs, puisque le sexe de certains individus est inconnu, nous avons également des lignes pour lesquelles le sexe affiché est `NA`. Pour les éliminer, il suffit de retirer les lignes du tableau pour lesquelles le sexe des individus est inconnu avant de recalculer les mêmes indices :

```
stats_esp_sex2 <- penguins |>
  filter(!is.na(sex)) |>
  summarise(moyenne = mean(bill_length_mm, na.rm = TRUE),
            ecart_type = sd(bill_length_mm, na.rm = TRUE),
            nb_obs = n(),
            erreur_std = ecart_type / sqrt(nb_obs),
            .by = c(species, sex))

stats_esp_sex2
```

```
# A tibble: 6 x 6
  species sex      moyenne ecart_type nb_obs erreur_std
  <fct>   <fct>   <dbl>     <dbl> <int>     <dbl>
1 Adelie  male     40.4       2.28    73        0.267
2 Adelie  female   37.3       2.03    73        0.237
3 Gentoo  female   45.6       2.05    58        0.269
4 Gentoo  male     49.5       2.72    61        0.348
5 Chinstrap female   46.6       3.11    34        0.533
6 Chinstrap male    51.1       1.56    34        0.268
```

Si vous ne comprenez pas la commande `filter(!is.na(sex))`, je vous encourage vivement à consulter [cette section du livre en ligne de Biométrie du semestre 3](#).

Enfin, lorsque nous groupons par plusieurs variables, il peut être utile de présenter les résultats sous la forme d'un tableau large (grâce à la fonction `pivot_wider()`) pour l'intégration dans un rapport par exemple. La fonction `pivot_wider()` permet de passer d'un tableau qui possède ce format :

```
penguins |>
  filter(!is.na(sex)) |>
  summarise(moyenne = mean(bill_length_mm, na.rm = TRUE),
            .by = c(species, sex))
```

```
# A tibble: 6 x 3
  species sex     moyenne
  <fct>   <fct>   <dbl>
1 Adelie male     40.4
2 Adelie female  37.3
3 Gentoo female  45.6
4 Gentoo male     49.5
5 Chinstrap female 46.6
6 Chinstrap male    51.1
```

à un tableau sous ce format :

```
penguins |>
  filter(!is.na(sex)) |>
  summarise(moyenne = mean(bill_length_mm, na.rm = TRUE),
            .by = c(species, sex)) |>
  pivot_wider(names_from = sex,
              values_from = moyenne)
```

```
# A tibble: 3 x 3
  species   male female
  <fct>   <dbl> <dbl>
1 Adelie  40.4  37.3
2 Gentoo  49.5  45.6
3 Chinstrap 51.1  46.6
```


Sous cette forme, les données ne sont plus “rangées”, c’est à dire que nous n’avons plus une observation par ligne et une variable par colonne. En effet ici, la variable `sex` est maintenant “étalée” dans 2 colonnes distinctes : chaque modalité du facteur de départ (`female` et `male`) est utilisée en tant que titre de nouvelles colonnes, et la variable `moyenne` est répartie dans deux colonnes. Ce format de tableau n’est pas idéal pour les statistiques ou les représentations graphiques, mais il est plus synthétique, et donc plus facile à inclure dans un rapport ou un compte-rendu.

1.2.4 Un raccourci pratique pour compter des effectifs

Il est extrêmement fréquent d’avoir à grouper des données en fonction d’une variable catégorielle puis d’avoir à compter le nombre d’observations de chaque modalité avec `n()` :

```
penguins |>
  summarise(effectif = n(),
            .by = species)
```

```
# A tibble: 3 x 2
  species    effectif
  <fct>      <int>
1 Adelie      152
2 Gentoo     124
3 Chinstrap   68
```

ou encore :

```
penguins |>
  group_by(species) |>
  summarise(effectif = n())
```

```
# A tibble: 3 x 2
  species    effectif
  <fct>      <int>
1 Adelie      152
2 Chinstrap   68
3 Gentoo     124
```

Ces deux opérations sont tellement fréquentes (regrouper puis compter) que le package `dplyr` nous fournit un raccourci : la fonction `count()`.

Le code ci-dessus est équivalent à celui-ci :

```
penguins |>
  count(species)

# A tibble: 3 x 2
  species      n
  <fct>    <int>
1 Adelie    152
2 Chinstrap  68
3 Gentoo   124
```

Notez qu'avec la fonction `count()`, la colonne qui contient les comptages s'appelle toujours `n` par défaut. Comme avec `.by` et `group_by()`, il est bien sûr possible d'utiliser `count()` avec plusieurs variables :

```
penguins |>
  count(species, sex)

# A tibble: 8 x 3
  species  sex      n
  <fct>   <fct> <int>
1 Adelie female   73
2 Adelie male    73
3 Adelie <NA>     6
4 Chinstrap female  34
5 Chinstrap male   34
6 Gentoo female  58
7 Gentoo male   61
8 Gentoo <NA>    5
```

```
penguins |>
  filter(!is.na(sex)) |>
  count(species, sex)
```

```
# A tibble: 6 x 3
  species sex      n
  <fct>   <fct> <int>
1 Adelie female   73
2 Adelie male    73
3 Chinstrap female  34
4 Chinstrap male   34
5 Gentoo female   58
6 Gentoo male    61
```

Et il est évidemment possible de présenter le résultats sous un format de tableau large :

```
penguins |>
  filter(!is.na(sex)) |>
  count(species, sex) |>
  pivot_wider(names_from = sex,
              values_from = n)
```

```
# A tibble: 3 x 3
  species female male
  <fct>     <int> <int>
1 Adelie       73   73
2 Chinstrap    34   34
3 Gentoo       58   61
```

Vous connaissez maintenant plusieurs méthodes pour calculer à la main des statistiques descriptives pour des variables entières, ou pour des sous-groupes de lignes (par espèce, par sexe, par sexe et par espèce...). Globalement, toutes les fonctions de R qui prennent une série de chiffres en guise d'argument, et qui renvoient une valeur unique, peuvent être utilisées avec la fonction `summarise()`. En particulier, vous pouvez utiliser les fonctions suivantes pour faire des analyses exploratoires :

- `mean()` : calcul de la moyenne
- `median()` : calcul de la médiane
- `min()` : affichage de la valeur minimale
- `max()` : affichage de la valeur maximale
- `n_distinct()` : calcul du nombre de valeurs différentes
- `n()` : calcul du nombre d'observations

- `var()` : calcul de la variance
- `sd()` : calcul de l'écart-type
- `IQR()` : calcul de l'intervalle inter-quartiles

Et la liste n'est bien sûr pas exhaustive

1.2.5 Exercices

1. Avec le tableau `diamonds` du package `ggplot2`, faites un tableau indiquant combien de diamants de chaque couleur on dispose. Vous devriez obtenir le tableau suivant :

```
# A tibble: 7 x 2
  color      n
  <ord> <int>
1 D      6775
2 E      9797
3 F      9542
4 G     11292
5 H      8304
6 I      5422
7 J      2808
```

2. Examinez le tableau `weather` du package `nycflights13` et lisez son fichier d'aide pour comprendre à quoi correspondent les données et comment elles ont été acquises.
3. À partir du tableau `weather` faites un tableau indiquant les vitesses de vents minimales, maximales et moyennes, enregistrées chaque mois dans chaque aéroport de New York. Indice : les 3 aéroports de New York sont Newark, LaGuardia Airport et John F. Kennedy, notés respectivement `EWR`, `LGA` et `JFK` dans la variable `origin`. Votre tableau devrait ressembler à ceci :

```
# A tibble: 36 x 5
  origin month max_wind min_wind moy_wind
  <chr>   <int>    <dbl>    <dbl>    <dbl>
1 EWR     1     42.6      0     9.87
2 EWR     2    1048.      0    12.2
3 EWR     3     29.9      0    11.6
```

```

4 EWR      4      25.3      0      9.63
5 EWR      5      33.4      0      8.49
6 EWR      6      34.5      0      9.55
7 EWR      7      20.7      0      9.15
8 EWR      8      21.9      0      7.62
9 EWR      9      23.0      0      8.03
10 EWR     10      26.5      0      8.32
# i 26 more rows

```

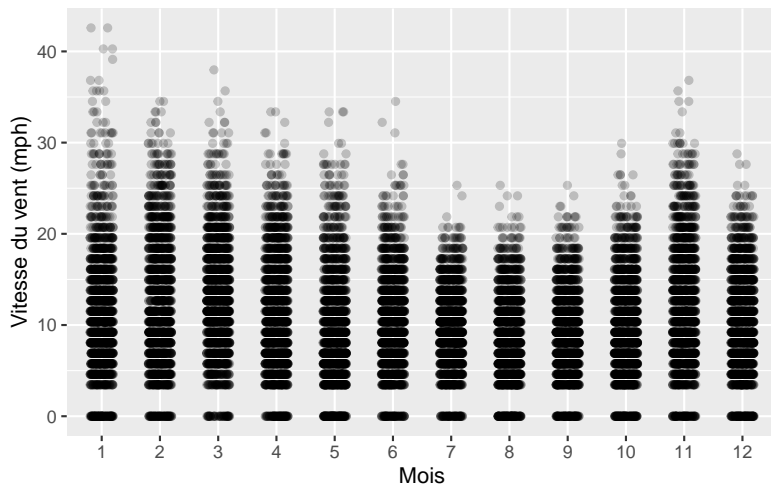
4. Sachant que les vitesses du vent sont exprimées en miles par heure, certaines valeurs sont-elles surprenantes ? À l'aide de la fonction `filter()`, éliminez la ou les valeurs aberrantes. Vous devriez obtenir ce tableau :

```

# A tibble: 36 x 5
  origin month max_wind min_wind moy_wind
  <chr>   <int>     <dbl>   <dbl>   <dbl>
1 EWR     1      42.6     0      9.87
2 EWR     2      31.1     0     10.7
3 EWR     3      29.9     0     11.6
4 EWR     4      25.3     0      9.63
5 EWR     5      33.4     0      8.49
6 EWR     6      34.5     0      9.55
7 EWR     7      20.7     0      9.15
8 EWR     8      21.9     0      7.62
9 EWR     9      23.0     0      8.03
10 EWR    10      26.5     0      8.32
# i 26 more rows

```

5. En utilisant les données de vitesse de vent du tableau `weather`, produisez le graphique suivant :



Indications :

- les vitesses de vent aberrantes ont été éliminées grâce à la fonction `filter()`
- la fonction `geom_jitter()` a été utilisée avec l'argument `height = 0`
- la transparence des points est fixée à 0.2

6. À votre avis :

- pourquoi les points sont-ils organisés en bandes horizontales ?
- pourquoi n'y a-t-il jamais de vent entre 0 et environ 3 miles à l'heure (mph) ?
- Sachant qu'en divisant des mph par 1.151 on obtient des vitesses en nœuds, que nous apprend cette commande :

```
sort(unique(weather$wind_speed)) / 1.151
```

```
[1] 0.000000  2.999427  3.999235  4.999044  5.998853  6.998662
[7] 7.998471  8.998280  9.998089 10.997897 11.997706 12.997515
[13] 13.997324 14.997133 15.996942 16.996751 17.996560 18.996368
[19] 19.996177 20.995986 21.995795 22.995604 23.995413 24.995222
[25] 25.995030 26.994839 27.994648 28.994457 29.994266 30.994075
[31] 31.993884 32.993692 33.993501 34.993310 36.992928 910.825873
```

1.3 Les fonctions `pivot_wider()` et `pivot_longer()`

1.3.1 Du format long au format large : `pivot_wider()`

Comme nous venons de le voir, la fonction `pivot_wider()` permet de passer d'un tableau au format long à un tableau au format large, qui contient donc moins de lignes mais plus de colonnes.

Par exemple, lorsque l'on dispose d'un tableau contenant un résumé de données pour plusieurs catégories et sous-catégories, il peut être utile de le transformer au format large pour l'intégrer dans un rapport (la présentation en est ainsi plus synthétique) ou pour faire certains types de graphiques. Par exemple, avec ce tableau de résumé de données :

```
resum <- penguins |>
  filter(!is.na(sex)) |>
  count(species, sex)

resum
```

```
# A tibble: 6 x 3
  species  sex      n
  <fct>    <fct> <int>
1 Adelie  female  73
2 Adelie  male    73
3 Chinstrap female  34
4 Chinstrap male    34
5 Gentoo  female  58
6 Gentoo  male    61
```

La présentation au format large serait plus appropriée dans un compte-rendu de TP :

```
resum_large1 <- resum |>
  pivot_wider(names_from = sex,
              values_from = n)

resum_large1
```

```
# A tibble: 3 x 3
  species  female  male
  <fct>    <int> <int>
1 Adelie      73    73
2 Chinstrap   34    34
3 Gentoo     58    61
```

L'argument `names_from` permet d'indiquer dans quelle colonne du tableau de départ aller chercher les noms de colonnes pour le nouveau tableau. Ici, on va chercher dans la colonne `sex` les noms de colonne du futur tableau (`female` et `male`). L'argument `values_from` permet d'indiquer dans quelle colonne du tableau de départ on souhaite aller chercher les valeurs que l'on souhaite mettre dans les nouvelles colonnes du futur tableau (ici, les effectifs stockés dans la colonne `n`).

Le terme “tableau large” peut-être trompeur car le tableau obtenu n'a pas forcément plus de colonnes que le tableau d'origine. En tous cas, il a toujours moins de lignes que le tableau de départ. Ici, `resum` avait 6 lignes et 3 colonnes, `resum_large1` possède 3 lignes et 3 colonnes.

Si on souhaite avoir les espèces en colonnes et les sexes en lignes, on peut taper ceci :

```
resum_large2 <- resum |>
  pivot_wider(names_from = species,
              values_from = n)

resum_large2
```

```
# A tibble: 2 x 4
  sex    Adelie Chinstrap Gentoo
  <fct>  <int>    <int>  <int>
1 female     73        34    58
2 male      73        34    61
```

Cette fois, `resum_large2` possède 2 lignes et 4 colonnes. Le caractère “large” de ce nouveau tableau est ici bien apparent. Notez bien que ce sont toujours les mêmes données qui figurent dans ces 3 objets : seule la présentation change.

1.3.2 Du format large au format long : `pivot_longer()`

À l'inverse, on dispose parfois de données au format large alors que la plupart des fonctions graphiques et statistiques de R requièrent des données au format long, c'est à dire, des tableaux dans lesquels il y a une correspondance stricte entre colonnes et variables : une variable est contenue dans une seule colonne d'un tableau, et chaque ligne correspond à une unique observation. Ainsi, le tableau `resum_large2` n'est pas au format long :

```
resum_large2

# A tibble: 2 x 4
  sex    Adelie Chinstrap Gentoo
<fct> <int>    <int> <int>
1 female     73      34    58
2 male       73      34    61
```

En effet, 3 de ses colonnes contiennent des données d'abondances qui pourraient (ou devraient) se trouver dans une unique colonne `Abondance`, et le titre de ces 3 colonnes devrait être les catégories d'une autre variable `Espece`. Pour transformer un tableau large en tableau long (qui contient donc toujours plus de lignes, et souvent moins de colonnes), on utilise `pivot_longer()`. Cette fonction possède 3 arguments :

- `cols` : quelles sont les colonnes que l'on souhaite regrouper
- `names_to` : comment s'appellera la variable qui contiendra les anciens noms de colonnes du tableau large
- `values_to` : comment s'appellera la variable qui contiendra les données contenues dans les cellules du tableau large

Voilà un exemple :

```
resum_long <- resum_large2 |>
  pivot_longer(cols = c(Adelie, Chinstrap, Gentoo),
               names_to = "Espece",
               values_to = "Abondance")
```

```
resum_long
```

```
# A tibble: 6 x 3
  sex     Espece  Abondance
<fct> <chr>      <int>
1 female Adelie      73
2 female Chinstrap  34
3 female Gentoo    58
4 male   Adelie      73
5 male   Chinstrap  34
6 male   Gentoo    61
```

Les 3 colonnes qui contenaient les abondance des 3 espèces de manchots ont été regroupées en 2 nouvelles colonnes dont les noms ont été précisés grâce à `names_to` et `values_to`.

Il est très fréquent d'avoir à passer d'un format large à un format long ou inversement. Il est donc important que vous appreniez à vous familiariser avec ces 2 fonctions.

! les tableaux rangés

La plupart des fonctions du `tidyverse` supposent que les tableaux soient rangés (une colonne par variable, une ligne par observation) et qu'ils aient donc un format long. À l'inverse, pour présenter des tableaux dans un rapport ou pour certaines méthodes particulières, les données peuvent être présentées au format large. Les fonctions `pivot_wider()` et `pivot_longer()` sont complémentaires et permettent de passer d'un format à l'autre.

1.4 Créer des résumés avec la fonction `reframe()`

Comme nous venons de le voir, les calculs que nous pouvons faire grâce à la fonction `summarise()` impliquent des fonctions statistiques qui ne renvoient qu'une valeur à la fois lorsqu'on leur fournit une série de valeurs. Par exemple, si

on dispose d'un vecteur numérique (les entiers compris entre 1 et 100 pour l'exemple) :

```
1:100
```

```
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
[19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
[37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
[55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
[73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
[91] 91 92 93 94 95 96 97 98 99 100
```

la fonction `mean()` ne renvoie qu'une valeur, la moyenne des 100 valeurs contenues dans le vecteur :

```
mean(1:100)
```

```
[1] 50.5
```

De même pour les fonctions `sd()`, ou `median()`, ou toutes les autres fonctions listées à la fin de la Section 1.2.4 :

```
sd(1:100)
```

```
[1] 29.01149
```

```
median(1:100)
```

```
[1] 50.5
```

Il existe toutefois des fonctions qui renvoient plus d'une valeur à la fois. Par exemple, la fonction `quantile()`, renvoie par défaut 5 éléments :

1. la valeur minimale contenue dans le vecteur (ou quantile 0%) : c'est la valeur la plus faible contenue dans la série de données

2. le premier quartile du vecteur (Q1 ou quantile 25%) est la valeur coupant l'échantillon en deux : 25% des observations du vecteur y sont inférieures et 75% y sont supérieures
3. la médiane du vecteur (Q2 ou quantile 50%) est la valeur coupant l'échantillon en deux : 50% des observations du vecteur sont inférieures à cette valeur et 50% y sont supérieures
4. le troisième quartile du vecteur (Q3 ou quantile 75%) est la valeur coupant l'échantillon en deux : 75% des observations du vecteur y sont inférieures et 25% y sont supérieures
5. la valeur maximale contenue dans le vecteur (ou quantile 100%) : c'est la valeur la plus élevée contenue dans la série de données.

Par exemple, toujours avec le vecteur des entiers contenus entre 1 et 100 :

```
quantile(1:100)
```

```

      0%    25%    50%    75%   100%
1.00  25.75  50.50  75.25 100.00

```

L'objet obtenu est un vecteur dont chaque élément porte un nom. Pour transformer cet objet en tibble, on utilise la fonction `enframe()` :

```
enframe(quantile(1:100))
```

```

# A tibble: 5 x 2
  name  value
<chr> <dbl>
1 0%      1
2 25%    25.8
3 50%    50.5
4 75%    75.2
5 100%  100

```

Il peut être très utile de calculer ces différentes valeurs pour plusieurs variables à la fois, ou pour plusieurs sous-groupes

d'un jeu de données. Le problème est que nous ne pouvons pas utiliser `summarise()` car la fonction `quantile()` ne renvoie pas qu'une unique valeur. Par exemple, pour calculer les quantiles des longueurs de becs pour chaque espèce de manchots, on pourrait être tenté de taper ceci :

```
penguins |>
  summarise(Indices = quantile(bill_length_mm, na.rm = TRUE),
            .by = species)
```

Warning: Returning more (or less) than 1 row per `summarise()` group was deprecated in dplyr 1.1.0.

i Please use `reframe()` instead.

i When switching from `summarise()` to `reframe()`, remember that `reframe()` always returns an ungrouped data frame and adjust accordingly.

```
# A tibble: 15 x 2
  species    Indices
  <fct>      <dbl>
1 Adelie     32.1
2 Adelie     36.8
3 Adelie     38.8
4 Adelie     40.8
5 Adelie     46
6 Gentoo     40.9
7 Gentoo     45.3
8 Gentoo     47.3
9 Gentoo     49.6
10 Gentoo    59.6
11 Chinstrap 40.9
12 Chinstrap 46.3
13 Chinstrap 49.6
14 Chinstrap 51.1
15 Chinstrap 58
```

C'est dans ces situations que la fonction `reframe()` est utile. Elle joue le même rôle que `summarise()`, mais dans les situations où les fonctions statistiques renvoient plus d'une valeur à la fois :

```

penguins |>
  reframe(Indices = quantile(bill_length_mm, na.rm = TRUE),
          .by = species)

# A tibble: 15 x 2
  species    Indices
  <fct>      <dbl>
1 Adelie     32.1
2 Adelie     36.8
3 Adelie     38.8
4 Adelie     40.8
5 Adelie      46
6 Gentoo     40.9
7 Gentoo     45.3
8 Gentoo     47.3
9 Gentoo     49.6
10 Gentoo    59.6
11 Chinstrap 40.9
12 Chinstrap 46.3
13 Chinstrap 49.6
14 Chinstrap 51.1
15 Chinstrap 58

```

Au contraire de `summarise()`, `reframe()` ne renvoie pas de message d'avertissement dans cette situation. Dans cet exemple, on ne sait malheureusement pas à quoi correspondent les chiffres renvoyés puisque l'information des quartiles a disparu (quelles valeurs correspondent aux médianes ou aux premiers quartiles par exemple). Pour y remédier, on doit transformer le vecteur renvoyé par `quantile()` en tibble. Nous avons déjà vu comment le faire grâce à la fonction `enframe()`. Par ailleurs, puisque la fonction va maintenant renvoyer un tableau, on n'a pas besoin de lui fournir de nom de colonnes (je retire donc `Indices =` de mon code) :

```

penguins |>
  reframe(enframe(quantile(bill_length_mm, na.rm = TRUE)),
          .by = species)

# A tibble: 15 x 3

```

| | species | name | value |
|----|-----------|-------|-------|
| | <fct> | <chr> | <dbl> |
| 1 | Adelie | 0% | 32.1 |
| 2 | Adelie | 25% | 36.8 |
| 3 | Adelie | 50% | 38.8 |
| 4 | Adelie | 75% | 40.8 |
| 5 | Adelie | 100% | 46 |
| 6 | Gentoo | 0% | 40.9 |
| 7 | Gentoo | 25% | 45.3 |
| 8 | Gentoo | 50% | 47.3 |
| 9 | Gentoo | 75% | 49.6 |
| 10 | Gentoo | 100% | 59.6 |
| 11 | Chinstrap | 0% | 40.9 |
| 12 | Chinstrap | 25% | 46.3 |
| 13 | Chinstrap | 50% | 49.6 |
| 14 | Chinstrap | 75% | 51.1 |
| 15 | Chinstrap | 100% | 58 |

Enfin, comme précédemment, il est possible de modifier la forme de ce tableau (avec `pivot_wider()`) pour le lire plus facilement et éventuellement l'intégrer dans un rapport ou compte-rendu :

```
penguins |>
  reframe(enframe(quantile(bill_length_mm, na.rm = TRUE)),
          .by = species) |>
  pivot_wider(names_from = species,
              values_from = value)
```

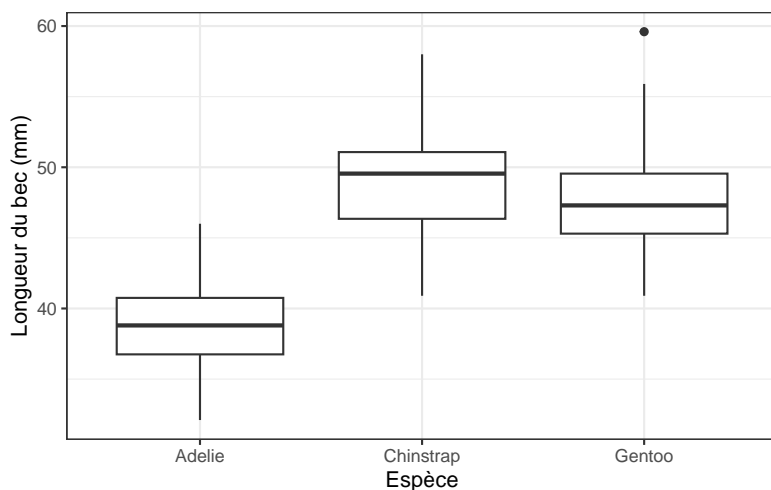
```
# A tibble: 5 x 4
  name Adelie Gentoo Chinstrap
  <chr> <dbl> <dbl> <dbl>
1 0%    32.1  40.9  40.9
2 25%    36.8  45.3  46.3
3 50%    38.8  47.3  49.6
4 75%    40.8  49.6  51.1
5 100%   46    59.6  58
```

Ces statistiques nous permettent de constater que les manchots de l'espèce Adélie semblent avoir des becs plus courts que les 2 autres espèces (les 5 quantiles le confirment). Les manchots Gentoo et Chinstrap ont en revanche des becs de

longueur à peu près similaires, bien que ceux des Chinstrap soient peut-être très légèrement plus longs (Q1, médiane et Q3 supérieurs à ceux des Gentoo). On peut vérifier tout ça graphiquement avec des boîtes à moustaches :

```
penguins |>
  ggplot(aes(x = species, y = bill_length_mm)) +
  geom_boxplot() +
  labs(x = "Espèce", y = "Longueur du bec (mm)") +
  theme_bw()
```

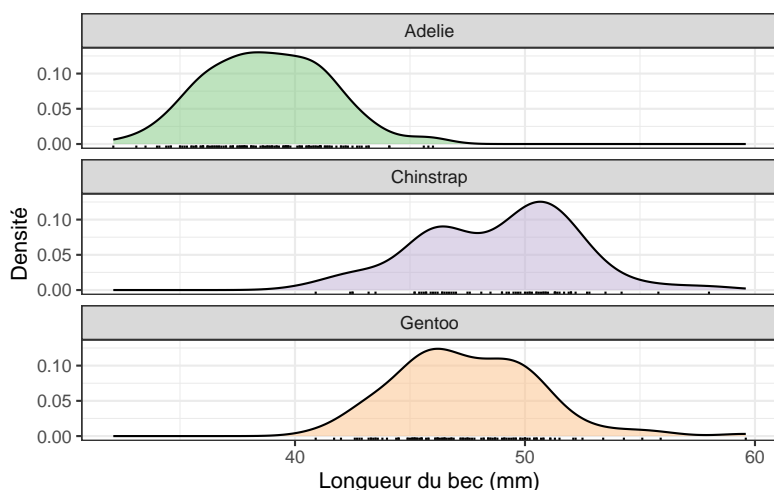
Warning: Removed 2 rows containing non-finite outside the scale range (`stat_boxplot()`).



Ou avec un graphique de densité :

```
penguins |>
  ggplot(aes(x = bill_length_mm, fill = species)) +
  geom_density(alpha = 0.5, show.legend = FALSE) +
  geom_rug() +
  labs(x = "Longueur du bec (mm)", y = "Densité") +
  facet_wrap(~species, ncol = 1) +
  scale_fill_brewer(palette = "Accent") +
  theme_bw()
```

Warning: Removed 2 rows containing non-finite outside the scale range (`stat_density()`).



À ce stade, vous devriez être capables de créer (et d'interpréter !) ce type de graphiques. Si ce n'est pas le cas, consultez d'urgence [le chapitre 3 du livre en ligne de Biométrie du semestre 3](#).

💡 À retenir

- la fonction `summarise()` s'utilise avec des fonctions statistiques qui ne renvoient qu'une valeur (par exemple `mean()`, `median()`, `sd()`, `var()`...)
- la fonction `reframe()` s'utilise avec des fonctions statistiques qui renvoient plusieurs valeurs (par exemple `quantile()`, `range()`...)

1.5 Créer des résumés de données avec des fonctions spécifiques

Les fonctions `summarise()` et `reframe()`, avec leur argument `.by()` (ou la fonction `group_by()`) permettent donc de calculer n'importe quel indice de statistique descriptive sur un tableau de données entier ou sur des modalités ou combinaisons de modalités de facteurs. Il existe par ailleurs de nombreuses fonctions, disponibles de base dans R ou dans certains packages spécifiques, qui permettent de fournir des résumés plus ou moins automatiques de tout ou partie des variables d'un jeu de données. Nous allons en décrire 2 ici, mais il en

existe beaucoup d'autres : à vous d'explorer les possibilités et d'utiliser les fonctions qui vous paraissent les plus pertinentes, les plus simples à utiliser, les plus visuelles ou les plus complètes.

1.5.1 La fonction `summary()`

La fonction `summary()` (à ne pas confondre avec `summarise()`) permet d'obtenir des résumés de données pour tous types d'objets dans R. Selon la classe des objets que l'on transmet à `summary()`, la nature des résultats obtenus changera. Nous verrons ainsi au semestre 6 que cette fonction peut être utilisée pour examiner les résultats de modèles de régressions linéaires ou d'analyses de variances. Pour l'instant, nous nous intéressons à 3 situations :

- ce que renvoie la fonction quand on lui fournit un vecteur
- ce que renvoie la fonction quand on lui fournit un facteur
- ce que renvoie la fonction quand on lui fournit un tableau

1.5.1.1 Variable continue : vecteur numérique

Commençons par fournir un vecteur numérique à la fonction `summary()`. Nous allons pour cela extraire les données de masses corporelles des manchots du tableau `penguins` :

```
penguins$body_mass_g  
  
 [1] 3750 3800 3250  NA 3450 3650 3625 4675 3475 4250 3300 3700 3200 3800 4400  
[16] 3700 3450 4500 3325 4200 3400 3600 3800 3950 3800 3800 3550 3200 3150 3950  
[31] 3250 3900 3300 3900 3325 4150 3950 3550 3300 4650 3150 3900 3100 4400 3000  
[46] 4600 3425 2975 3450 4150 3500 4300 3450 4050 2900 3700 3550 3800 2850 3750  
[61] 3150 4400 3600 4050 2850 3950 3350 4100 3050 4450 3600 3900 3550 4150 3700  
[76] 4250 3700 3900 3550 4000 3200 4700 3800 4200 3350 3550 3800 3500 3950 3600  
[91] 3550 4300 3400 4450 3300 4300 3700 4350 2900 4100 3725 4725 3075 4250 2925  
[106] 3550 3750 3900 3175 4775 3825 4600 3200 4275 3900 4075 2900 3775 3350 3325  
[121] 3150 3500 3450 3875 3050 4000 3275 4300 3050 4000 3325 3500 3500 4475 3425  
[136] 3900 3175 3975 3400 4250 3400 3475 3050 3725 3000 3650 4250 3475 3450 3750
```

```

[151] 3700 4000 4500 5700 4450 5700 5400 4550 4800 5200 4400 5150 4650 5550 4650
[166] 5850 4200 5850 4150 6300 4800 5350 5700 5000 4400 5050 5000 5100 4100 5650
[181] 4600 5550 5250 4700 5050 6050 5150 5400 4950 5250 4350 5350 3950 5700 4300
[196] 4750 5550 4900 4200 5400 5100 5300 4850 5300 4400 5000 4900 5050 4300 5000
[211] 4450 5550 4200 5300 4400 5650 4700 5700 4650 5800 4700 5550 4750 5000 5100
[226] 5200 4700 5800 4600 6000 4750 5950 4625 5450 4725 5350 4750 5600 4600 5300
[241] 4875 5550 4950 5400 4750 5650 4850 5200 4925 4875 4625 5250 4850 5600 4975
[256] 5500 4725 5500 4700 5500 4575 5500 5000 5950 4650 5500 4375 5850 4875 6000
[271] 4925 NA 4850 5750 5200 5400 3500 3900 3650 3525 3725 3950 3250 3750 4150
[286] 3700 3800 3775 3700 4050 3575 4050 3300 3700 3450 4400 3600 3400 2900 3800
[301] 3300 4150 3400 3800 3700 4550 3200 4300 3350 4100 3600 3900 3850 4800 2700
[316] 4500 3950 3650 3550 3500 3675 4450 3400 4300 3250 3675 3325 3950 3600 4050
[331] 3350 3450 3250 4050 3800 3525 3950 3650 3650 4000 3400 3775 4100 3775

```

Nous avons donc 344 valeurs de masses en grammes qui correspondent aux 344 manchots du jeu de données. La fonction `summary()` renvoie le résumé suivant lorsqu'on lui fournit ces valeurs :

```
summary(penguins$body_mass_g)
```

```

  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.   NA's
 2700   3550    4050   4202   4750   6300     2

```

Nous obtenons ici 7 valeurs, qui correspondent respectivement à :

- la valeur minimale observée dans le vecteur. Ici, le manchot le plus léger de l'échantillon pèse donc 2700 grammes.
- la valeur du premier quartile du vecteur. Le premier quartile est la valeur qui coupe l'échantillon en 2 groupes : 25% des observations du vecteur sont inférieures au premier quartile, et 75% des observations du vecteur sont supérieures au premier quartile. Ici, 25% des manchots de l'échantillon (soit 86 individus) ont une masse inférieure à 3550 grammes, et 75% des individus de l'échantillon (soit 258 individus) ont une masse supérieure à 3550 grammes.
- la valeur de médiane du vecteur. La médiane est la valeur qui coupe l'échantillon en 2 groupes : 50% des observations du vecteur sont inférieures à la médiane, et

50% des observations du vecteur sont supérieures à la médiane. Ici, 50% des manchots de l'échantillon (soit 172 individus) ont une masse inférieure à 4050 grammes, et 50% des individus de l'échantillon (soit 172 individus) ont une masse supérieure à 4050 grammes.

- la moyenne du vecteur. Ici, les manchots des 3 espèces du jeu de données ont en moyenne une masse 4202 grammes.
- la valeur du troisième quartile du vecteur. Le troisième quartile est la valeur qui coupe l'échantillon en 2 groupes : 75% des observations du vecteur sont inférieures au troisième quartile, et 25% des observations du vecteur sont supérieures au troisième quartile. Ici, 75% des manchots de l'échantillon (soit 258 individus) ont une masse inférieure à 4700 grammes, et 25% des individus de l'échantillon (soit 86 individus) ont une masse supérieure à 4750 grammes.
- la valeur maximale observée dans le vecteur. Ici, le manchot le plus lourd de l'échantillon pèse donc 6300 grammes.
- le nombre de données manquantes. Ici, 2 manchots n'ont pas été pesés et présentent donc la mention NA (comme **N**ot **A**vailable) pour la variable `body_mass_g`.

Cette fonction fournit donc 2 indices de plus que la fonction `quantile()` (la moyenne et le nombre de valeurs manquantes) :

```
quantile(penguins$body_mass_g, na.rm = TRUE)
```

```
0% 25% 50% 75% 100%  
2700 3550 4050 4750 6300
```

Mais contrairement à ce que nous avons vu plus haut, la fonction `summary()` ne possède pas d'argument `.by()` et il n'est pas possible de l'utiliser avec la fonction `group_by()`. Par conséquent, il n'est pas possible de se servir de cette fonction pour avoir des valeurs pour chaque modalités d'un facteur (pour chaque espèce par exemple).

Les différents indices statistiques fournis nous renseignent à la fois sur la position de la distribution et sur la dispersion des données.

- La *position* correspond à la tendance centrale et indique quelles sont les valeurs qui caractérisent le plus grand nombre d'individus. La moyenne et la médiane sont les deux indices de position les plus fréquemment utilisés. Lorsqu'une variable a une distribution parfaitement symétrique, la moyenne et la médiane sont strictement égales. Mais lorsqu'une distribution est asymétrique, la moyenne et la médiane diffèrent. En particulier, la moyenne est beaucoup plus sensible aux valeurs extrêmes que la médiane. Cela signifie que quand une distribution est très asymétrique, la médiane est souvent une meilleure indication des valeurs les plus fréquemment observées.

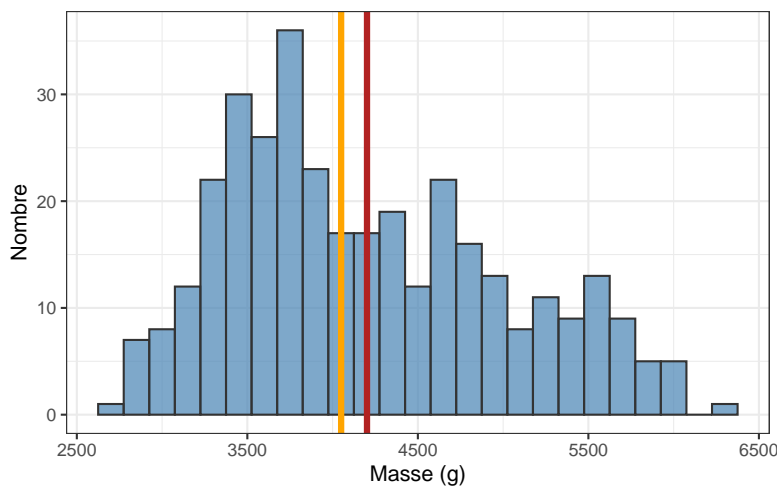


Figure 1.3: Distribution des masses corporelles des manchots

L'histogramme de la Figure 1.3 montre la distribution de la taille des manchots (toutes espèces confondues). Cette distribution présente une asymétrie à droite. Cela signifie que la distribution n'est pas symétrique et que la "queue de distribution" est plus longue à droite qu'à gauche. La plupart des individus ont une masse comprise entre 3500 et 3700 grammes, au niveau du pic principal du graphique. La médiane, en orange et qui vaut 4050 grammes est plus proche du pic que la moyenne, en rouge, qui vaut 4202 grammes. Ici, la différence entre moyenne et médiane n'est pas énorme, mais elle peut le devenir si la distribution est vraiment très asymétrique, par exemple, si quelques individus seulement avaient une masse supérieure à 7000 grammes, la moyenne serait tirée vers la droite du graphique alors que la médiane ne serait

presque pas affectée. La moyenne représenterait alors encore moins fidèlement la tendance centrale.

Si l'on revient à la fonction `summary()`, observer des valeurs proches pour la moyenne et la médiane nous indique donc le degré de symétrie de la distribution.

- La *dispersion* des données nous renseigne sur la dispersion des points autour des indices de position. Les quartiles et les valeurs minimales et maximales renvoyées par la fonction `summary()` nous renseignent sur l'étalement des points. Les valeurs situées entre le premier et le troisième quartile correspondent aux 50% des valeurs de l'échantillon les plus centrales. Plus l'étendue entre ces quartiles (notée IQR pour "intervalle interquartile") sera grande, plus la dispersion sera importante. D'ailleurs, lorsque la dispersion est très importante, les moyennes et médianes ne renseignent que très moyennement sur la tendance centrale. Les indices de position sont surtout pertinents lorsque la dispersion des points autour de cette tendance centrale n'est pas trop large. Par exemple, si la distribution des données ressemblait à ceci (Figure 1.4), la moyenne et la médiane seraient fort peu utiles car très éloignées de la plupart des observations :

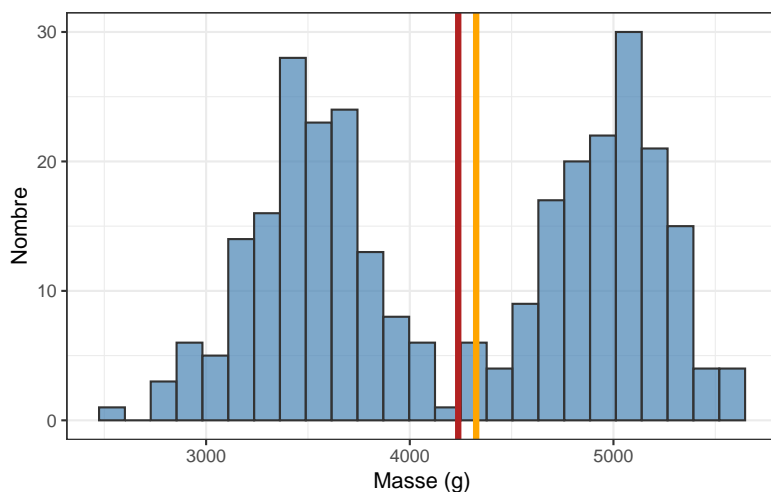


Figure 1.4: Distribution des masses corporelles (données fictives)

On comprend donc l'importance de considérer les indices de

dispersion en plus des indices de position pour caractériser et comprendre une série de données numériques. L'intervalle interquartile est toujours utile pour connaître l'étendue des données qui correspondent aux 50% des observations les plus centrales. Les autres indices de dispersion très fréquemment utilisés, mais qui ne sont pas proposés par défaut par la fonction `summary()`, sont la variance et l'écart-type. Il est possible de calculer tous les indices renvoyés par la fonction `summary()` et ceux qui nous manquent grâce à la fonction `summarise()` :

```
penguins |>
  summarise(min = min(body_mass_g, na.rm = TRUE),
            Q1 = quantile(body_mass_g, 0.25, na.rm = TRUE),
            med = median(body_mass_g, na.rm = TRUE),
            moy = mean(body_mass_g, na.rm = TRUE),
            Q3 = quantile(body_mass_g, 0.75, na.rm = TRUE),
            max = max(body_mass_g, na.rm = TRUE),
            var = var(body_mass_g, na.rm = TRUE),
            et = sd(body_mass_g, na.rm = TRUE))
```

```
# A tibble: 1 x 8
  min    Q1  med  moy   Q3  max   var   et
<int> <dbl> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl>
1  2700  3550  4050  4202.  4750  6300 643131.  802.
```

Vous notez que le code est beaucoup plus long, et qu'utiliser `summary()` peut donc faire gagner beaucoup de temps, même si cette fonction ne nous fournit ni la variance ni l'écart-type. Mais comme souvent dans R, il est possible de calculer à la main toutes ces valeurs si besoin. Comme indiqué plus haut, les fonctions suivantes peuvent être utilisées :

- `mean()` permet de calculer la moyenne.
- `median()` permet de calculer la médiane.
- `min()` et `max()` permettent de calculer les valeurs minimales et maximales respectivement.
- `quantile()` permet de calculer les quartiles. Vous notez que contrairement aux exemples de la partie précédente, on utilise ici la fonction `quantile()` en précisant une valeur supplémentaire pour n'obtenir qu'une valeur à la fois : 0.25 pour le premier quartile, et 0.75 pour le troisième.

- `sd()` permet de calculer l'écart-type.
- `var()` permet de calculer la variance.

Pour toutes ces fonctions l'argument `na.rm = TRUE` permet d'obtenir les résultats même si certaines valeurs sont manquantes. Enfin, la fonction `IQR()` permet de calculer l'intervalle inter-quartiles :

```
IQR(penguins$body_mass_g, na.rm = TRUE)
```

```
[1] 1200
```

Ici, les 50% des valeurs les plus centrales de l'échantillon sont situées dans un intervalle de 1200 grammes autour de la médiane.

1.5.1.2 Variable quantitative : facteur

Si l'on fournit une variable catégorielle ou facteur à `summary()`, le résultat obtenu sera naturellement différent : calculer des moyennes, médianes ou quartiles n'aurait en effet pas de sens lorsque la variable fournie ne contient que des catégories :

```
summary(penguins$species)
```

```
Adelie Chinstrap   Gentoo
   152      68      124
```

Pour les facteurs, `summary()` compte simplement le nombre d'observations pour chaque modalité. Ici, la variable `species` est un facteur qui compte 3 modalités. La fonction `summary()` nous indique donc le nombre d'individus pour chaque modalité : notre jeu de données se compose de 152 individus de l'espèce Adélie, 68 individus de l'espèce Chinstrap, et 124 individus de l'espèce Gentoo.

Comme pour les vecteurs numériques, si le facteur présente des données manquantes, la fonction `summary()` compte également leur nombre :


```
summary(penguins$sex)
```

```
female  male  NA's  
   165   168   11
```

Pour les facteurs, la fonction `summary()` est donc tout à fait équivalente à la fonction `count()` :

```
penguins |>  
  count(species)
```

```
# A tibble: 3 x 2  
  species      n  
  <fct>    <int>  
1 Adelie    152  
2 Chinstrap  68  
3 Gentoo   124
```

L'avantage de la fonction `count()` est qu'il est possible d'utiliser plusieurs facteurs pour compter le nombre d'observations de toutes les combinaisons de modalités (par exemple, combien d'individus de chaque sexe pour chaque espèce), ce qui n'est pas possible avec la fonction `summary()`.

1.5.1.3 Les tableaux : data.frame ou tibble

L'avantage de la fonction `summary()` par rapport à la fonction `count()` apparaît lorsque l'on souhaite obtenir des informations sur toutes les variables d'un tableau à la fois :

```
summary(penguins)
```

```
      species      island  bill_length_mm  bill_depth_mm  
Adelie  :152  Biscoe    :168  Min.      :32.10  Min.      :13.10  
Chinstrap: 68  Dream     :124  1st Qu.  :39.23  1st Qu.  :15.60  
Gentoo  :124  Torgersen: 52  Median   :44.45  Median   :17.30  
                          Mean      :43.92  Mean     :17.15  
                          3rd Qu.  :48.50  3rd Qu.  :18.70
```

```

Max.      :59.60   Max.      :21.50
NA's      :2       NA's      :2
flipper_length_mm  body_mass_g      sex      year
Min.      :172.0   Min.      :2700   female:165  Min.      :2007
1st Qu.   :190.0   1st Qu.   :3550   male  :168   1st Qu.   :2007
Median    :197.0   Median    :4050   NA's   : 11   Median    :2008
Mean      :200.9   Mean      :4202                   Mean      :2008
3rd Qu.   :213.0   3rd Qu.   :4750                   3rd Qu.   :2009
Max.      :231.0   Max.      :6300                   Max.      :2009
NA's      :2       NA's      :2

```

Ici, on obtient un résumé pour chaque colonne du tableau. Les colonnes numériques sont traitées comme les vecteurs numériques (on obtient alors les minimas et maximas, les quartiles, les moyennes et médianes) et les colonnes contenant des variables catégorielles sont traitées comme des facteurs (et on obtient alors le nombre d'observations pour chaque modalité).

On constate au passage que la variable `year` est considérée ici comme une variable numérique, alors qu'elle devrait plutôt être considérée comme un facteur, ce qui nous permettrait de savoir combien d'individus ont été échantillonnés chaque année :

```

penguins |>
  mutate(year = factor(year)) |>
  summary()

```

```

species      island  bill_length_mm  bill_depth_mm
Adelie      :152   Biscoe      :168   Min.      :32.10   Min.      :13.10
Chinstrap   :68    Dream       :124   1st Qu.   :39.23   1st Qu.   :15.60
Gentoo      :124   Torgersen   :52   Median    :44.45   Median    :17.30
Mean        :43.92   Mean        :17.15
3rd Qu.     :48.50   3rd Qu.     :18.70
Max.        :59.60   Max.        :21.50
NA's        :2       NA's        :2
flipper_length_mm  body_mass_g      sex      year
Min.      :172.0   Min.      :2700   female:165  2007:110
1st Qu.   :190.0   1st Qu.   :3550   male  :168   2008:114
Median    :197.0   Median    :4050   NA's   : 11   2009:120
Mean      :200.9   Mean      :4202
3rd Qu.   :213.0   3rd Qu.   :4750

```

```
Max.      :231.0      Max.      :6300
NA's     :2          NA's     :2
```

Au final, la fonction `summary()` est très utile dans certaines situations, notamment pour avoir rapidement accès à des statistiques descriptives simples sur toutes les colonnes d'un tableau. Elle reste cependant limitée car d'une part, elle ne fournit pas les variances ou les écarts-types pour les variables numériques, et il est impossible d'avoir des résumés plus fins, pour chaque modalité d'un facteur par exemple. Ici, il serait en effet intéressant d'avoir des informations synthétiques concernant les mesures biométriques des manchots, espèce par espèce, plutôt que toutes espèces confondues. C'est là que la fonction `skim()` intervient.

1.5.2 La fonction `skim()`

La fonction `skim()` fait partie du package `skimr`. Avant de pouvoir l'utiliser, pensez donc à l'installer et à le charger en mémoire si ce n'est pas déjà fait. Comme pour la fonction `summary()`, on peut utiliser la fonction `skim()` sur plusieurs types d'objets. Nous nous contenterons d'examiner ici le cas le plus fréquent : celui des tableaux, groupés avec `group_by()` ou non.

1.5.2.1 Tableau non groupé

Commençons par examiner le résultat avec le tableau `penguins` non groupé :

```
skim(penguins)
```

Table 1.1: Data summary

| | |
|------------------------|----------|
| Name | penguins |
| Number of rows | 344 |
| Number of columns | 8 |
| Column type frequency: | |
| factor | 3 |
| numeric | 5 |

| | |
|-----------------|------|
| Group variables | None |
|-----------------|------|

Variable type: factor

| skim_variable | n_missing | complete_rate | ordered | n_unique | top_counts |
|---------------|-----------|---------------|---------|----------|-----------------------------|
| species | 0 | 1.00 | FALSE | 3 | Ade: 152, Gen: 124, Chi: 68 |
| island | 0 | 1.00 | FALSE | 3 | Bis: 168, Dre: 124, Tor: 52 |
| sex | 11 | 0.97 | FALSE | 2 | mal: 168, fem: 165 |

Variable type: numeric

| skim_variable | n_missing | complete_rate | min | p0 | p25 | p50 | p75 | p100 | hist |
|-------------------|-----------|---------------|----------|---------|--------|--------|-------|------|------|
| bill_length2mm | 0.99 | 43.925.46 | 32.1 | 39.23 | 44.45 | 48.5 | 59.6 | | |
| bill_depth2mm | 0.99 | 17.151.97 | 13.1 | 15.60 | 17.30 | 18.7 | 21.5 | | |
| flipper_length_mm | 0.99 | 200.924.06 | 172.01 | 190.00 | 197.02 | 213.02 | 31.0 | | |
| body_mass2_g | 0.99 | 4201.851.92 | 700.35 | 50.00 | 50.00 | 50.63 | 00.0 | | |
| year | 0 | 1.00 | 2008.032 | 2007.20 | 07.20 | 08.20 | 09.20 | 09.0 | |

Les résultats obtenus grâce à cette fonction sont nombreux. La première section nous donne des informations sur le tableau :

- son nom, son nombre de lignes et de colonnes
- la nature des variables qu’il contient (ici 3 facteurs et 5 variables numériques)
- la présence de variables utilisées pour faire des regroupements (il n’y en a pas encore à ce stade)

Ensuite, un bloc apporte des informations sur chaque facteur présent dans le tableau :

- le nom de la variable catégorielle (`skim_variable`)
- le nombre de données manquantes (`n_missing`) et le taux de “données complètes” (`complete_rate`)
- des informations sur le nombre de modalités (`n_unique`)

- le nombre d'observations pour les modalités les plus représentées (`top_counts`)

En un coup d'œil, on sait donc que 3 espèces sont présentes (et on connaît leurs effectifs), on sait que les manchots ont été échantillonnées sur 3 îles, et on sait que le sexe de 11 individu (sur 344) est inconnu. Pour le reste, il y a presque autant de femelles que de mâles.

Le dernier bloc renseigne sur les variables numériques. Pour chaque d'elle, on a :

- le nom de la variable numérique (`skim_variable`)
- le nombre de données manquantes (`n_missing`) et le taux de “données complètes” (`complete_rate`)
- la moyenne (`mean`) et l'écart-type (`sd`), ce qui est une nouveauté par rapport à la fonction `summary()`
- les valeurs minimales (`p0`), de premier quartile (`p25`), de second quartile (`p50`, c'est la médiane !), de troisième quartile (`p75`) et la valeur maximale (`p100`)
- un histogramme très simple qui donne un premier aperçu grossier de la forme de la distribution

Là encore, en un coup d'œil, on dispose donc de toutes les informations pertinentes pour juger de la distribution, de la position et de la dispersion de chaque variable numérique du jeu de données.

1.5.2.2 Tableau groupé

La fonction `skim()`, déjà très pratique, le devient encore plus lorsque l'on choisit de lui fournir seulement certaines variables, et qu'on fait certains regroupements. Par exemple, on peut sélectionner les variables relatives aux dimensions du bec (`bill_length_mm` et `bill_depth_mm`) avec la fonction `select()` que nous connaissons déjà, et demander un résumé des données pour chaque espèce grâce à la fonction `group_by()` que nous connaissons également :

```
penguins |>                                     # Avec le tableau penguins...
  select(species,
         bill_length_mm,
         bill_depth_mm) |>                       # Je sélectionne les variables d'intérêt...
```

```
group_by(species) |> # Je regroupe par espèce...
skim() # Et je produis un résumé des données
```

Table 1.4: Data summary

| | |
|------------------------|---------------|
| Name | group_by(...) |
| Number of rows | 344 |
| Number of columns | 3 |
| Column type frequency: | |
| numeric | 2 |
| Group variables | species |

Variable type: numeric

| skim_variable | species | missing | complete | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
|----------------|-----------|---------|----------|-------|--------|-------|-------|-------|------|------|------|
| bill_length_mm | Adelie | 0.99 | 38.79 | 66.32 | 136.73 | 38.80 | 40.75 | 46.0 | | | |
| bill_length_mm | Chinstrap | 1.00 | 48.83 | 34.40 | 946.35 | 49.55 | 51.05 | 58.0 | | | |
| bill_length_mm | Gentoo | 0.99 | 47.50 | 30.84 | 40.94 | 45.30 | 47.30 | 49.55 | 59.6 | | |
| bill_depth_mm | Adelie | 0.99 | 18.35 | 1.22 | 15.51 | 17.50 | 18.40 | 19.00 | 21.5 | | |
| bill_depth_mm | Chinstrap | 1.00 | 18.42 | 1.14 | 16.41 | 17.50 | 18.45 | 19.40 | 20.8 | | |
| bill_depth_mm | Gentoo | 0.99 | 14.98 | 0.98 | 13.11 | 14.20 | 15.00 | 15.70 | 17.3 | | |

On constate ici que pour chaque variable numérique sélectionnée, des statistiques descriptives détaillées nous sont fournies pour chacune des 3 espèces. Ce premier examen semble montrer que :

- L'espèce Adélie est celle qui possède le bec le plus court (ses valeurs de moyennes, médianes et quartiles sont plus faibles que celles des 2 autres espèces).
- L'espèce Gentoo est celle qui possède le bec le plus fin, ou le moins épais (ses valeurs de moyennes, médianes et quartiles sont plus faibles que celles des 2 autres espèces)
- Il ne semble pas y avoir de fortes différences d'écart-types (donc des dispersions des points autour des moyennes) entre les 3 espèces : pour chacune des 2 variables numériques, des valeurs d'écart-types comparables sont en effet observées pour les 3 espèces

- La distribution des 2 variables numériques semble approximativement suivre une distribution symétrique pour les 3 espèces, avec une forme de courbe en cloche. Les distributions devraient donc suivre à peu une distribution normale

i Note

Vous comprenez j'espère l'importance d'examiner ce genre de résumé des données avant de vous lancer dans des tests statistiques. Ils sont un complément indispensable aux explorations graphiques que vous devez également prendre l'habitude de réaliser pour mieux appréhender et comprendre la nature de vos données. Puisque chaque jeu de données est unique, vous devrez vous adapter à la situation et aux questions scientifiques qui vous sont posées (ou que vous vous posez !) : les choix qui seront pertinents pour une situation ne le seront pas nécessairement pour une autre. Mais dans tous les cas, pour savoir où vous allez et pour ne pas faire de bêtise au moment des tests statistiques et de leur interprétation, **vous devrez toujours explorer vos données, avec des graphiques exploratoires et des statistiques descriptives**.

1.5.3 Exercice

En utilisant les fonctions de résumé abordées jusqu'ici et le tableau `weather`, répondez aux questions suivantes :

1. Dans quel aéroport de New York les précipitations moyennes ont-elle été les plus fortes en 2013 ?
2. Dans quel aéroport de New York la vitesse du vent moyenne était-elle la plus forte en 2013 ? Quelle est cette vitesse ?
3. Dans quel aéroport de New York les rafales de vent étaient-elles les plus variables en 2013 ? Quel indice statistique vous donne cette information et quelle est sa valeur ?
4. Les précipitations dans les 3 aéroports de New-York ont-elles une distribution symétrique ?
5. Quelle est la température médiane observée en 2013 tous aéroports confondus ?

6. Tous aéroports confondus, quel est le mois de l'année où la température a été la plus variable en 2013 ? Quelles étaient les températures minimales et maximales observées ce mois-là ?

2 Dispersion et incertitude

2.1 Pré-requis

Nous avons ici besoin des packages suivants :

```
library(tidyverse)
library(palmerpenguins)
library(nycflights13)
```

Pensez à les charger en mémoire si ce n'est pas déjà fait ou si vous venez de démarrer une nouvelle session de travail.

2.2 La notion de dispersion

Comme expliqué plus haut, les **indices de dispersion** nous renseignent sur la variabilité des données autour de la valeur centrale (moyenne ou médiane) d'une population ou d'un échantillon. L'écart-type, la variance et l'intervalle interquartile sont 3 exemples d'indices de dispersion. Prenons l'exemple de l'écart-type. Un écart-type faible indique que la majorité des observations ont des valeurs proches de la moyenne. À l'inverse, un écart-type important indique que la plupart des points sont éloignés de la moyenne. L'écart-type est une caractéristique de la population que l'on étudie grâce à un échantillon, au même titre que la moyenne. En travaillant sur un échantillon, on espère accéder aux vraies grandeurs de la population. Même si ces vraies grandeurs sont à jamais inaccessibles (on ne connaîtra jamais parfaitement quelle est la vraie valeur de moyenne μ ou d'écart-type σ de la population), on espère qu'avec un échantillonnage réalisé correctement, la moyenne de l'échantillon (\bar{x} ou m) et l'écart-type (s) de l'échantillon reflètent assez fidèlement les valeurs de la population générale. C'est la notion d'estimateur, intimement liée à la notion d'**inférence**

statistique : la moyenne de l'échantillon, que l'on connaît avec précision, est un estimateur de la moyenne μ de la population qui restera à jamais inconnue. C'est la raison pour laquelle la moyenne de l'échantillon est parfois notée $\hat{\mu}$ (en plus de \bar{x} ou m). De même, l'écart-type s et la variance s^2 d'un échantillon sont des estimateurs de l'écart-type σ et de la variance σ^2 de la population générale. C'est la raison pour laquelle on les note parfois $\hat{\sigma}$ et $\hat{\sigma}^2$ respectivement. L'accent circonflexe se prononce "chapeau". On dit donc que $\hat{\sigma}$ (sigma chapeau, l'écart-type de l'échantillon) est un estimateur de l'écart-type de la population générale. Comme nous l'avons vu, les indices de dispersion doivent accompagner les indices de position lorsque l'on décrit des données, car présenter une valeur de moyenne, ou de médiane seule n'a pas de sens : il faut savoir à quel point les données sont proches ou éloignées de la tendance centrale pour savoir si, dans la population générale, les indicateurs de position correspondent ou non, aux valeurs portées par la majorité des individus.

Nous avons vu plus haut comment calculer des indices de position et de dispersion. Tout ceci devrait donc être clair pour vous à ce stade.

2.3 La notion d'incertitude

Par ailleurs, puisqu'on ne sait jamais avec certitude si nos estimations (de moyennes ou d'écart-types ou de tout autre paramètre) reflètent fidèlement ou non les vraies valeurs de la population, nous devons quantifier à quel point nos estimations s'écartent de celles de la population générale. C'est tout l'intérêt des statistiques et c'est ce que permettent les **indices d'incertitude** : on ne connaîtra jamais la vraie valeur de moyenne ou d'écart-type de la population, mais on peut quantifier à quel point nos estimations (basées sur un échantillon) sont précises ou imprécises.

Les deux indices d'incertitude les plus connus (et les plus utilisés) sont l'**intervalle de confiance à 95%** (de la moyenne ou de tout autre estimateur ; les formules sont nombreuses et il n'est pas utile de les détailler ici : nous verrons comment les calculer plus bas) et l'**erreur standard** de la moyenne ($se_{\bar{x}}$), dont la formule est la suivante :

$$se_{\bar{x}} = \frac{s}{\sqrt{n}}$$

avec s , l'écart-type de l'échantillon et n la taille de l'échantillon.

Comme pour la moyenne, on peut calculer l'erreur standard d'un écart-type, d'une médiane, d'une proportion, ou de tout autre estimateur calculé sur un échantillon. Cet indice d'incertitude ne nous renseigne pas sur une grandeur de la population générale qu'on chercherait à estimer, mais bien sur l'incertitude associée à une estimation que nous faisons en travaillant sur un échantillon de taille forcément limitée. Tout processus d'échantillonnage est forcément entaché d'incertitude, causée entre autre par le hasard de l'échantillonnage (ou fluctuation d'échantillonnage). Puisque nous travaillons sur des échantillons forcément imparfaits, les indices d'incertitude vont nous permettre de quantifier à quel point nos estimations s'écartent des vraies valeurs de la population. Ces "vraies valeurs", faute de pouvoir collecter tous les individus de la population, resteront à jamais inconnues.

! Autrement dit...

Quand on étudie des populations naturelles grâce à des échantillons **on se trompe toujours**. Les statistiques nous permettent de **quantifier à quel point on se trompe** grâce aux indices d'incertitude, et c'est déjà pas mal !

En examinant la formule de l'erreur standard de la moyenne présentée ci-dessus, on comprend intuitivement que plus la taille de l'échantillon (n) augmente, plus l'erreur standard (donc l'incertitude) associée à notre estimation de moyenne diminue. Autrement dit, plus les données sont abondantes dans l'échantillon, meilleure sera notre estimation de moyenne, et donc, moins le risque de raconter des bêtises sera grand.

L'autre indice d'incertitude très fréquemment utilisé est l'intervalle de confiance à 95% (de la moyenne, de la médiane, de la variance, ou de toute autre estimateur calculé dans un échantillon). L'intervalle de confiance nous renseigne sur la gamme des valeurs les plus probables pour un paramètre de la population étudiée. Par exemple, si j'observe, dans

un échantillon, une moyenne de 10, avec un intervalle de confiance calculé de $[7 ; 15]$, cela signifie que, dans la population générale, la vraie valeur de moyenne a de bonnes chances de se trouver dans l'intervalle $[7 ; 15]$. Dans la population générale, toutes les valeurs comprises entre 7 et 15 sont vraisemblables pour la moyenne alors que les valeurs situées en dehors de cet intervalle sont moins probables. Une autre façon de comprendre l'intervalle de confiance est de dire que si je récupère un grand nombre d'échantillons dans la même population, en utilisant exactement le même protocole expérimental, 95% des échantillons que je vais récupérer auront une moyenne située à l'intérieur de l'intervalle de confiance à 95%, et 5% des échantillons auront une moyenne située à l'extérieur de l'intervalle de confiance à 95%. C'est une notion qui n'est pas si évidente que ça à comprendre, donc prenez bien le temps de relire cette section si besoin, et de poser des questions le cas échéant.

Concrètement, plus l'intervalle de confiance est large, moins notre confiance est grande. Si la moyenne d'un échantillon vaut $\bar{x} = 10$, et que son intervalle de confiance à 95% vaut $[9,5 ; 11]$, la gamme des valeurs probables pour la moyenne de la population est étroite. Autrement dit, la moyenne de l'échantillon (10), a de bonnes chances d'être très proche de la vraie valeur de moyenne de la population générale (vraisemblablement comprise quelque part entre 9,5 et 11). À l'inverse, si l'intervalle de confiance à 95% de la moyenne vaut $[4 ; 17]$, la gamme des valeurs possibles pour la vraie moyenne de la population est grande. La moyenne de l'échantillon aura donc de grandes chances d'être assez éloignée de la vraie valeur de la population.

La notion d'intervalle de confiance à 95% est donc très proche de celle d'erreur standard. D'ailleurs, pour de nombreux paramètres, l'intervalle de confiance est calculé à partir de l'erreur standard.

2.4 Calcul de l'erreur standard de la moyenne

Contrairement aux indices de position et de dispersion, il n'existe pas de fonction intégrée à \mathbb{R} qui permette de calculer l'erreur standard de la moyenne. Toutefois, sa formule très

simple nous permet de la calculer à la main quand on en a besoin grâce à la fonction `summarise()`.

Par exemple, reprenons les données de température (tableau `weather` du package `nycflights13`, colonne `temp`) dans les 3 aéroports de New York (colonne `origin`). Imaginons que nous souhaitions étudier les fluctuations de températures au fil des mois de l'année 2013 :

1. Je vais commencer par transformer les températures (fournies en degrés Fahrenheit) en degrés Celsius :

```
weather |>
  mutate(temp_celsius = (temp - 32) / 1.8)

# A tibble: 26,115 x 16
  origin year month   day hour temp dewp humid wind_dir wind_speed
  <chr>  <int> <int> <int> <int> <dbl> <dbl> <dbl> <dbl>    <dbl>
1 EWR    2013     1     1     1  39.0  26.1  59.4    270     10.4
2 EWR    2013     1     1     2  39.0  27.0  61.6    250      8.06
3 EWR    2013     1     1     3  39.0  28.0  64.4    240     11.5
4 EWR    2013     1     1     4  39.9  28.0  62.2    250     12.7
5 EWR    2013     1     1     5  39.0  28.0  64.4    260     12.7
6 EWR    2013     1     1     6  37.9  28.0  67.2    240     11.5
7 EWR    2013     1     1     7  39.0  28.0  64.4    240     15.0
8 EWR    2013     1     1     8  39.9  28.0  62.2    250     10.4
9 EWR    2013     1     1     9  39.9  28.0  62.2    260     15.0
10 EWR   2013     1     1    10  41    28.0  59.6    260     13.8
# i 26,105 more rows
# i 6 more variables: wind_gust <dbl>, precip <dbl>, pressure <dbl>,
#   visib <dbl>, time_hour <dtm>, temp_celsius <dbl>
```

2. Ensuite, je détermine, pour chaque jour de chaque mois de l'année, et pour chaque aéroport, quelle est la température maximale atteinte, et je stocke ces valeurs dans un nouvel objet pour pouvoir le ré-utiliser :

```
temp_jour <- weather |>
  mutate(temp_celsius = (temp - 32) / 1.8) |>
  summarise(temperature_max = max(temp_celsius, na.rm = TRUE),
            .by = c(origin, month, day))

temp_jour
```

```
# A tibble: 1,092 x 4
  origin month   day temperature_max
  <chr>   <int> <int>           <dbl>
1 EWR     1     1             5
2 EWR     1     2            1.10
3 EWR     1     3            1.10
4 EWR     1     4            4.4
5 EWR     1     5            6.7
6 EWR     1     6            8.9
7 EWR     1     7            8.3
8 EWR     1     8            9.4
9 EWR     1     9            10
10 EWR    1    10            10
# i 1,082 more rows
```

3. Je peux maintenant calculer la température moyenne mensuelle pour chaque aéroport :

```
temp_jour |>
  summarise(moyenne = mean(temperature_max, na.rm = TRUE),
            .by = c(origin, month))
```

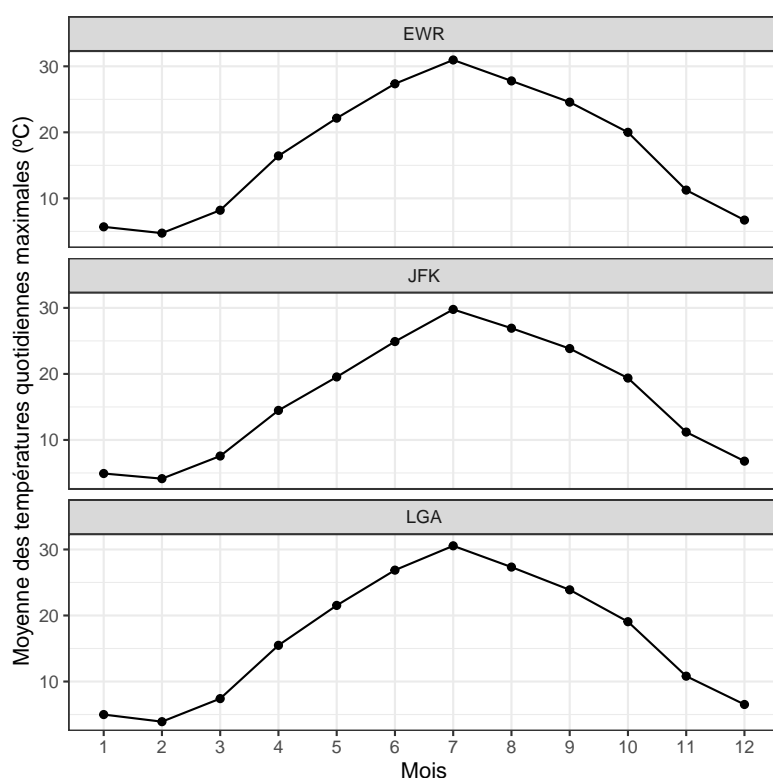
```
# A tibble: 36 x 3
  origin month moyenne
  <chr>   <int>   <dbl>
1 EWR     1     5.69
2 EWR     2     4.74
3 EWR     3     8.20
4 EWR     4    16.4
5 EWR     5    22.2
6 EWR     6    27.4
7 EWR     7    31.0
8 EWR     8    27.8
9 EWR     9    24.6
10 EWR    10    20.0
# i 26 more rows
```

Pour pouvoir réutiliser ce tableau, je lui donne un nom :

```
temp_moyennes <- temp_jour |>
  summarise(moyenne = mean(temperature_max, na.rm = TRUE),
            .by = c(origin, month))
```

Au final, je peux faire un graphique de l'évolution de ces températures :

```
temp_moyennes |>
  ggplot(aes(x = factor(month), y = moyenne)) +
  geom_line(aes(group = 1)) +
  geom_point() +
  facet_wrap(~origin, ncol = 1) +
  labs(x = "Mois",
       y = "Moyenne des températures quotidiennes maximales (°C)") +
  theme_bw()
```



Vous remarquerez que :

1. j'associe `factor(month)`, et non simplement `month`, à l'axe des x afin d'avoir, sur l'axe des abscisses, des chiffres cohérents allant de 1 à 12, et non des chiffres à virgule
2. l'argument `group = 1` doit être ajouté pour que la ligne reliant les points apparaisse. En effet, les lignes

sont censées relier des points qui appartiennent à une même série temporelle. Or ici, nous avons transformé `month` en facteur. Préciser `group = 1` permet d'indiquer à `geom_line()` que toutes les catégories du facteur `month` appartiennent au même groupe, que ce facteur peut être considéré comme une variable continue, et qu'il est donc correct de relier les points.

Pour les 3 aéroports, les profils de températures sont très proches. C'est tout à fait logique puisqu'ils sont situés dans un rayon de quelques kilomètres seulement. Le problème de ce graphique est que chaque point a été obtenu en calculant une moyenne. En janvier, nous avons fait la moyenne de 31 valeurs de températures quotidiennes pour chaque aéroport. En février, nous avons fait la moyenne de 28 valeurs de températures quotidiennes pour chaque aéroport. Et ainsi de suite pour tous les mois de l'année 2013. Puisque nous présentons des valeurs de moyennes, il nous faut présenter également **l'incertitude associée à ces calculs de moyennes**. Pour cela, nous devons calculer l'erreur standard des moyennes :

```
temp_jour |>
  summarise(moyenne = mean(temperature_max, na.rm = TRUE),
            N_obs = n(),
            erreur_standard = sd(temperature_max, na.rm = TRUE) / sqrt(N_obs),
            .by = c(origin, month))
```

A tibble: 36 x 5

| | origin | month | moyenne | N_obs | erreur_standard |
|----|--------|-------|---------|-------|-----------------|
| | <chr> | <int> | <dbl> | <int> | <dbl> |
| 1 | EWR | 1 | 5.69 | 31 | 1.14 |
| 2 | EWR | 2 | 4.74 | 28 | 0.762 |
| 3 | EWR | 3 | 8.20 | 31 | 0.649 |
| 4 | EWR | 4 | 16.4 | 30 | 0.919 |
| 5 | EWR | 5 | 22.2 | 31 | 1.02 |
| 6 | EWR | 6 | 27.4 | 30 | 0.769 |
| 7 | EWR | 7 | 31.0 | 31 | 0.700 |
| 8 | EWR | 8 | 27.8 | 31 | 0.385 |
| 9 | EWR | 9 | 24.6 | 30 | 0.716 |
| 10 | EWR | 10 | 20.0 | 31 | 0.882 |

i 26 more rows

Notre tableau de statistiques descriptives possède maintenant

2 colonnes supplémentaires : le nombre d'observations (que j'ai nommé `N_obs`), et l'erreur standard associée à chaque moyenne, calculée grâce à la formule vue plus haut $se_{\bar{x}} = \frac{s}{\sqrt{n}}$ (la fonction `sqrt()` permet de calculer la racine carrée). On constate que l'erreur standard, qui s'exprime dans la même unité que la moyenne, est variable selon les mois de l'année. Ainsi, pour l'aéroport de Newark, l'incertitude semble particulièrement faible pour le mois d'août (0.385 °C) mais presque 3 fois plus forte pour le mois de janvier (1.14 °C).

Une fois de plus, je donne un nom à ce tableau de données pour pouvoir le réutiliser plus tard :

```
temperatures_se <- temp_jour |>
  summarise(moyenne = mean(temperature_max, na.rm = TRUE),
            N_obs = n(),
            erreur_standard = sd(temperature_max, na.rm = TRUE) / sqrt(N_obs),
            .by = c(origin, month))
```

Notez que le package `ggplot2` contient une fonction permettant de calculer à la fois la moyenne et erreur standard de la moyenne d'un échantillon : `mean_se()`. Puisque cette fonction renvoie 3 valeurs (\bar{x} , $\bar{x}-se$ et $\bar{x}+se$), on utilise `reframe()` :

```
temp_jour |>
  reframe(mean_se(temperature_max),
          .by = c(origin, month))
```

```
# A tibble: 36 x 5
  origin month     y ymin ymax
  <chr>   <int> <dbl> <dbl> <dbl>
1 EWR     1  5.69  4.55  6.82
2 EWR     2  4.74  3.97  5.50
3 EWR     3  8.20  7.55  8.85
4 EWR     4 16.4  15.5 17.4
5 EWR     5 22.2  21.1 23.2
6 EWR     6 27.4  26.6 28.1
7 EWR     7 31.0  30.3 31.7
8 EWR     8 27.8  27.4 28.2
9 EWR     9 24.6  23.9 25.3
10 EWR    10 20.0  19.1 20.9
```

```
# i 26 more rows
```

Les résultats obtenus ne sont pas exactement au même format :

- la colonne `y` contient les valeurs de moyennes (\bar{x})
- la colonne `ymin` contient la valeur de moyenne moins une fois l'erreur standard ($\bar{x} - se$)
- la colonne `ymax` contient la valeur de moyenne plus une fois l'erreur standard ($\bar{x} + se$)

Il ne nous restera plus qu'à ajouter des barres d'erreur sur notre graphique pour visualiser l'incertitude associée à chaque valeur de moyenne.

2.5 Calculs d'intervalles de confiance à 95%

Comme pour les erreurs standard, il est possible de calculer des intervalles de confiance de n'importe quel estimateur calculé à partir d'un échantillon, pour déterminer la gamme des valeurs les plus probables pour les paramètres équivalents dans la population générale. Nous nous concentrerons ici sur le calcul des intervalles de confiance à 95% de la moyenne, mais nous serons amenés à examiner également l'intervalle de confiance de la médiane, puis, au cours de votre L3, l'intervalle de confiance à 95% d'une différence de moyennes.

Contrairement à l'erreur standard, il n'y a pas qu'une bonne façon de calculer l'intervalle de confiance à 95% d'une moyenne. Plusieurs formules existent et le choix de la formule dépend en partie de la distribution des données (la distribution suit-elle une loi Normale ou non) et de la taille de l'échantillon dont nous disposons (n est-il supérieur à 30 ou non ?). Dans la situation idéale d'une variable qui suit la distribution Normale, les bornes inférieures et supérieures de l'intervalle de confiance à 95% sont obtenues grâce à cette formule

$$\bar{x} - 1.96 \cdot \frac{s}{\sqrt{n}} < \mu < \bar{x} + 1.96 \cdot \frac{s}{\sqrt{n}}$$

Autrement dit, la vraie moyenne μ d'une population a de bonnes chances de se trouver dans un intervalle de plus ou moins 1.96 fois l'erreur standard de la moyenne. En première approximation, l'intervalle de confiance est donc la moyenne de l'échantillon \bar{x} plus ou moins 2 fois l'erreur standard (que nous avons appris à calculer à la main un peu plus tôt). On peut donc calculer à la main les bornes inférieures et supérieures de l'intervalle de confiance ainsi :

```
temp_jour |>
  reframe(mean_se(temperature_max, mult = 1.96),
          .by = c(origin, month))
```

```
# A tibble: 36 x 5
  origin month     y  ymin  ymax
  <chr>   <int> <dbl> <dbl> <dbl>
1 EWR         1  5.69  3.46  7.92
2 EWR         2  4.74  3.24  6.23
3 EWR         3  8.20  6.93  9.47
4 EWR         4 16.4  14.6  18.2
5 EWR         5 22.2  20.1  24.2
6 EWR         6 27.4  25.8  28.9
7 EWR         7 31.0  29.6  32.3
8 EWR         8 27.8  27.0  28.5
9 EWR         9 24.6  23.2  26.0
10 EWR        10 20.0  18.3  21.7
# i 26 more rows
```

Ici, grâce à l'argument `mult = 1.96` de la fonction `mean_se()` :

- la colonne `ymin` contient maintenant les valeurs de moyennes moins 1.96 fois l'erreur standard
- la colonne `ymax` contient maintenant les valeurs de moyennes plus 1.96 fois l'erreur standard

Dans la pratique, puisque cette méthode reste approximative et dépend de la nature des données dont on dispose, on utilisera plutôt des fonctions spécifiques qui calculeront pour nous les intervalles de confiance à 95% de nos estimateurs. C'est ce que permet en particulier la fonction `mean_cl_normal()` du package `ggplot2`. Il est toutefois important de bien comprendre qu'il y a un lien étroit

entre l'erreur standard (l'incertitude associées à l'estimation d'un paramètre d'une population à partir des données d'un échantillon), et l'intervalle de confiance à 95% de ce paramètre.

```
temp_jour |>
  reframe(mean_cl_normal(temperature_max),
          .by = c(origin, month))
```

```
# A tibble: 36 x 5
  origin month     y  ymin  ymax
  <chr>   <int> <dbl> <dbl> <dbl>
1 EWR         1  5.69  3.36  8.01
2 EWR         2  4.74  3.17  6.30
3 EWR         3  8.20  6.88  9.53
4 EWR         4 16.4   14.6  18.3
5 EWR         5 22.2   20.1  24.2
6 EWR         6 27.4   25.8  28.9
7 EWR         7 31.0   29.5  32.4
8 EWR         8 27.8   27.0  28.6
9 EWR         9 24.6   23.1  26.0
10 EWR        10 20.0   18.2  21.8
# i 26 more rows
```

Comme dans les tableaux précédents, 3 nouvelles colonnes ont été créés :

- `y` contient toujours la moyenne des températures mensuelles pour chaque aéroport
- `ymin` contient maintenant les bornes inférieures de l'intervalle à 95% des moyennes
- `ymax` contient maintenant les bornes supérieures de l'intervalle à 95% des moyennes

Pour que la suite soit plus claire, nous allons afficher et donner des noms à ces différents tableaux en prenant soin de renommer les colonnes pour plus de clarté.

Tout d'abord, nous disposons du tableau `temperatures_se`, qui contient, les moyennes des températures mensuelles de chaque aéroport de New York en 2013, et les erreurs standard de ces moyennes :

```
temperatures_se
```

```
# A tibble: 36 x 5
  origin month moyenne N_obs erreur_standard
  <chr>   <int>   <dbl> <int>         <dbl>
1 EWR     1     5.69   31           1.14
2 EWR     2     4.74   28           0.762
3 EWR     3     8.20   31           0.649
4 EWR     4    16.4   30           0.919
5 EWR     5    22.2   31           1.02
6 EWR     6    27.4   30           0.769
7 EWR     7    31.0   31           0.700
8 EWR     8    27.8   31           0.385
9 EWR     9    24.6   30           0.716
10 EWR    10    20.0   31           0.882
# i 26 more rows
```

Ensuite, nous avons produit un tableau presque équivalent que nous allons nommer `temperature_se_bornes` et pour lequel nous allons modifier le nom des colonnes `y`, `ymin` et `ymax` :

```
temperature_se_bornes <- temp_jour |>
  reframe(mean_se(temperature_max),
          .by = c(origin, month)) |>
  rename(moyenne = y,
         moyenne_moins_se = ymin,
         moyenne_plus_se = ymax)

temperature_se_bornes
```

```
# A tibble: 36 x 5
  origin month moyenne moyenne_moins_se moyenne_plus_se
  <chr>   <int>   <dbl>         <dbl>         <dbl>
1 EWR     1     5.69           4.55           6.82
2 EWR     2     4.74           3.97           5.50
3 EWR     3     8.20           7.55           8.85
4 EWR     4    16.4           15.5           17.4
5 EWR     5    22.2           21.1           23.2
6 EWR     6    27.4           26.6           28.1
7 EWR     7    31.0           30.3           31.7
```

```

 8 EWR      8  27.8          27.4          28.2
 9 EWR      9  24.6          23.9          25.3
10 EWR     10  20.0          19.1          20.9
# i 26 more rows

```

Nous avons ensuite calculé manuellement des intervalles de confiance approximatifs, avec la fonction `mean_se()` et son argument `mult = 1.96`. Là encore, nous allons stocker cet objet dans un tableau nommé `temperatures_ci_approx`, et nous allons modifier le nom des colonnes `y`, `ymin`, et `ymin` :

```

temperature_ci_approx <- temp_jour |>
  reframe(mean_se(temperature_max, mult = 1.96),
    .by = c(origin, month)) |>
  rename(moyenne = y,
    ci_borne_inf = ymin,
    ci_borne_sup = ymax)

temperature_ci_approx

```

```

# A tibble: 36 x 5
  origin month moyenne ci_borne_inf ci_borne_sup
  <chr>   <int>   <dbl>       <dbl>       <dbl>
1 EWR     1     5.69         3.46         7.92
2 EWR     2     4.74         3.24         6.23
3 EWR     3     8.20         6.93         9.47
4 EWR     4    16.4         14.6        18.2
5 EWR     5    22.2         20.1        24.2
6 EWR     6    27.4         25.8        28.9
7 EWR     7    31.0         29.6        32.3
8 EWR     8    27.8         27.0        28.5
9 EWR     9    24.6         23.2        26.0
10 EWR    10    20.0         18.3        21.7
# i 26 more rows

```

Enfin, nous avons calculé les intervalles de confiance avec une fonction spécialement dédiée à cette tâche : la fonction `mean_cl_normal()`. Nous allons stocker cet objet dans un tableau nommé `temperatures_ci`, et nous allons modifier le nom des colonnes `y`, `ymin`, et `ymin` :

```

temperature_ci <- temp_jour |>
  reframe(mean_cl_normal(temperature_max),
    .by = c(origin, month)) |>
  rename(moyenne = y,
    ci_borne_inf = ymin,
    ci_borne_sup = ymax)

temperature_ci

```

```

# A tibble: 36 x 5
  origin month moyenne ci_borne_inf ci_borne_sup
  <chr>   <int>   <dbl>     <dbl>     <dbl>
1 EWR     1     5.69       3.36       8.01
2 EWR     2     4.74       3.17       6.30
3 EWR     3     8.20       6.88       9.53
4 EWR     4    16.4      14.6      18.3
5 EWR     5    22.2      20.1      24.2
6 EWR     6    27.4      25.8      28.9
7 EWR     7    31.0      29.5      32.4
8 EWR     8    27.8      27.0      28.6
9 EWR     9    24.6      23.1      26.0
10 EWR    10    20.0      18.2      21.8
# i 26 more rows

```

Maintenant, si l'on compare les 2 tableaux contenant les calculs d'intervalles de confiance de la moyenne, on constate que les résultats sont très proches :

```

temperature_ci_approx
temperature_ci

```

Les bornes inférieures et supérieures des intervalles de confiance à 95% des moyennes ne sont pas égales quand on les calcule manuellement de façon approchée et quand on les calcule de façon exacte, mais les différences sont minimes.

```

# A tibble: 36 x 5      # A tibble: 36 x 5
  origin month moyenne ci_borne_inf origin month moyenne ci_borne_inf ci_borne_sup
  <chr> <int> <dbl> <dbl> <chr> <int> <dbl> <dbl> <dbl>
1 EWR      1  5.69  1 EWR 3.46  1  57.692  3.36  8.01
2 EWR      2  4.74  2 EWR 3.24  2  46.723  3.17  6.30
3 EWR      3  8.20  3 EWR 6.93  3  89.247  6.88  9.53
4 EWR      4 16.4  4 EWR14.6  4 168.42 14.6 18.3
5 EWR      5 22.2  5 EWR20.1  5 224.22 20.1 24.2
6 EWR      6 27.4  6 EWR25.8  6 278.49 25.8 28.9
7 EWR      7 31.0  7 EWR29.6  7 332.03 29.5 32.4
8 EWR      8 27.8  8 EWR27.0  8 278.85 27.0 28.6
9 EWR      9 24.6  9 EWR23.2  9 246.60 23.1 26.0
10 EWR     10 20.0 10 EWR18.3 10 201.07 18.2 21.8
# i 26 more rows      # i 26 more rows

```


3 Visualiser l'incertitude et la dispersion

3.1 Pré-requis

Nous avons ici besoin des packages suivants :

```
library(tidyverse)
library(palmerpenguins)
library(nycflights13)
```

Pensez à les charger en mémoire si ce n'est pas déjà fait ou si vous venez de démarrer une nouvelle session de travail.

Il existe plusieurs façons de représenter visuellement les **positions**, les **dispersions** et les incertitudes. Concernant les positions et les dispersions tout d'abord, nous avons déjà vu plusieurs façons de faire au semestre 3, en particulier dans les parties consacrées aux histogrammes, aux stripcharts et aux boxplots. Nous reprenons ici brièvement chacun de ces 3 types de graphique afin de les remettre en contexte avec ce que nous avons appris ici.

Dans un dernier temps, nous verrons comment visualiser l'**incertitude** associée à des calculs de moyennes ou de variances grâce aux barres d'erreurs ou aux encoches des boîtes à moustaches.

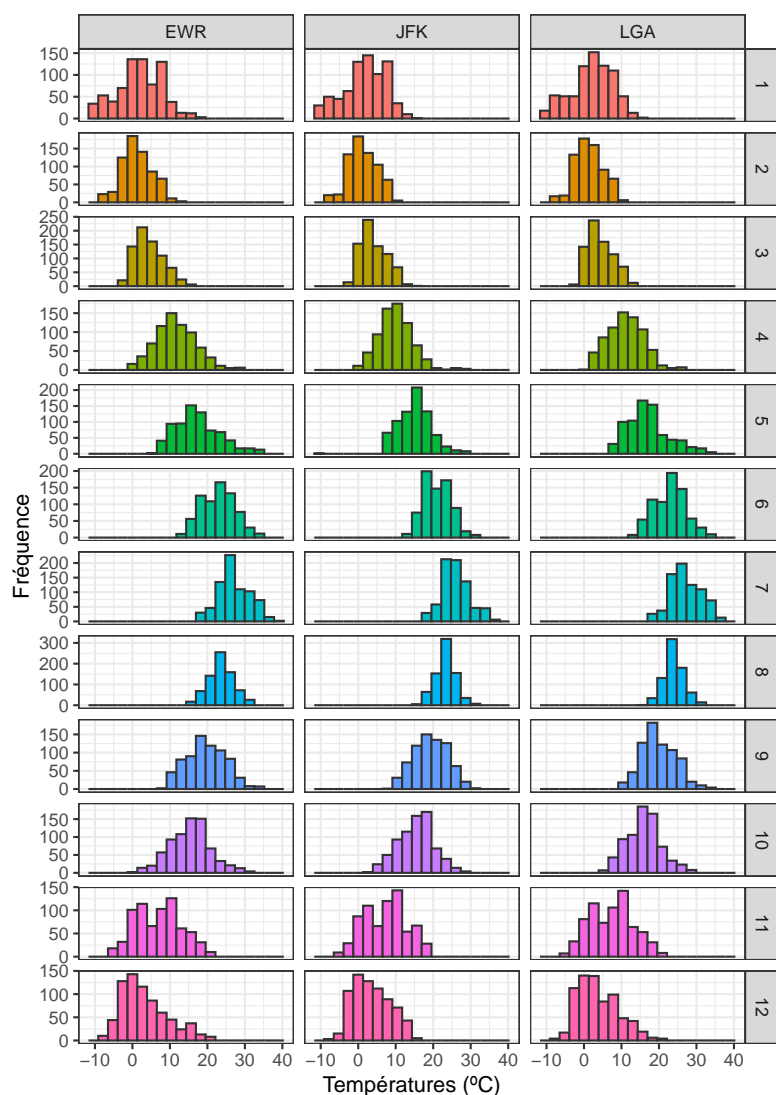
3.2 Position et dispersion : les histogrammes

Je vous renvoie à [la partie sur les histogrammes](#) du livre en ligne de biométrie du semestre 3 si vous avez besoin de vous rafraîchir la mémoire. Jetez aussi un œil la partie sur [les histogrammes facettés](#).

Les histogrammes permettent de déterminer à la fois où se trouvent les valeurs les plus fréquemment observées (la position du pic principal correspond à la tendance centrale), et la dispersion (ou variabilité) des valeurs autour de la tendance centrale. Par exemple, la fonction `facet_grid()` permet de faire des histogrammes des températures pour chaque aéroport de New York et chaque mois de l'année 2013 :

```
weather |>
  mutate(temp_celsius = (temp - 32) / 1.8) |>
  ggplot(aes(x = temp_celsius, fill = factor(month))) +
  geom_histogram(bins = 20, color = "grey20", show.legend = FALSE) +
  facet_grid(factor(month) ~ origin, scales = "free_y") +
  labs(x = "Températures (°C)", y = "Fréquence") +
  theme_bw()
```

Warning: Removed 1 row containing non-finite outside the scale range (``stat_bin()``).



Ici, 36 histogrammes sont produits. Ils permettent de constater que :

- les températures évoluent à peu près de la même façon dans les 3 aéroports (les 3 colonnes de graphiques se ressemblent beaucoup).
- les températures moyennes sont plus faibles en hiver qu'en été, et qu'elles sont intermédiaires au printemps et à l'automne. C'est bien la position des pics sur l'axe des abscisses qui nous renseigne là-dessus. On sait aussi que les températures moyennes les plus fortes sont autour de 25°C en juillet, alors que ces mêmes tempéra-

tures moyennes sont proches de 0°C en janvier, février et décembre.

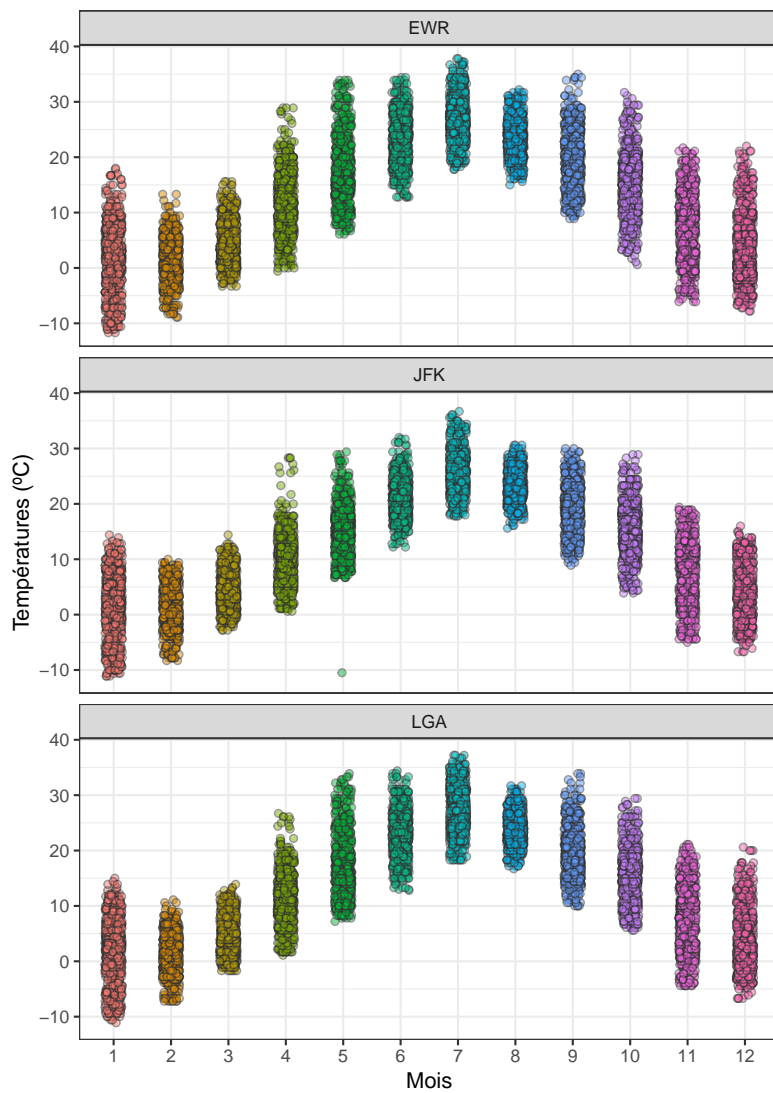
- la variabilité des températures est comparable pour la plupart des mois de l'année, avec une exception au mois d'août où la dispersion des valeurs semble plus limitée. Cette fois, c'est l'étalement de l'histogramme qui nous renseigne sur la dispersion.

3.3 Position et dispersion : les stripcharts

Une autre façon de visualiser à la fois les tendances centrales et les dispersions consiste à produire un nuage de points "stripchart". Là encore, je vous renvoie à [la partie sur les stripcharts](#) du livre en ligne de biométrie du semestre 3 si vous avez besoin de vous rafraîchir la mémoire.

```
weather |>
  mutate(temp_celsius = (temp - 32) / 1.8) |>
  ggplot(aes(x = factor(month), y = temp_celsius, fill = factor(month))) +
  geom_jitter(shape = 21, color = "grey20", show.legend = FALSE,
             width = 0.15, height = 0,
             alpha = 0.5) +
  facet_wrap(~ origin, ncol = 1) +
  labs(x = "Mois", y = "Températures (°C)") +
  theme_bw()
```

Warning: Removed 1 row containing missing values or values outside the scale range (``geom_point()``).



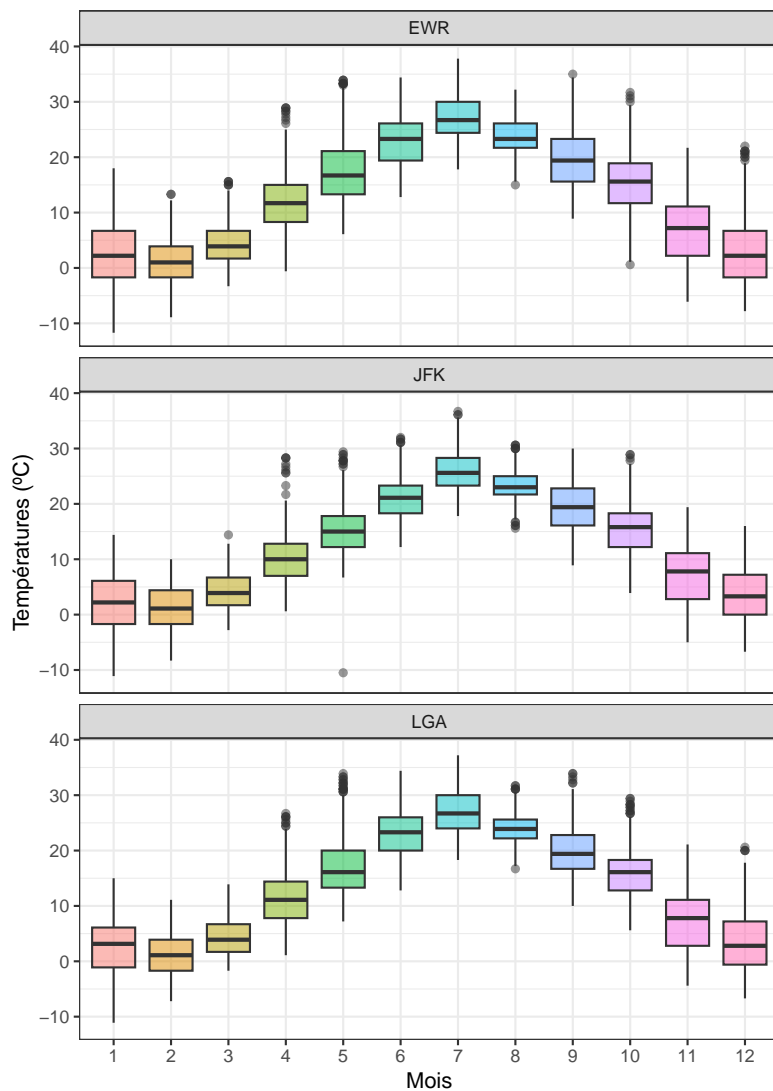
Cette fois, nous visualisons la totalité des données disponibles, et non les données regroupées dans des classes plus ou moins arbitraires. Mais là encore, on peut facilement comparer la position de chaque série de données : pour les mois d'été, les températures sont plus élevées que pour les mois d'hiver. Et la dispersion des données est aussi facile à comparer entre les mois. Par exemple, la variabilité des températures en janvier est nettement supérieure à celle du mois de février. C'est ici l'étendue du nuage de points sur l'axe des ordonnées qui nous permet de le dire.

3.4 Position et dispersion : les boxplots

La dernière façon classique de visualiser à la fois les tendances centrales et les dispersions consiste à produire un graphique boîtes à moustaches, ou “boxplot”. Là encore, je vous renvoie à [la partie sur les boxplots](#) du livre en ligne de biométrie du semestre 3 si vous avez besoin de vous rafraîchir la mémoire.

```
weather |>
  mutate(temp_celsius = (temp - 32) / 1.8) |>
  ggplot(aes(x = factor(month), y = temp_celsius, fill = factor(month))) +
  geom_boxplot(show.legend = FALSE, alpha = 0.5) +
  facet_wrap(~ origin, ncol = 1) +
  labs(x = "Mois", y = "Températures (°C)") +
  theme_bw()
```

Warning: Removed 1 row containing non-finite outside the scale range (``stat_boxplot()``).



Vous voyez que le code est très proche pour produire un strip-chart ou un boxplot. Comme indiqué au semestre 3, les différents éléments de chaque boîte nous renseignent sur la position et sur la dispersion des données pour chaque mois et chaque aéroport :

- La limite inférieure de la boîte correspond au premier quartile : 25% des données de l'échantillon sont situées au-dessous de cette valeur.
- La limite supérieure de la boîte correspond au troisième quartile : 75% des données de l'échantillon sont situées au-dessous de cette valeur.

- Le segment épais à l'intérieur de la boîte correspond au second quartile : c'est la médiane de l'échantillon, qui nous renseigne sur la position de la distribution. 50% des données de l'échantillon sont situées au-dessus de cette valeur, et 50% au-dessous.
- La hauteur de la boîte correspond à l'étendue (ou intervalle) interquartile ou Inter Quartile Range (IQR) en anglais. On trouve dans cette boîte 50% des observations de l'échantillon. C'est une mesure de la dispersion des 50% des données les plus centrales. Une boîte plus allongée indique donc une plus grande dispersion.
- Les moustaches correspondent à des valeurs qui sont en dessous du premier quartile (pour la moustache du bas) et au-dessus du troisième quartile (pour la moustache du haut). La règle utilisée dans R est que ces moustaches s'étendent jusqu'aux valeurs minimales et maximales de l'échantillon, mais elles ne peuvent en aucun cas s'étendre au-delà de 1,5 fois la hauteur de la boîte (1,5 fois l'IQR) vers le haut et le bas. Si des points apparaissent au-delà des moustaches (vers le haut ou le bas), ces points sont appelés "outliers". On peut en observer ici pour plusieurs mois et pour les 3 aéroports (par exemple, en avril dans les 3 aéroports). Ce sont des points qui s'éloignent du centre de la distribution de façon importante puisqu'ils sont au-delà de 1,5 fois l'IQR de part et d'autre du premier ou du troisième quartile. Il peut s'agir d'anomalies de mesures, d'anomalies de saisie des données, ou tout simplement, d'enregistrements tout à fait valides mais atypiques ou extrêmes ; il ne s'agit donc pas toujours de point aberrants. J'attire votre attention sur le fait que la définition de ces outliers est relativement arbitraire. Nous pourrions faire le choix d'étendre les moustaches jusqu'à 1,8 fois l'IQR (ou 2, ou 2,5). Nous observerions alors beaucoup moins d'outliers. D'une façon générale, la longueur des moustaches renseigne sur la variabilité des données en dehors de la zone centrale. Plus elles sont longues, plus la variabilité est importante. Très souvent, l'examen attentif des outliers est utile car il nous permet d'en apprendre plus sur le comportement extrême de certaines observations.

Lorsque les boîtes ont une forme à peu près symétrique de part et d'autre de la médiane (c'est le cas pour cet

exemple dans la plupart des catégories), cela signifie qu'un histogramme des mêmes données serait symétrique également.

Les stripcharts et les boxplots sont donc un bon moyen de comparer rapidement la position et la dispersion d'un grand nombre de séries de données : ici, en quelques lignes de code, nous en comparons 12 pour chacun des 3 aéroports de New York.

Les histogrammes sont plus utiles lorsqu'il y a moins de catégories à comparer. Ils permettent en outre de mieux visualiser les distributions non symétriques, ou qui présentent plusieurs pics (distribution bi- ou poly-modales).

3.5 Visualiser l'incertitude : les barres d'erreur

Comme évoqué plus haut, il est important de ne pas confondre **dispersion** et **incertitude**. Lorsque l'on visualise des moyennes calculées à partir des données d'un échantillon, il est important de faire apparaître des barres d'erreurs, qui correspondent en général :

- soit à l'erreur standard de la moyenne
- soit à l'intervalle de confiance à 95% de la moyenne

Puisque deux choix sont possibles, il sera important de préciser systématiquement dans la légende du graphique, la nature des barres représentées. Commençons par visualiser les températures mensuelles avec les erreurs standards. Pour cela, je reprends le tableau `temperatures_se` créé précédemment :

```
temperatures_se |>
  ggplot(aes(x = factor(month), y = moyenne)) +
  geom_line(aes(group = 1)) +
  geom_point() +
  geom_errorbar(aes(ymin = moyenne - erreur_standard,
                    ymax = moyenne + erreur_standard),
                width = 0.1) +
  facet_wrap(~origin, ncol = 1) +
  labs(x = "Mois",
       y = "Moyenne des températures quotidiennes maximales (°C)" ) +
```

theme_bw()

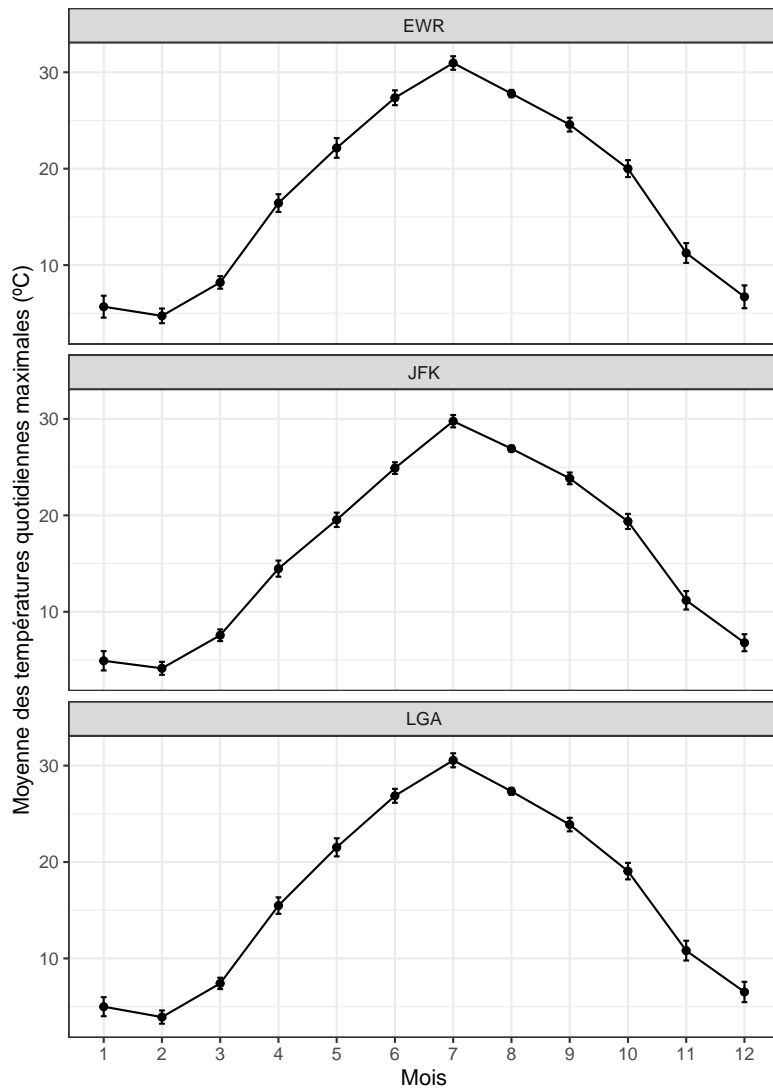


Figure 3.1: Températures moyennes mensuelles observées en 2013 dans les 3 aéroports de New York. Les barres d'erreur sont les erreurs standard

Vous remarquerez que :

1. j'associe `factor(month)`, et non simplement `month`, à l'axe des x afin d'avoir, sur l'axe des abscisses, des chiffres cohérents allant de 1 à 12, et non des chiffres à virgule.

2. l'argument `group = 1` doit être ajouté pour que la ligne reliant les points apparaisse. En effet, les lignes sont censées relier des points qui appartiennent à une même série temporelle. Or ici, nous avons transformé `month` en facteur. Préciser `group = 1` permet d'indiquer à `geom_line()` que toutes les catégories du facteur `month` appartiennent au même groupe, que ce facteur peut être considéré comme une variable continue, et qu'il est donc correct de relier les points.
3. la fonction `geom_errorbar()` contient de nouvelles caractéristiques esthétiques qu'il nous faut obligatoirement renseigner : les extrémités inférieures et supérieures des barres d'erreur. Il nous faut donc associer 2 variables à ces caractéristiques esthétiques. Ici, nous utilisons `moyenne - erreur_std` pour la borne inférieure des barres d'erreur, et `moyenne + erreur_std` pour la borne supérieure. Les variables `moyenne` et `erreur_standard` faisant partie du tableau `temperatures_se`, `geom_errorbar()` les trouve sans difficulté.
4. l'argument `width` de la fonction `geom_errorbar()` permet d'indiquer la longueur des segments horizontaux qui apparaissent à chaque extrémité des barres d'erreur.

Ici, bien que moins lisible, on peut aussi faire apparaître les trois courbes sur le même graphique, afin de mieux visualiser les similarités des fluctuations de températures entre les 3 aéroports :

```
temperatures_se |>
  ggplot(aes(x = factor(month), y = moyenne, color = origin, group = origin)) +
  geom_line() +
  geom_point() +
  geom_errorbar(aes(ymin = moyenne - erreur_standard,
                    ymax = moyenne + erreur_standard),
                width = 0.1) +
  labs(x = "Mois",
       y = "Moyenne des températures quotidiennes maximales (°C)",
       color = "Aéroport") +
  theme_bw()
```

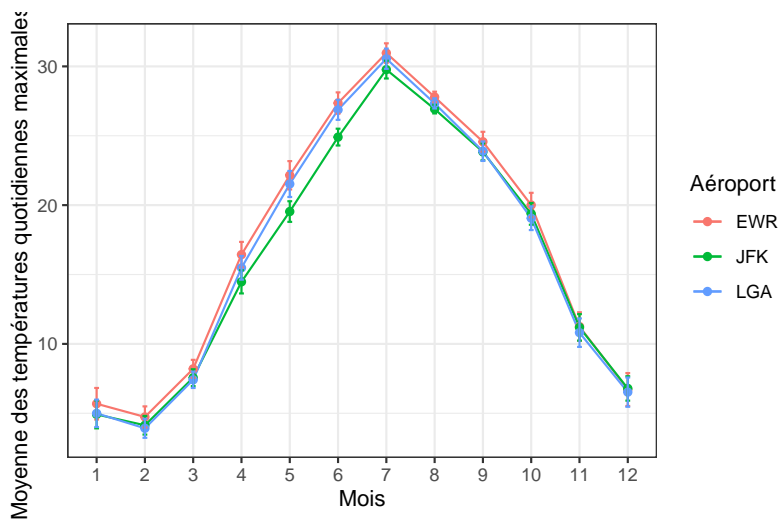


Figure 3.2: Températures moyennes mensuelles observées en 2013 dans les 3 aéroports de New York. Les barres d'erreur sont les erreurs standard.

Nous pouvons arriver au même résultats en utilisant le tableau `temperature_se_bornes`, qui contient des variables différentes :

```
temperature_se_bornes |>
  ggplot(aes(x = factor(month), y = moyenne, color = origin, group = origin)) +
  geom_line() +
  geom_point() +
  geom_errorbar(aes(ymin = moyenne_moins_se,
                    ymax = moyenne_plus_se,
                    width = 0.1)) +
  labs(x = "Mois",
       y = "Moyenne des températures quotidiennes maximales (°C)",
       color = "Aéroport") +
  theme_bw()
```

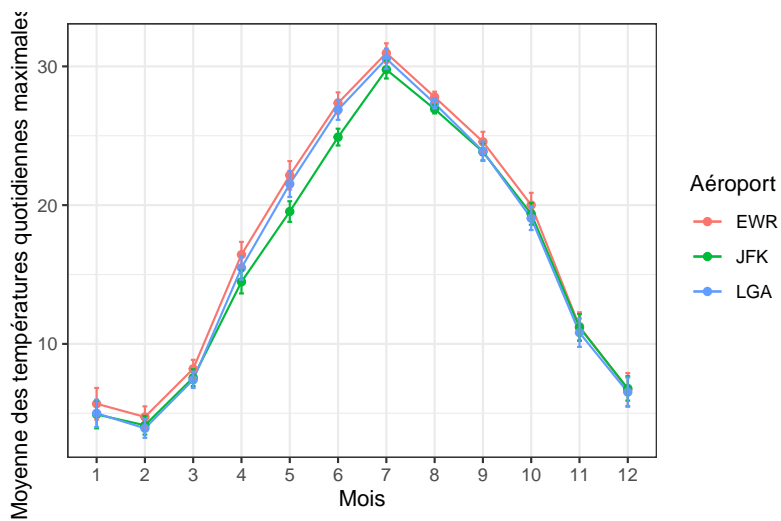


Figure 3.3: Températures moyennes mensuelles observées en 2013 dans les 3 aéroports de New York. Les barres d'erreur sont les erreurs standard.

De la même façon, nous pouvons parfaitement faire apparaître, au lieu des erreurs standards, les intervalles de confiance à 95% de chaque valeur de température moyenne. Il nous suffit pour cela d'utiliser le tableau `temperatures_ci` qui contient les valeurs de moyennes et des bornes supérieures et inférieures de ces intervalles :

```
temperature_ci |>
  ggplot(aes(x = factor(month), y = moyenne, group = 1)) +
  geom_line() +
  geom_point() +
  geom_errorbar(aes(ymin = ci_borne_inf, ymax = ci_borne_sup), width = 0.1) +
  facet_wrap(~origin, ncol = 1) +
  labs(x = "Mois",
       y = "Moyenne des températures quotidiennes maximales (°C)",
       color = "Aéroport") +
  theme_bw()
```

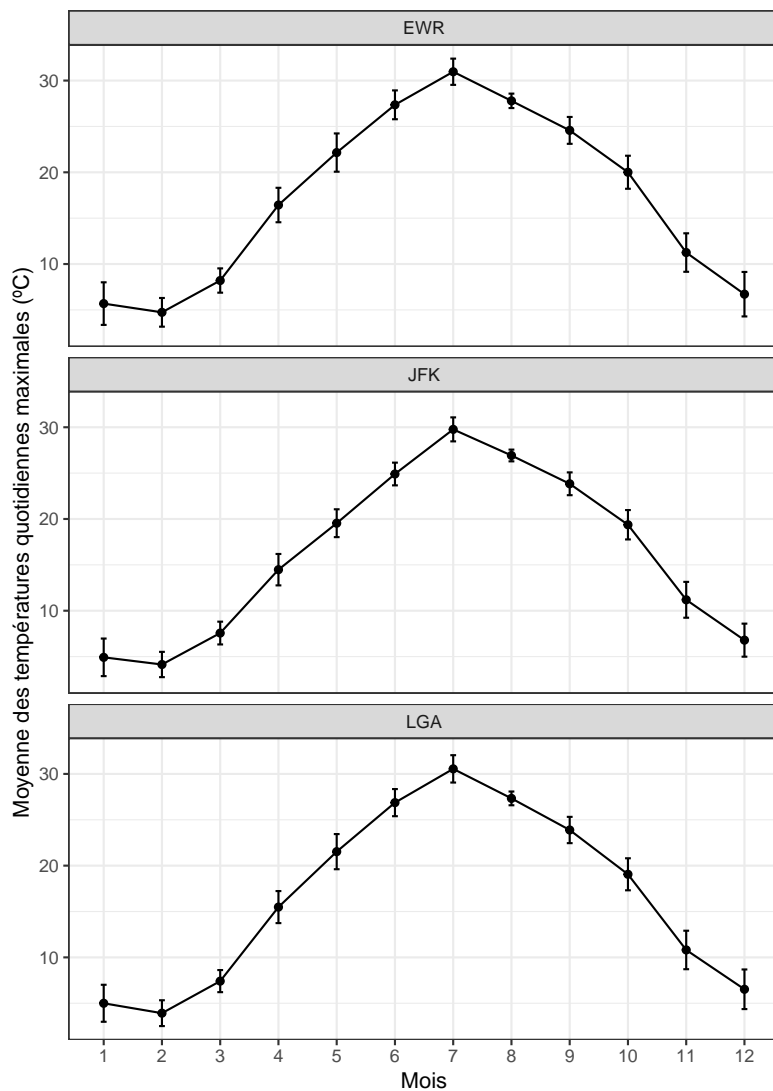


Figure 3.4: Températures moyennes mensuelles observées en 2013 dans les 3 aéroports de New York. Les barres d'erreur sont les intervalles de confiance à 95% des moyennes mensuelles.

Comme vous voyez, les barres d'erreurs sont maintenant plus longues que sur la Figure 3.1. C'est normal car rappelez-vous que les intervalles de confiance sont à peu près équivalents à 2 fois les erreurs standards. L'intérêt de représenter les intervalles de confiance est qu'ils sont directement liés aux tests statistiques que nous aborderons en L3. Globalement, quand 2 séries de données ont des intervalles de confiance

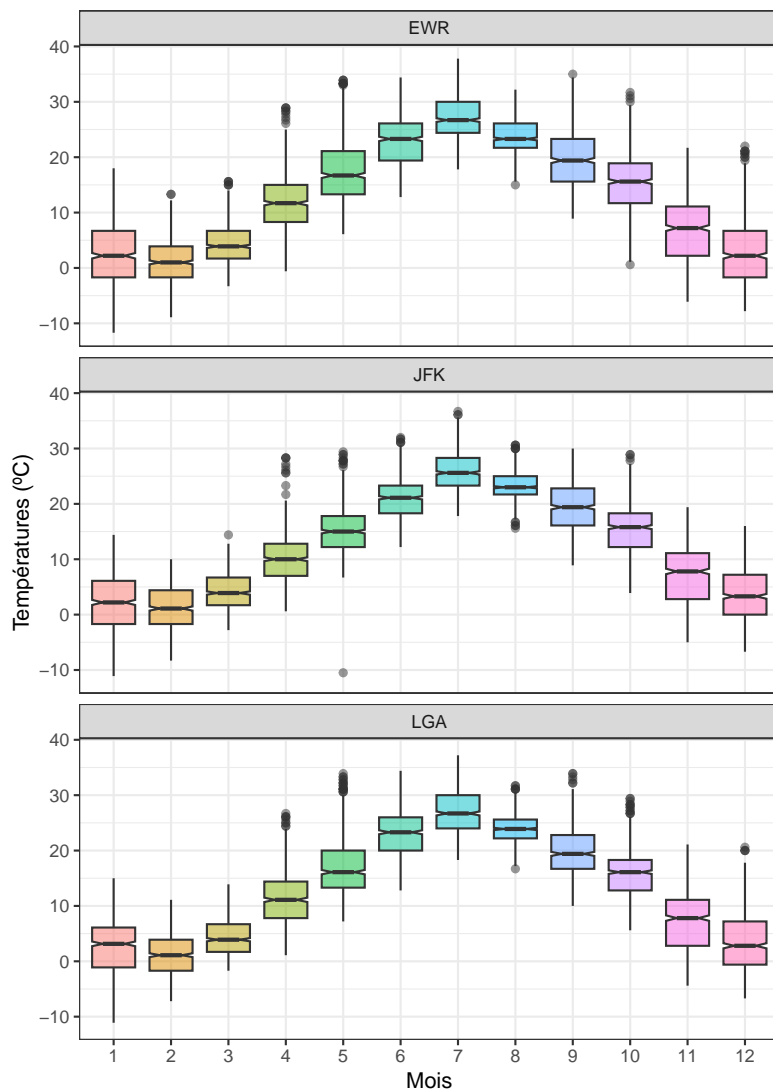
qui se chevauchent largement (comme les mois de janvier et février par exemple), alors, un test d'hypothèses conclurait presque toujours à l'absence de différence significative entre les 2 groupes. À l'inverse, quand 2 séries de données ont des intervalles de confiance qui ne se chevauchent pas du tout (comme les mois de mars et d'avril par exemple), alors, un test d'hypothèses conclurait presque toujours à l'existence d'une différence significative entre les 2 groupes. Lorsque les intervalles de confiance entre 2 catégories se chevauchent faiblement ou partiellement (comme entre les mois de juin et juillet pour l'aéroport LGA), la situation est moins tranchée, et nous devons nous en remettre aux résultats du test pour savoir si la différence observée devrait être considérée comme significative ou non.

3.6 Visualiser l'incertitude : les boîtes à moustaches

Outre les informations de position et de dispersion, les boîtes à moustaches permettent également de visualiser l'incertitude associée aux médianes. Il suffit pour cela d'ajouter l'argument `notch = TRUE` dans la fonction `geom_boxplot()` :

```
weather |>
  mutate(temp_celsius = (temp - 32) / 1.8) |>
  ggplot(aes(x = factor(month), y = temp_celsius, fill = factor(month))) +
  geom_boxplot(show.legend = FALSE, alpha = 0.5, notch = TRUE) +
  facet_wrap(~ origin, ncol = 1) +
  labs(x = "Mois", y = "Températures (°C)") +
  theme_bw()
```

Warning: Removed 1 row containing non-finite outside the scale range (``stat_boxplot()``).



Des encoches ont été ajoutées autour de la médiane de chaque boîte à moustache. Ces encoches sont des encoches d'incertitudes. Les limites inférieures et supérieures de ces encoches correspondent aux bornes inférieures et supérieures de l'intervalle de confiance à 95% des médianes. Comme pour les moyennes, le chevauchement ou l'absence de chevauchement entre les encoches de 2 séries de données nous renseigne sur l'issue probable des futurs tests statistiques que nous serions amenés à réaliser. Il sera donc important de bien examiner ces encoches en amont des tests statistiques pour éviter de faire/dire des bêtises...

3.7 Exercice

1. Avec le tableau `penguins`, calculez les grandeurs suivantes pour chaque espèce de manchot et chaque sexe :

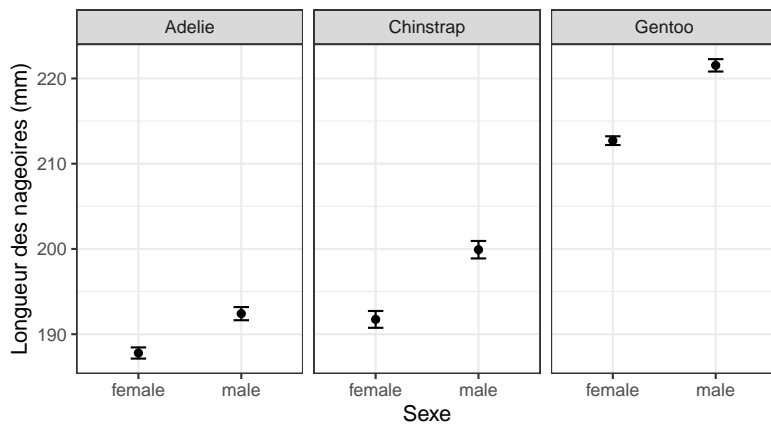
- la moyenne de la longueur des nageoires
- la variance de la longueur des nageoires
- l'écart-type de la longueur des nageoires
- l'erreur standard de la longueur moyenne des nageoires
- la moyenne de la masse corporelle
- la variance de la masse corporelle
- l'écart-type de la masse corporelle
- l'erreur standard de la masse corporelle moyenne

Attention : pensez à retirer les individus dont le sexe est inconnu.

2. Vérifiez avec la fonction `skim()` que les moyennes et écart-types calculés ci-dessus sont corrects.

3. Avec ces données synthétiques faites le graphique suivant :

Moyennes (et erreurs standard) des longueurs de nageoires chez les mâles et les femelles de trois espèces de manchots



References

- Horst, Allison, Alison Hill, et Kristen Gorman. 2022. *palmer-penguins: Palmer Archipelago (Antarctica) Penguin Data*. <https://CRAN.R-project.org/package=palmerpenguins>.
- Waring, Elin, Michael Quinn, Amelia McNamara, Eduardo Arino de la Rubia, Hao Zhu, et Shannon Ellis. 2022. *skimr: Compact and Flexible Summaries of Data*. <https://CRAN.R-project.org/package=skimr>.
- Wickham, Hadley. 2021. *nycflights13: Flights that Departed NYC in 2013*. <https://github.com/hadley/nycflights13>.
- . 2022. *tidyverse: Easily Install and Load the Tidyverse*. <https://CRAN.R-project.org/package=tidyverse>.
- Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, et Dewey Dunnington. 2022. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. <https://CRAN.R-project.org/package=ggplot2>.
- Wickham, Hadley, Romain François, Lionel Henry, Kirill Müller, et Davis Vaughan. 2023. *dplyr: A Grammar of Data Manipulation*. <https://CRAN.R-project.org/package=dplyr>.
- Wickham, Hadley, Jim Hester, et Jennifer Bryan. 2022. *readr: Read Rectangular Text Data*. <https://CRAN.R-project.org/package=readr>.