Curriculum Vitae

0.1 Please visit my properly formatted CV: ht tps://nineluijendijk.github.io/

Contact

LinkedIn: Nine Luijendijk
GitHub: nineluijendijk Email: nineluijendijk@gmail.com

Education

Hogeschool Utrecht
Bachelor of Science - BS, Life Sciences, Sep. 2020 - Aug. 2024
Specialized in Data Sciences for Biology
GPA of 4.0

Radboud University
Minor in Adaptive Organisms, Jan. 2023 - Jul. 2023

Experience

Netherlands Institute of Ecology (NIOO-KNAW) - Wageningen
Intern, Sep. 2023 - present

Zeeman textielSupers - Leusden
Sales Associate, Nov. 2021 - Jul. 2023

StudyWorks BV - Leusden
Tutor, Jan. 2020 - Sep. 2021

Tutored high school students in English, math and chemistry.

Albert Heijn - Leusden
Sales Associate, Jan. 2020 - Jul. 2020

Skills

R programming language
Bash command language
SQL
Excel
CSS stylesheet language
(Cell) culture
RNA-Seq analysis

Languages

Dutch - native speaker
English - C1

Chapter 1

Guerilla Analytics framework

To keep my data manageable I use the Guerilla Analytics framework (Ridge 2014). This means I make sure every project has its own folder, every folder that needs it has a README and no raw data file is altered. I also make sure to version control my code using GitHub.

The way I manage my data is visible in the tree below:

```
dir_tree("/Users/nineluijendijk/Desktop/daur2")
```

```
knitr::include_graphics(here("data/screenshot_dirtree.png"))
```

```
> dir_tree("/Users/nineluijendijk/Desktop/daur2")
/Users/nineluijendijk/Desktop/daur2
├── daur2.Rproj
├── metagenomics_formatieve_opdracht
|   └── metagenomics_formatieveopdracht.Rmd
├── metagenomics_lessen
|   └── Metagenomics.Rmd
├── rna_seq_eindopdracht
|   ├── Daur2_eindopdracht.Rmd
|   ├── Daur2_eindopdracht.html
|   ├── data
|   |   ├── README.txt
|   |   └── SRR7866699_1_screenshot.png
|   ├── data_raw
|   |   ├── README.txt
|   |   ├── SRR7866699_1_fastqc.html
|   |   ├── SRR7866699_2_fastqc.html
|   |   ├── SRR7866700_1_fastqc.html
|   |   ├── SRR7866700_2_fastqc.html
|   |   ├── SRR7866705_1_fastqc.html
|   |   ├── SRR7866705_2_fastqc.html
|   |   ├── SRR7866706_1_fastqc.html
|   |   └── SRR7866706_2_fastqc.html
|   ├── docs
|   |   ├── EindopdrachtDAUR2_V6.pdf
|   |   └── README.txt
|   └── eindopdracht.R
├── rna_seq_formatieve_opdracht
|   ├── Formatieve_opdracht1.R
|   ├── Formatieve_opdracht_RNA-seq.Rmd
|   ├── Formatieve_opdracht_RNA-seq.html
|   ├── data
|   ├── data_raw
|   |   ├── README.txt
|   |   ├── SRR7866687_1_fastqc.html
|   |   ├── SRR7866687_2_fastqc.html
|   |   ├── SRR7866688_1_fastqc.html
|   |   ├── SRR7866688_2_fastqc.html
|   |   ├── SRR7866689_1_fastqc.html
|   |   ├── SRR7866689_2_fastqc.html
|   |   ├── SRR7866690_1_fastqc.html
|   |   ├── SRR7866690_2_fastqc.html
|   |   ├── SRR7866691_1_fastqc.html
|   |   ├── SRR7866691_2_fastqc.html
|   |   ├── SRR7866692_1_fastqc.html
|   |   ├── SRR7866692_2_fastqc.html
|   |   ├── SRR7866693_1_fastqc.html
|   |   ├── SRR7866693_2_fastqc.html
|   |   ├── SRR7866694_1_fastqc.html
|   |   ├── SRR7866694_2_fastqc.html
|   |   └── alignment_statistics.rds
|   └── scripts
|       ├── README.txt
|       ├── script_htmlopener.sh
|       ├── scriptfastqdownload.sh
|       └── scriptrsubread.sh
└── rna_seq_lessen
    ├── data
    ├── data_raw
    |   ├── README.txt
    |   ├── SRR1039521_1_fastqc.html
    |   ├── alignment_statistics.rds
    |   └── read_counts.rds
    ├── les_1.R
    ├── les_2.R
    ├── les_3.R
    ├── les_4.R
    └── scripts
        ├── README.txt
        └── scriptfastqdump.sh
```

Figure 1.1: Screenshot of my directory tree

Chapter 2

Reproducible Research

2.1 Scoring the reproducibility of a research article

2.1.1 Reproducible research and criteria

In this chapter I will be scoring a scientific publication on how reproducible the research is. The criteria for reproducibility can be found in table 2.1. Keeping research reproducible and openly available enables researchers to work together or in parallel towards the same goal. (Sumner et al. 2020)

Generating the table

```
dataframe_criteria <- data.frame(TransparencyCriteria = c("Study Purpose", "Data Availability Sta
           Definition = c("A concise statement in the introduction of the article, often in the l
           ResponseType = c("Binary", "Binary", "Found Value", "Binary; Found Value", "Found Valu
```

Table containing the criteria of reproducibility

```
knitr::kable(dataframe_criteria, caption = "Table showing the criteria for reproducibilty from @s
```

2.1.2 The research article

The research article I will be scoring for reproducibility is Michalak et al. (2020), "Sounds of sickness: can people identify infectious disease using sounds of coughs and sneezes?". The study objective is to determine whether humans can identify infectious diseases by hearing. Multiple sound clips, for example of coughing or sneezing, were presented to participants in random order, half of the sounds

Table 2.1: Table showing the criteria for reproducibilty from @sumnerReproducibilityReportingPractices2020.

TransparencyCriteria	Definition
Study Purpose	A concise statement in the introduction of the article, often in the last
Data Availability Statement	A statement, in an individual section offset from the main body of text
Data Location	Where the article's data can be accessed, either raw or processed.
Study Location	Author has stated in the methods section where the study took place o
Author Review	The professionalism of the contact information that the author has pro
Ethics Statement	A statement within the manuscript indicating any ethical concerns, inc
Funding Statement	A statement within the manuscript indicating whether or not the autho
Code Availability	Authors have shared access to the most updated code that they used in

Table 2.2: Table showing how the article scored on reproducibility.

TransparencyCriteria	Definition
Study Purpose	A concise statement in the introduction of the article, often in the last
Data Availability Statement	A statement, in an individual section offset from the main body of text
Data Location	Where the article's data can be accessed, either raw or processed.
Study Location	Author has stated in the methods section where the study took place o
Author Review	The professionalism of the contact information that the author has pro
Ethics Statement	A statement within the manuscript indicating any ethical concerns, inc
Funding Statement	A statement within the manuscript indicating whether or not the autho
Code Availability	Authors have shared access to the most updated code that they used in

being from a people with an infectious illness and half from people with a non-infectious condition. The participants were asked to determine the nature of these sounds (infectious or not) and how certain they were. This study was done 3 more times, with slight alterations, for example also asking the participants to score how disgusting a sound was. Using R to calculate statistics, it was found that people can not accurately determine whether a sneeze or cough is infectious, even when they were pretty certain of their judgements.

2.1.3 Reproducibility of the research

The reproducibility of this article is scored in table 2.2 below:

```
criteria_scored <- dataframe_criteria %>% mutate(Score = c("Present", "Present", "Prese

knitr::kable(criteria_scored, caption = "Table showing how the article scored on reproc
```

Getting a perfect score, this research seems to be very reproducible.

2.2 Scoring the reproducibility of the code of the research

2.2.1 Criteria for the code

Now I will be scoring how easy it is to reproduce one of the figures from the article, using the code provided by the authors. I will also be judging how easy it is to read the code.

2.2.2 The code

There is code available that was used to prepare the raw data for further analysis, but the raw data itself is not publically available. I assume this is to protect participants' identities. The preparation script comes with comments explaining what every step does and is easy to read. Since the researchers mainly use the package {tidyverse} (Hadley Wickham et al. 2019) their code is especially easy to read as I like to use it as well. Using the processed data, it is very easy to recreate the figures from this study. Below is part of the analysis script downloaded from the study's Open Science Framework (OSF) project (linked in table 2.2). I would rate the readabilty of this code a 5/5.

Install and/or load packages

```
library(tidyverse)
library(knitr)
library(psych)
```

Data

```
#Read data from preparation script
qualtrics1 <- read_tsv("data/study1/cleandata/qualtrics1.tsv")
lqualtrics1 <- read_tsv("data/study1/cleandata/lqualtrics1.tsv")
```

Multiple-item Psychological Measures

```
#Perceived Infectability
qualtrics1 %>%
  select(PVD8, PVD12r, PVD2, PVD14r, PVD10, PVD5r, PVD6) %>%
  omega(title = "Perceived Infectability")
```

```
#Scatterplot Matrix (ignoring stimuli variance)
lqualtrics1 %>%
  select(certainty, clarity, pvd_pinfect, pvd_germ, rchild_uncertain, rchild_ses, recentill) %>%
  pairs.panels(scale = FALSE, pch = ".")
```

This code didn't work immediately, there were 2 things I needed to change first. The first thing was the path to the data, since the file names on OSF didn't perfectly match those in the script. The directory also had to be changed, I used the {here} package Müller (n.d.) so that if I want to refer to the same file from another computer it will still work.

The {GPArotation} package (Coen A. Bernaards and Jennrich 2005) also was not loaded in, when it is needed for the `omega()` function. Besides these 2 things, everything worked fine. The code now looks as follows:

Install and/or load packages

```
library(tidyverse)
library(knitr)
library(psych)
library(here)
library(GPArotation)
```

Data

```
#Read data from preparation script
qualtrics1 <- read_tsv(here("data_raw/qualtrics1_share.tsv"))
lqualtrics1 <- read_tsv(here("data_raw/lqualtrics1_share.tsv"))
```

Multiple-item Psychological Measures

```
#Perceived Infectability
qualtrics1 %>%
  select(PVD8, PVD12r, PVD2, PVD14r, PVD10, PVD5r, PVD6) %>%
  omega(title = "Perceived Infectability")
```

Omega estimates results

```
## Perceived Infectability
## Call: omegah(m = m, nfactors = nfactors, fm = fm, key = key, flip = flip,
##     digits = digits, title = title, sl = sl, labels = labels,
##     plot = plot, n.obs = n.obs, rotate = rotate, Phi = Phi, option = option,
##     covar = covar)
## Alpha:                 0.91
## G.6:                   0.92
## Omega Hierarchical:    0.75
## Omega H asymptotic:    0.79
## Omega Total            0.94
##
## Schmid Leiman Factor loadings greater than  0.2
```

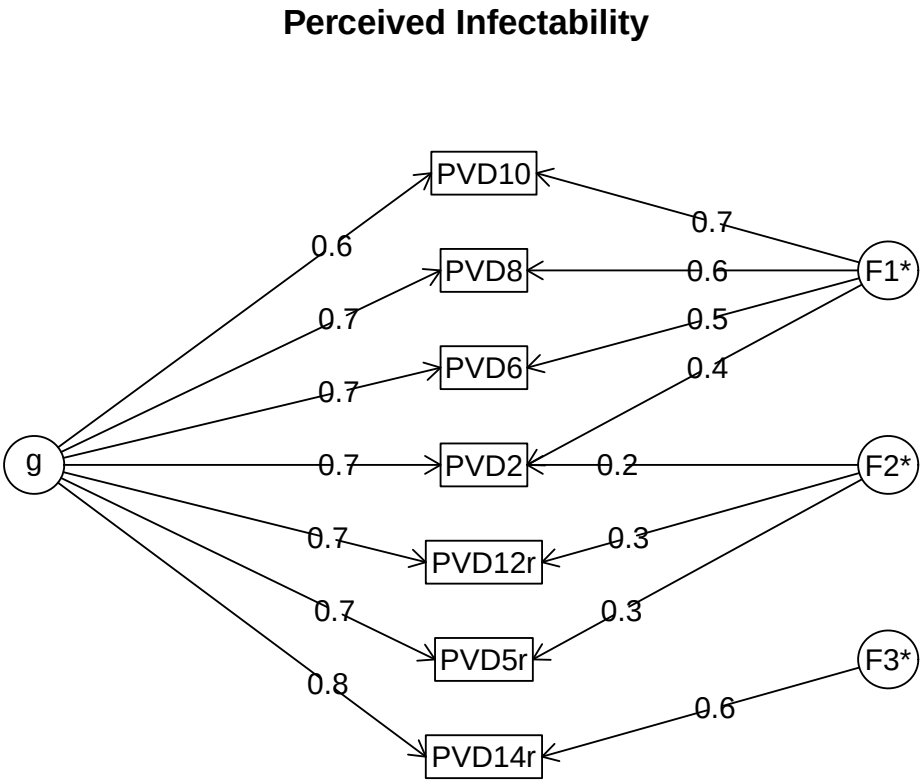

Figure 2.1: Plot showing McDonald's omega estimates (Michalak et al. 2020).

```
##           g   F1*   F2*   F3*   h2   u2   p2
## PVD8   0.68  0.60             0.83 0.17 0.56
## PVD12r 0.73        0.34       0.65 0.35 0.81
## PVD2   0.71  0.37  0.20       0.68 0.32 0.73
## PVD14r 0.77              0.57 0.91 0.09 0.65
## PVD10  0.62  0.67             0.83 0.17 0.47
## PVD5r  0.70        0.26       0.59 0.41 0.83
## PVD6   0.70  0.48             0.73 0.27 0.67
##
## With Sums of squares  of:
##    g  F1*  F2*  F3*
## 3.45 1.18 0.23 0.37
##
## general/max  2.94   max/min =   5.09
## mean percent general =  0.67    with sd =  0.13 and cv of  0.19
## Explained Common Variance of the general factor =  0.66
##
## The degrees of freedom are 3  and the fit is  0.01
## The number of observations was  157  with Chi Square =  1.68  with prob <  0.64
## The root mean square of the residuals is  0.01
## The df corrected root mean square of the residuals is  0.02
## RMSEA index =  0  and the 10 % confidence intervals are  0 0.108
## BIC =  -13.49
##
## Compare this with the adequacy of just a general factor and no group factors
## The degrees of freedom for just the general factor are 14  and the fit is  1.13
## The number of observations was  157  with Chi Square =  171.53  with prob <  3.4e-29
## The root mean square of the residuals is  0.16
## The df corrected root mean square of the residuals is  0.19
##
## RMSEA index =  0.268  and the 10 % confidence intervals are  0.233 0.305
## BIC =  100.74
##
## Measures of factor score adequacy
##                                                 g  F1*   F2*  F3*
## Correlation of scores with factors            0.89 0.81  0.52 0.74
## Multiple R square of scores with factors      0.79 0.66  0.27 0.54
## Minimum correlation of factor score estimates 0.57 0.33 -0.45 0.09
##
##  Total, General and Subset omega for each subset
##                                                 g  F1*  F2*  F3*
## Omega total for total scores and subscales    0.94 0.92 0.74 0.91
## Omega general for total scores and subscales  0.75 0.57 0.63 0.59
## Omega group for total scores and subscales    0.16 0.35 0.11 0.32
```

```
#Scatterplot Matrix (ignoring stimuli variance)
lqualtrics1 %>%
  select(certainty, clarity, pvd_pinfect, pvd_germ, rchild_uncertain, rchild_ses, recentill) %>%
  pairs.panels(scale = FALSE, pch = ".") %>% title(main = "Scatterplot Matrix (ignoring stimuli v
", line = 1.5)
```

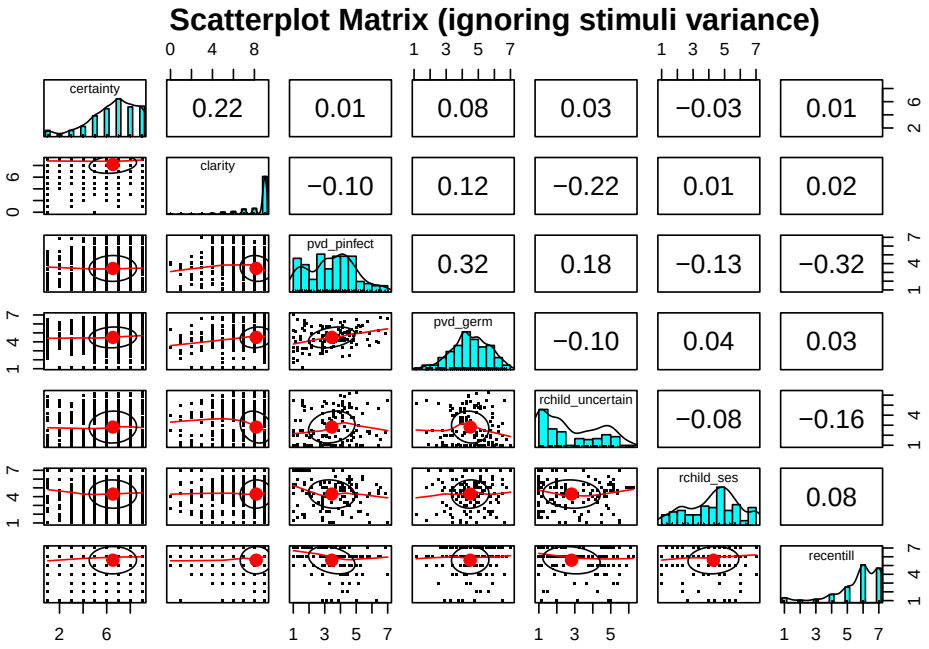

Figure 2.2: Scatterplot Matrix ignoring stimula variance (Michalak et al. 2020).

I decided to also add a title to the scatter plot matrix (figure 2.2), since it was missing in the original code.

Overall, it was not hard to generate these two figures using the authors' script. I'd rate the effort it took to recreate the figures a 4.5/5, 5 being very easy. I immediately saw what had to be changed before even running the script and this only took a minute to do. The {GPArotation} package wasn't in their list of libraries to load since it's loaded automatically when running the `omega()` function. This doesn't work when the package is not already installed, and running the code just produces an error.

2.3 Final rating

I think this research article deserves a 9.5/10 for reproducibility. Chapters in the article describing ethics and data accessibility have their own headers so they're easy to find in the table of contents, in the text itself every claim has a source,

the data and scripts were one click away from the article itself and really the only thing making it not 10/10 reproducible was the absence of the raw data, which I assume is to protect participants' privacy.

Chapter 3

Analyzing and visualizing data

The data that is to be visualized comes from an experiment involving C. elegans nematodes and was supplied by J. Louter of the INT/ILC. In this experiment, the number of offspring of adult C. elegans was counted after exposing them to different substances in multiple concentrations.

3.1 Preparing the data

3.1.1 Inspecting the data in R

```
celegans_data <- read_excel(here("data_raw/CE.LIQ.FLOW.062_Tidydata.xlsx"))
```

Before the data can be visualized, it has to be cleaned up. The dataframe is already in tidy format, but the data types of the columns need to be changed. Expected data types would be double/integer for RawData (number of offspring), character for compName (name of the compound) and double for compConcentration (concentration of the compound). The actual data types of the columns are double for RawData, character for compName and character for compConcentration. The data type for compConcentration has not been correctly assigned which will cause the following issue if it's not changed:

```
#Create a scatter plot
plot_chr <- ggplot(data = celegans_data, aes(x = compConcentration, y = RawData))+
  geom_point(aes(color = compName,
                 shape = expType),
             size = 1)+
```

```
  labs(x = "Concentration",
       y = "Number of offspring",
       title = "Number of C. elegans offspring under\na number of circumstances, alphal
       shape = "Type",
       color = "Compound")+
  scale_colour_manual(values = c("red", "darkgoldenrod1", "green", "royalblue3", "viole
  theme_minimal()+
  theme(legend.key.size = unit(0.75,"line"),
        legend.text = element_text(size = 8),
        axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
plot_chr
```

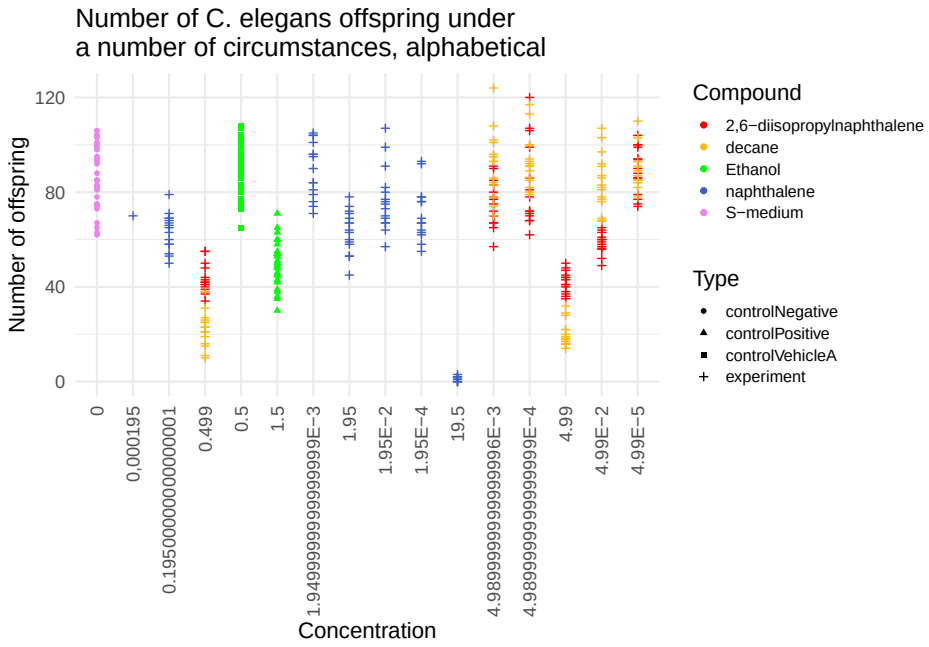

Figure 3.1: Number of C. elegans offspring under a number of circumstances where the x-axis is ordered alphabetically

The x-axis labels are ordered alphabetically because the data type of the compConcentration is character instead of double. R probably read it this way because Excel adds an E to showcase exponential values.

3.1.2 Fixing the character/double issue

Since not all compounds share the same compUnit (the unit in which their concentration is measured), 2 separate plots need to be made and then combined: one for the compounds measured in nM and one for the compounds measured in

percentage. After changing the data type from character to double and adding jitter to the plot to make it more readable it looks as follows:

```
#Change data type from chr to dbl
celegans_data$compConcentration <- as.double(celegans_data$compConcentration)

#Create a scatter plot for the concentration in nM
celegans_data_nM <- celegans_data %>% filter(compUnit == "nM")
plot_nM <- ggplot(data = celegans_data_nM, aes(x = log10(compConcentration), y = RawData))+
  geom_jitter(aes(color = compName, #add jitter
                  shape = expType),
              width = 0.5, height = 0.2)+
  labs(x = "log10 concentration (nM)",
       y = "Number of offspring")+
  coord_cartesian(ylim = c(0, 120))+
  scale_shape_manual(values = 3)+
  scale_colour_manual(values = c("red", "darkgoldenrod1", "royalblue3"))+
  theme_minimal()+
  theme(legend.position = "none")

#Create a scatter plot for the concentration in pct
celegans_data_pct <- celegans_data %>% filter(compUnit == "pct")
plot_pct <- ggplot(data = celegans_data_pct, aes(x = expType, y = RawData))+
  geom_jitter(aes(color = compName, #add jitter
                  shape = expType),
              size = 1.3,
              width = 0.1, height = 0.075)+
  xlab("Type")+
  coord_cartesian(ylim = c(23.7, 121))+
  scale_colour_manual(values = c("green", "violet"))+
  theme_minimal()+
  theme(legend.position = "none",
        axis.text.x=element_text(vjust=0.5, hjust=0.5, size = 8.75),
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.title.y = element_blank())+ #remove any y-axis labeling
  rotate_axis_labels("x", 90)

#obtain legend
legend <- get_legend(plot_chr)

#combine figures and legend for readability
ggdraw(plot_grid(plot_grid(plot_nM, plot_pct),
                 plot_grid(NULL, legend, ncol = 3),
                 rel_widths = c(1, 0.35)))+
  plot_annotation("Number of C. elegans offspring under\na number of circumstances")
```

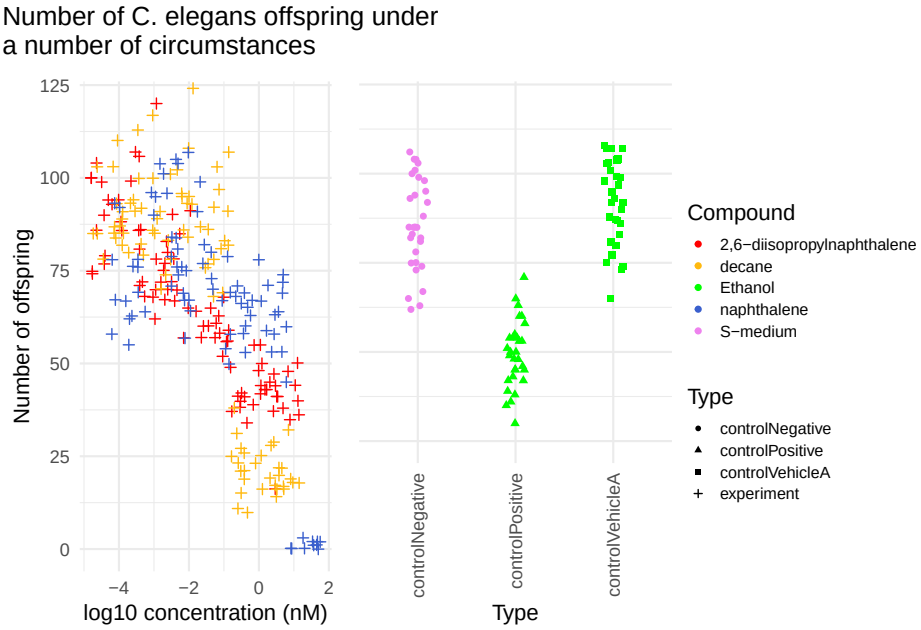

Figure 3.2: Total number of C. elegans offspring under a number of circumstances, with on the y-axis the number of offspring and on the x-axis the log10 concentration in nM.

The positive control for this experiment is ethanol. The negative control for this experiment is S-medium.

3.2 Analyzing and visualizing the data

3.2.1 Plan for IC50/ED50 visualization

Since this chapter is focused on visualization, there won't be a statistical analysis of the dose-response interactions. If I were to perform a statistical analysis, it would be done using the {drc} package (Ritz et al. 2015), which has functions to calculate IC50s and ED50s, which then could be compared. Using {ggplot2} (Hadley Wickham 2016), a scatter plot can be made with on the y-axis the number of offspring and on the x-axis the compound concentration. The calculated IC50/ED50 values could be added as lines over the plot, giving every compound (line) a different color. An ANOVA can also be conducted to look for differences between concentrations, after checking if the data comes from a normal distribution using the Shapiro-Wilk test.

3.2.2 Normalizing the data

To make is easier to read the results visualized in the plot, the number of offspring was normalized as a fraction of the negative control, meaning any value under 1 shows a reduction in the number of offspring. To normalize the data, the following code was used:

```
#Obtain the mean of the RawData (negativeControl)
controlNeg <- celegans_data %>% filter(compName == "S-medium") %>% summarize(mean = mean(RawData,

#Use the mean to calculate fractions
mutated <- celegans_data %>% filter(RawData > 0) %>%
  select(RawData, compName, compConcentration, expType) %>% na.omit() %>%
  mutate(normalized = RawData/controlNeg$mean)

#Obtain the means of the other 2 control groups
controlPos <- mutated %>% filter(expType == "controlPositive") %>% summarize(mean = mean(normaliz
controlVeh <- mutated %>% filter(expType == "controlVehicleA") %>% summarize(mean = mean(normaliz
```

3.2.3 Creating a legend

To give the final figure a proper legend, the legend needs to be generated in a separate figure since we don't want to include the test-compounds in it. The following code was used to generate a legend, making use of dummy data.

Generating a legend using dummy data

```
dummydf <- data.frame(a = rep(c(1, 2, 3), each=2), #create dummy dataframe for the dummy plot
                      b = c(1, 4, 6, 3, 4, 7),
```

```
                      Control = rep(c("controlNegative", "controlPositive", "controlVe

dummyplot <- ggplot(dummydf, aes(x=a, y=b, group=Control))+ #create dummy plot to obta
  geom_line(aes(linetype=Control, color=Control))+
  scale_linetype_manual(values=c("solid", "solid", "longdash"))+
  scale_color_manual(values=c("violet", "green", "green"))

dummylegend <- get_legend(dummyplot) #obtain dummy legend
```

3.2.4 Generating the normalized plot

With the data normalized and the legend obtained, the normalized plot can be generated.

```
#create normalized plot
plotfraction <- mutated %>% filter(compName == "2,6-diisopropylnaphthalene" | compName
  ggplot(aes(x = log10(compConcentration), y = normalized))+
  geom_jitter(aes(color = compName), width = 0.5, height = 0.1)+
  scale_colour_manual(values = c("red", "darkgoldenrod1",  "royalblue3"))+
  coord_cartesian(ylim = c(0, 1.5))+
  labs(x = "log10 concentration (nM)",
       y = "Normalized number of offspring",
       title = "Number of C. elegans offspring as a fraction\nof the negative control g
       color = "Compound")+
  geom_hline(yintercept = 1, color = "violet")+
  geom_hline(yintercept = controlPos$mean, color = "green")+
  geom_hline(yintercept = controlVeh$mean, color = "green", linetype = "longdash")+
  theme_minimal()+
  guides(color = "none")+
  facet_wrap(~ compName)

#combine figure and legend
ggdraw(plot_grid(plotfraction, dummylegend,
       rel_widths = c(5, 1)))
```

This way it's easier to read the graph. Everything below the pink line means less offspring than control C. elegans and everything above it means more. The figure (3.3) shows a decrease in the number of offspring for all 3 groups.

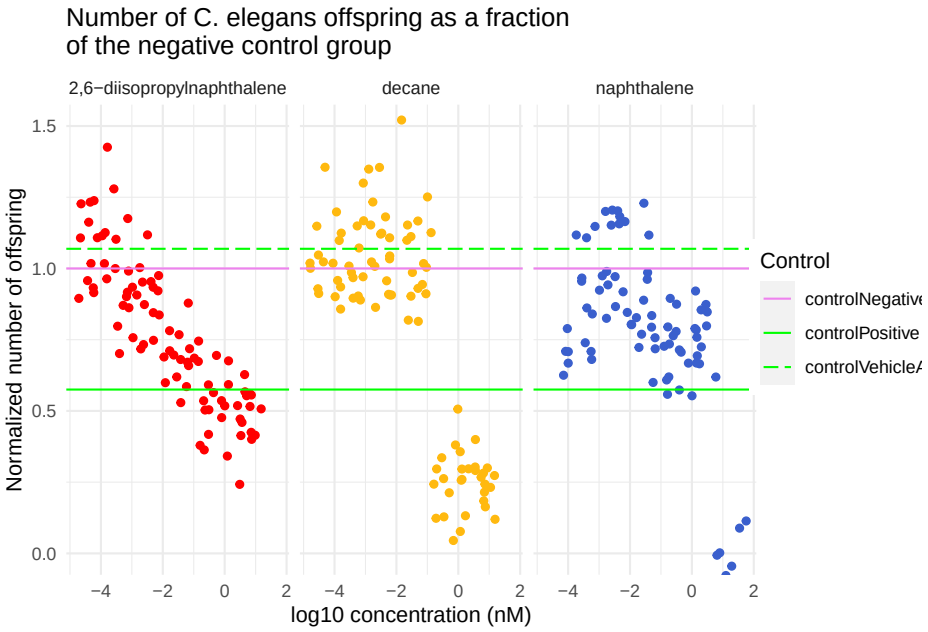

Figure 3.3: Number of C. elegans offspring as a fraction of the negative control group, with on the y-axis the normalized number of offspring and on the x-axis the log10 concentration of the compounds in nM.

Chapter 4

Drug Screening

Every year, 400,000 children and young people are diagnosed with cancer (Lewandowska et al. 2021). In developed countries, over 80% of children survive for at least 5 years after diagnosis (Schüz and Roman 2021), and surival rates keep going up (SteliarovaFoucher et al. 2004) as seen in figure 4.1. This does mean that around 20% of children don't. Researchers all over the world are working on bringing this percentage down even more, including the doctors and researchers at the Princess Máxima Center in Utrecht.

```
knitr::include_graphics(here("data/cancerstatistics.png"))
```

Princess Máxima Center is a children's cancer research hospital, where all children diagnosed with cancer in the Netherlands are treated (Prinses Máxima Centrum n.d.). A core focus of the Princess Máxima Center is combining treatment with research and vice versa, which how new methods for treatment are found.

When treating cancer, it's important to use a drug that both kills the tumor and has the lowest number of side effects. Specialized cancer hospitals (like the Princess Máxima Center) run tests to determine which drug this would be, using a high throughput facility that can screen hundreds of different drugs used on patient material at once ("High-Throughput Screening (HTS)" n.d.). In theory, this could be done for every patient, meaning they could be treated with the most suitable drug for their type of cancer.

Where the pipeline for single-drug testing is established, there is no set way to analyse the data obtained from a combination-drug screening. When doing a combination-drug screening, multiple drugs are combined. Researchers hope to find combinations of drugs that have a greater effect in combination with each other than when their separate effects are added together, see figure 4.2. Synergy, as this is called, is desired, so patients can experience the same positive

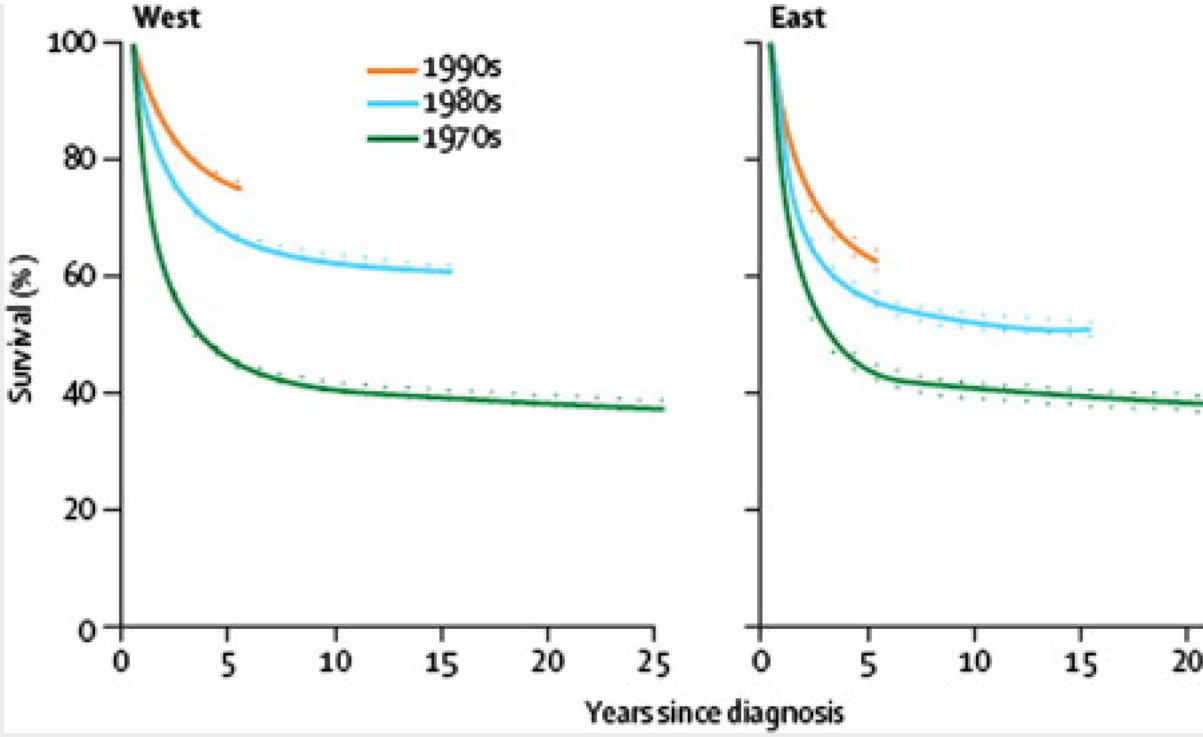

Figure 4.1: Cancer survival rates for children in West and East Europe [@steliarovafoucherGeographicalPatternsTime2004b].

effects but less side effects from the drugs (Cokol 2012).

```
knitr::include_graphics(here("data/synergy_antagonism.png"))
```

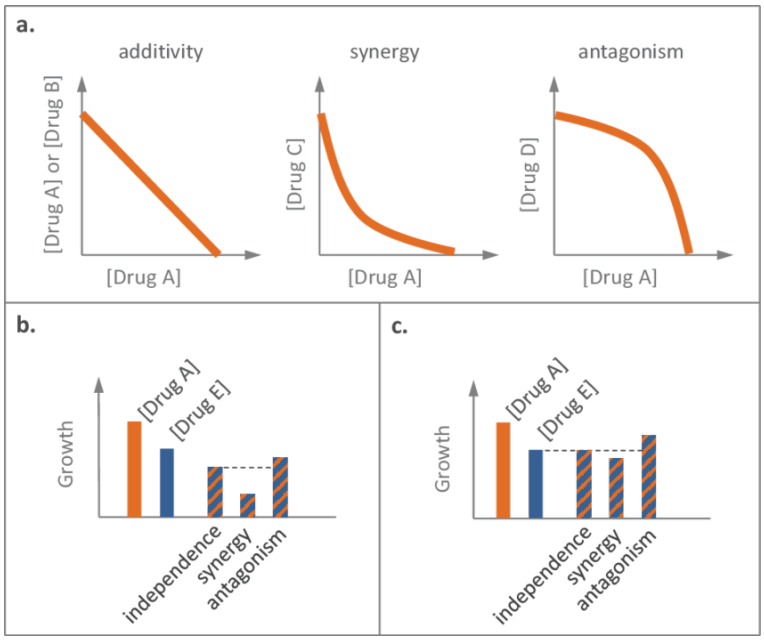

Figure 4.2: Different types of interactions between drugs [@cokolDrugsTheirInteractions2012].

However, analyzing combination-screen data is more complex than analyzing single-screen data (Rounsaville, Petry, and Carroll 2003). It is up to me and 2 of my fellow students to extend the existing pipeline to make is simpler for researchers to visualize and interpret their data from combination-screens, focusing on the different types of synergy (Jaaks et al. 2022). The R package {SynergyFinder} (Zheng et al. 2022) contains a multitude of functions to visualize dose-response and synergy data and will be used in the package that we are going to create ourselves.

The main focus will be on creating a function (or multiple functions that go together) that can filter the data from the high throughput facility for the "best" drug combinations, after the data has been molded to fit into the {SynergyFinder} functions. It is important that every parameter can be set manually instead of being hard-coded into the function, so that the function can be used to help answer any research question, instead of being specific for one single situation.

With this package will come a `DESCRIPTION` file containing metadata (like the package title and version for example), a `NAMESPACE` file that ensures R will be able to find every function written, a vignette (guide) and of course all

the functions we think are necessary for the data analysis of combination-drug screening data.

Chapter 5

Working with relational databases

Having spent so much time analyzing data, it would be useful to learn more about obtaining data from databases, and how to manipulate and analyze relational data using SQL.

5.1 Preparing the data

5.1.1 Introduction

In this report I will be analyzing 2 datasets obtained from Google Dengue Trends and Google Flu Trends. Sadly, the metadata is incomplete and I am not sure what the numbers mean, it's not documented. I will be assuming that the datasets contain the number of dengue and flu cases per week of the year, from 2002 through 2015. This shows just how important complete metadata is.

5.1.2 Importing the data and making it tidy

The first step in the data analysis is to import the data into Rstudio and make it tidy, meaning each variable gets its own column, each observation gets its own row and each value its own cell. In this case neither of the datasets were in tidy format, as there were multiple observations in each row. After making the data tidy, the class of every column was checked and changed to match the class of that column in the gapminder ({dslabs}, (Irizarry and Gill 2021)) dataframe, so they can later be joined together. A "year" column was added to both the dengue and the flu data as well, to match the "year" column of gapminder.

```
dengue_data <- read_csv(here("data_raw/dengue_data.csv"), skip = 11) #first 11 rows co
flu_data <- read_csv(here("data_raw/flu_data.csv"), skip = 11) #first 11 rows contain

dengue_tidy <- pivot_longer(data = dengue_data,
                            cols = c(2:ncol(dengue_data)),
                            names_to = "country",
                            values_to = "cases") #each observation must have its own r

flu_tidy <- pivot_longer(data = flu_data,
                         cols = c(2:ncol(flu_data)),
                         names_to = "country",
                         values_to = "cases") #each observation must have its own row

dengue_tidy <- dengue_tidy %>% mutate("year" = substr(Date, 1, 4)) #add "year" column
dengue_tidy$year <- as.integer(dengue_tidy$year) #change class of column year from chr

flu_tidy <- flu_tidy %>% mutate("year" = substr(Date, 1, 4)) #add "year" column
flu_tidy$year <- as.integer(flu_tidy$year) #change class of column year from chr to in

dengue_tidy$country <- as.factor(dengue_tidy$country) #the column country in gapminder
flu_tidy$country <- as.factor(flu_tidy$country) #the column country in gapminder is of
```

The tidy dataframes were then saved as separate rds and csv files:

```
saveRDS(gapminder, here("data/gapminder.rds")) #store the three tables as .rds and .cs
saveRDS(dengue_tidy, here("data/dengue.rds"))
saveRDS(flu_tidy, here("data/flu.rds"))
write_csv(gapminder, here("data/gapminder.csv"))
write_csv(dengue_tidy, here("data/dengue.csv"))
write_csv(flu_tidy, here("data/flu.csv"))
```

5.1.3 Connecting to the database

For this analysis, a PostgreSQL database was created in Dbeaver. To connect to the database in Rstudio, the {RPostgres} (H. Wickham, Ooms, and Müller 2022) package was used.

```
con <- dbConnect(Postgres(), #connect to the database
                 dbname = "workflowsdb",
                 host="localhost",
                 port="5432",
                 user="postgres",
                 password="password")
```

Table 5.1: 0 records

country	year	infant_mortality	life_expectancy	fertility	population	gdp	continent	region

Table 5.2: 0 records

country	year	infant_mortality	life_expectancy	fertility	population	gdp	continent	region

5.1.4 Exporting the tidy dataframes to the database

```
dbWriteTable(con, "gapminder", gapminder) #insert the tables into the database
dbWriteTable(con, "dengue", dengue_tidy)
dbWriteTable(con, "flu", flu_tidy)
```

5.2 Inspecting the data

5.2.1 Inspecting the data using SQL

To make sure no important data is missing, the tables were checked for NULL values. Data I consider important is the year and the country data, since if there are any missing values here errors may occur.

Checking for NULL values

```
SELECT *
  FROM gapminder
WHERE year IS NULL; --make sure no important values are missing
```

```
SELECT *
  FROM gapminder
WHERE country IS NULL;
```

```
SELECT *
  FROM dengue
WHERE year IS NULL;
```

```
SELECT *
  FROM dengue
WHERE country IS NULL;
```

Table 5.3: 0 records

Date	country	cases	year

Table 5.4: 0 records

Date	country	cases	year

Table 5.5: 0 records

Date	country	cases	year

```
SELECT *
  FROM flu
WHERE year IS NULL;
```

```
SELECT *
  FROM flu
WHERE country IS NULL;
```

There were no records for any of these selects, meaning there are no NULL values anywhere in “year” or “country”.

Now to actually look at the data, there were a couple of statistics I was interested in, like:

How many observations were done

```
SELECT COUNT(cases) --count the number of observations (ignoring NULL)
  FROM dengue;
```

```
SELECT COUNT(cases)
  FROM flu;
```

The total number of cases

```
SELECT SUM(cases) --calculate the total number of cases
  FROM dengue;
```

```
SELECT SUM(cases)
  FROM flu;
```

The countries with the highest number of cases

```
SELECT year, country, cases*1000 AS cases1000 --look at the highest observed numbers o
  FROM dengue
WHERE cases IS NOT NULL
```

Table 5.6: 0 records

Date	country	cases	year

Table 5.7: 1 records

count
6263

Table 5.8: 1 records

count
17266

```
ORDER BY cases1000 DESC;
```

```
SELECT year, country, cases
  FROM flu
WHERE cases IS NOT NULL
ORDER BY cases DESC;
```

5.2.2 Inspecting the data using R

Now it's time to inspect the data in R. When importing the dengue data from its csv, I noticed every value in the dataframe was a fraction instead of a whole number. Since no Readme was provided with the data, I am not sure why this is. To make the values whole, I multiplied the number of cases by 1000.

To find out which countries had the highest and lowest observed number of cases, the following code was used:

```
dengue_mutated <- dengue_tidy %>% mutate("cases1000" = cases * 1000) #add a column with the multi

maxcasesdengue <- subset(dengue_mutated, cases1000 == max(na.omit(dengue_mutated$cases1000)))
mincasesdengue <- subset(dengue_mutated, cases1000 == min(na.omit(dengue_mutated$cases1000)))
unique(maxcasesdengue$country) #look at which countries have the highest observed number of dengu
```

```
##  [1] Indonesia   Singapore   Brazil      Bolivia     Argentina   Mexico      Venezuela   Phili
## [10] Thailand
## Levels: Argentina Bolivia Brazil India Indonesia Mexico Philippines Singapore Thailand Venezue
```

```
unique(mincasesdengue$country) #look at which countries have the lowest observed number of dengue
```

```
## [1] India   Bolivia
```

Table 5.9: 1 records

sum
870.376

Table 5.10: 1 records

sum
8179518

Table 5.11: Displaying records 1 - 10

year	country	cases1000
2008	Brazil	1000
2004	Indonesia	1000
2012	India	1000
2009	Mexico	1000
2010	Philippines	1000
2009	Bolivia	1000
2013	Thailand	1000
2009	Argentina	1000
2010	Venezuela	1000
2005	Singapore	1000

Table 5.12: Displaying records 1 - 10

year	country	cases
2013	United States	10555
2013	United States	10112
2009	Canada	9688
2009	Canada	9630
2013	United States	9408
2012	United States	8618
2003	United States	8196
2012	United States	7904
2013	Canada	7698
2003	Canada	7531

Table 5.13: (#tab:dengue dataframemeans)Table containing dengue numbers per country

country	total	mean
Venezuela	180893	277.44325
Thailand	66761	184.93352
Indonesia	97124	147.38088
Singapore	94747	143.77390
Brazil	93638	142.09105
Mexico	92989	141.53577
Philippines	88106	136.38700
India	76889	116.67527
Argentina	49779	76.34816
Bolivia	29450	44.68892

```
## Levels: Argentina Bolivia Brazil India Indonesia Mexico Philippines Singapore Thailand Venezue
```

```
maxcasesflu <- subset(flu_tidy, cases == max(na.omit(flu_tidy$cases)))
mincasesflu <- subset(flu_tidy, cases == min(na.omit(flu_tidy$cases)))
unique(maxcasesflu$country) #look at which countries have the highest observed number of flu case
```

```
## [1] United States
## 29 Levels: Argentina Australia Austria Belgium Bolivia Brazil Bulgaria Canada Chile France Ger
```

```
unique(mincasesflu$country) #look at which countries have the lowest observed number of flu cases
```

```
## [1] Chile  Sweden
## 29 Levels: Argentina Australia Austria Belgium Bolivia Brazil Bulgaria Canada Chile France Ger
```

Another statistic is the total number and the average number of cases per country of all years combined.

```
dengue_country_means <- dengue_mutated %>% group_by(country) %>%
  summarize(total = sum(cases1000, na.rm = TRUE), mean = mean(cases1000, na.rm = TRUE)) %>%
  arrange(desc(mean)) #calculate the total number of cases and mean number of cases per country

flu_country_means <- flu_tidy %>% group_by(country) %>%
  summarize(total = sum(cases, na.rm = TRUE), mean = mean(cases, na.rm = TRUE)) %>%
  arrange(desc(mean))
```

Dengue numbers

Flu numbers

Though interesting, these numbers only represent the total number of cases, not taking into account the size of the difference in population between countries. That is where the gapminder dataframe comes into play, containing exactly the data needed to include the population in the calculations.

Table 5.14: Table containing flu numbers per country

country	total	mean
South Africa	1349388	2698.776000
Canada	1169673	1886.569355
United States	1142909	1843.401613
Mexico	703432	1134.567742
Austria	500332	806.987097
Germany	496073	800.117742
Romania	463130	709.234303
Bulgaria	331824	595.734291
Russia	262781	463.458554
Australia	219016	436.286853
Ukraine	204308	397.486381
Bolivia	192202	295.241167
Japan	160897	261.196429
Paraguay	135348	253.460674
Peru	144810	219.742033
Brazil	138825	210.660091
Argentina	100391	153.503058
Belgium	76869	123.982258
Uruguay	76395	117.893518
France	61625	99.395161
Hungary	50572	90.146168
Norway	40462	72.124777
Switzerland	43573	70.850407
Poland	30456	53.619718
Spain	31856	51.380645
New Zealand	19595	39.033864
Netherlands	22408	36.200323
Chile	7544	11.482496
Sweden	2824	5.548134

5.3 Combining the dataframes

5.3.1 Preparing the tidy dataframes

To prepare the dataframes for the joining NA values were removed, and both the sum and the weekly mean of the number of cases were added.

```
#calculate the number of cases per year and the average number of cases per week per country
dengue_combine <- na.omit(dengue_mutated) %>% group_by(country, year) %>%
  summarize(total_cases_dengue = sum(cases1000), mean_weekly_dengue = mean(cases1000))

flu_combine <- na.omit(flu_tidy) %>% group_by(country, year) %>%
  summarize(total_cases_flu = sum(cases), mean_weekly_flu = mean(cases))
```

5.3.2 Joining the dataframes together

The new dataframes were then exported to the database, where they were joined using SQL:

```
dbWriteTable(con, "dengue_combine", dengue_combine) #insert the new tables into the database
dbWriteTable(con, "flu_combine", flu_combine)
```

```
CREATE TABLE dengue_gapminder AS --combine dengue_data and gapminder
SELECT
    gapminder.country,
    gapminder.year,
    gapminder.population,
    gapminder.continent,
    gapminder.region,
    dengue_combine.total_cases_dengue,
    dengue_combine.mean_weekly_dengue
  FROM gapminder
LEFT JOIN dengue_combine
  ON gapminder.country = dengue_combine.country
    AND gapminder.year = dengue_combine.year;
```

```
CREATE TABLE flu_gapminder AS --combine flu_data and gapminder
SELECT
    gapminder.country,
    gapminder.year,
    gapminder.population,
    gapminder.continent,
    gapminder.region,
    flu_combine.total_cases_flu,
    flu_combine.mean_weekly_flu
  FROM gapminder
LEFT JOIN flu_combine
  ON gapminder.country = flu_combine.country
```

```
    AND gapminder.year = flu_combine.year;
```

```
CREATE TABLE flu_gapminder_dengue AS    --combine all 3 tables
SELECT
    flu_gapminder.country,
    flu_gapminder.year,
    flu_gapminder.population,
    flu_gapminder.continent,
    flu_gapminder.region,
    flu_gapminder.total_cases_flu,
    flu_gapminder.mean_weekly_flu,
    dengue_combine.total_cases_dengue,
    dengue_combine.mean_weekly_dengue
  FROM flu_gapminder
INNER JOIN dengue_combine
  ON flu_gapminder.country = dengue_combine.country
    AND flu_gapminder.year = dengue_combine.year;
```

5.3.3 Loading the combined tables into R

```
dengue_gapminder <- dbReadTable(con, "dengue_gapminder") #load the combined tables int
flu_gapminder <- dbReadTable(con, "flu_gapminder")
flu_gapminder_dengue <- dbReadTable(con, "flu_gapminder_dengue")

dengue_gapminder_nona <- na.omit(dengue_gapminder) #remove NA values
flu_gapminder_nona <- na.omit(flu_gapminder)
flu_gapminder_dengue_nona <- na.omit(flu_gapminder_dengue)
```

5.4 Visualizing the data

5.4.1 Normalizing the data

Now that the dataframes are combined, the gapminder column "population" can be used to normalize the data. We now know the average number of weekly cases per 100.000 population, of every country per year.

```
#normalize the data by calculating the average number of cases per 100,000 population
dengue_gapminder_normalized <- dengue_gapminder_nona %>% mutate("normalized_dengue" =
                                                                "normalized_dengue_mea

flu_gapminder_normalized <- flu_gapminder_nona %>% mutate("normalized_flu" = (total_ca
                                                          "normalized_flu_mean" = (mea

flu_gapminder_dengue_normalized <- flu_gapminder_dengue_nona %>% mutate("normalized_de
                                                                        "normalized_fl
```

```
                                                    "normalized_dengue_mean"
                                                    "normalized_flu_mean" = (
```

5.4.2 Calculating descriptive statistics

In an attempt to find some descriptive statistics, I tried calculating the means, standard deviations and Shapiro-Wilk p-values (to see if the data is from a normal distribution).

```
#calculate means, standard deviations and Shapiro-Wilk p-values per region
dengue_gap_summary <- dengue_gapminder_normalized %>% group_by(region) %>%
  summarize(mean = mean(total_cases_dengue),
            sd = sd(total_cases_dengue),
            pvalue_sw = shapiro.test(total_cases_dengue)$p.value)

flu_gap_summary <-flu_gapminder_normalized %>% group_by(region) %>%
  summarize(mean = mean(total_cases_flu),
            sd = sd(total_cases_flu),
            pvalue_sw = shapiro.test(total_cases_flu)$p.value)

#calculate means, standard deviations and Shapiro-Wilk p-values per country
dengue_gap_summary_country <- dengue_gapminder_normalized %>% group_by(country) %>%
  summarize(mean = mean(total_cases_dengue),
            sd = sd(total_cases_dengue),
            pvalue_sw = shapiro.test(total_cases_dengue)$p.value)

flu_gap_summary_country <-flu_gapminder_normalized %>% group_by(country) %>%
  summarize(mean = mean(total_cases_flu),
            sd = sd(total_cases_flu),
            pvalue_sw = shapiro.test(total_cases_flu)$p.value)
```

Not every country has a normal distribution (normal distribution is where $p > 0.05$). After finding the countries that do, levene's test was performed to check for equal variances:

```
#find the countries where the data is a normal distribution
countries_normal_dengue <- subset(dengue_gap_summary_country, pvalue_sw > 0.05)[,1] %>% pull()
countries_normal_flu <- subset(flu_gap_summary_country, pvalue_sw > 0.05)[,1] %>% pull()

#filter for the rows with observations from those countries
dengue_normal <- dengue_gapminder_normalized[dengue_gapminder_normalized$country %in% countries_n
flu_normal <- flu_gapminder_normalized[flu_gapminder_normalized$country %in% countries_normal_flu

#perform levene's test
leveneTest(total_cases_dengue ~ country, data = dengue_normal, center = mean)[1,3]
```

```
## [1] 0.1828471
```

```
leveneTest(total_cases_flu ~ country, data = flu_normal, center = mean)[1,3]
```

```
## [1] 5.411843e-33
```

5.4.3 Visualizing correlation

After calculating these p-values, I realized I'm more interested in correlation than difference between the groups, since it's already known the groups are differt from each other (different countries). Instead of performing an ANOVA or Kruskal-Wallis test, it was decided that the pearson's correlation coefficient would be calculated. A scatter plot was generated to visualize correlation:

```
#perform a pearson correlation analysis
dengue_flu_cor <- round(cor.test(flu_gapminder_dengue_normalized$total_cases_dengue,
                                 flu_gapminder_dengue_normalized$total_cases_flu,
                                 method = "pearson")$estimate, 3)

cor_pvalue <- cor.test(flu_gapminder_dengue_normalized$total_cases_dengue,
         flu_gapminder_dengue_normalized$total_cases_flu,
         method = "pearson")$p.value

#visualize the correlation in a scatter plot
ggplot(data = flu_gapminder_dengue_normalized, aes(y = total_cases_dengue, x = total_ca
  geom_point()+
  labs(title = "Relation between dengue and flu cases",
       subtitle = paste("pearson's r = ", dengue_flu_cor),
       y = "Number of dengue cases",
       x = "Number of flu cases")+
  theme_minimal()
```

There is a significant correlation between the number of dengue and flu cases in a country, as p is smaller than 0.05 at `cor_pvalue`. The correlation coefficient of `dengue_flu_cor` indicates a weak correlation (Taylor 1990).

5.4.4 Visualizing the dengue data

Dengue is endemic in most tropical countries / regions (Gubler 2002), as shown below:

```
#visualize the number of dengue cases per region in a bar plot
ggplot(dengue_gap_summary,
       aes(x = region, y = mean,
           fill = region))+
  geom_col()+
  geom_errorbar(aes(ymin = mean-sd, ymax = mean+sd, width = 0.2))+
  labs(title = "Number of dengue cases per region",
       subtitle = "Error bars depict 1 standard deviation",
```

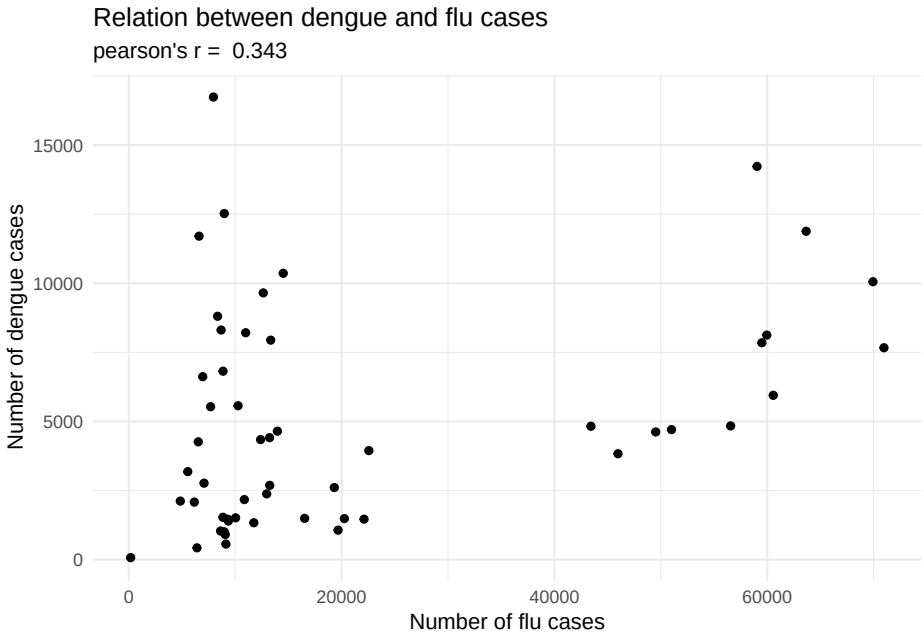

Figure 5.1: Scatter plot showing the correlation between the number of dengue and flu cases, with on the y-axis the number of dengue cases and on the x-axis the number of flu cases. Each dot represents a year of observations from one country.

```
       x="Region",
       y="Average number of dengue cases")+
  theme_minimal()+
  theme(legend.position = "none")
```

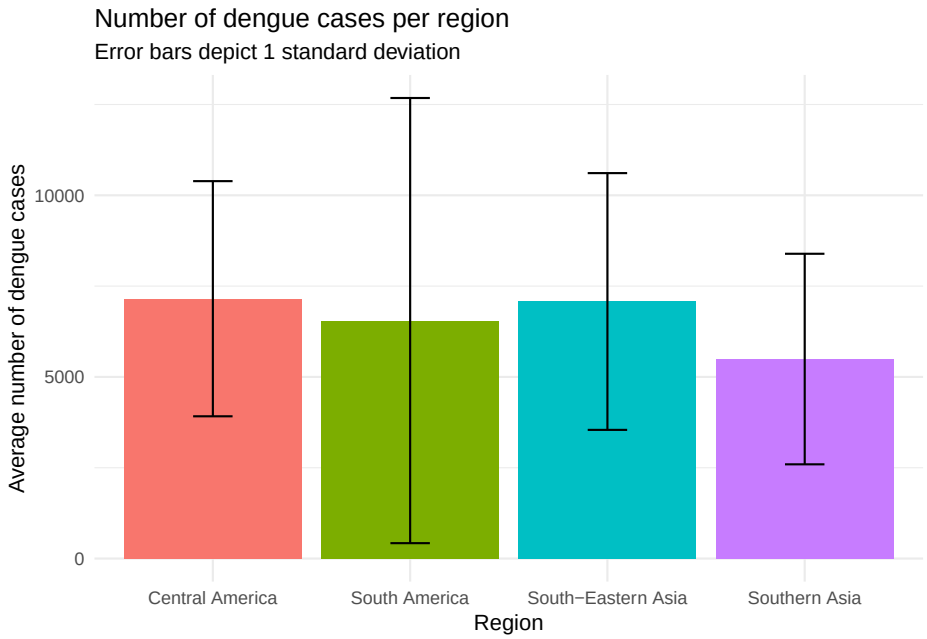

Figure 5.2: Bar plot showing the number of dengue cases per region, with on the y-axis the yearly average number of dengue cases.

This graph was made by looking at the average number of cases per region per year, meaning it uses the total number of cases rather that the normalized ones that were calculated earlier. A more representative plot would use these numbers:

```
#create a scatter plot of the dengue data
dengue_plot <- ggplot(dengue_gapminder_normalized,
       aes(x = year, y = mean_weekly_dengue, group = country,
           color = country))+
  geom_line()+
  geom_point()+
  labs(title = "Weekly dengue cases per country",
       y = "Average number of dengue cases per week\nper 100,000 population",
       x = "Year",
       color = "Country")+
  scale_x_continuous(breaks = seq(from = min(dengue_gapminder_normalized$year),
                                  to = max(dengue_gapminder_normalized$year),
```

Table 5.15: 1 records

country	region
South Africa	Southern Africa

```
                                by = 1))+
  theme_minimal()

ggplotly(dengue_plot) #visualize the number of dengue cases per country in an interactive scatter
```

Interactive scatter plot showing the normalized number of dengue cases per country, with on the y-axis the average number of dengue cases per week per 100,000 population and on the x-axis the year.

5.4.5 Visualizing the flu data

The same graphs can be made using the flu data:

```
#visualize the number of flu cases per region in a bar plot
ggplot(flu_gap_summary,
       aes(x = region, y = mean,
           fill = region)) +
  geom_col()+
  geom_errorbar(aes(ymin = mean-sd, ymax = mean+sd, width = 0.2))+
  labs(title = "Number of flu cases per region",
       subtitle = "Error bars depict 1 standard deviation",
       x="Region",
       y="Average number of flu cases")+
  theme_minimal()+
  theme(legend.position = "none",
        axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
```

This graph has the same issue as before, using the total number of cases rather than the normalized data. Northern America and Southern Africa stand out with their high number of cases. It is true that Northern America has a high number of weekly flu cases, but seeing their large population this was to be expected. More interesting is Southern Africa, where the population is smaller but the number of cases is higher. Looking at some random observations:

```
SELECT DISTINCT country, region
FROM flu_gapminder
WHERE region = 'Southern Africa'
AND total_cases_flu IS NOT NULL
```

South Africa is the only country in region Southern Africa with flu data.

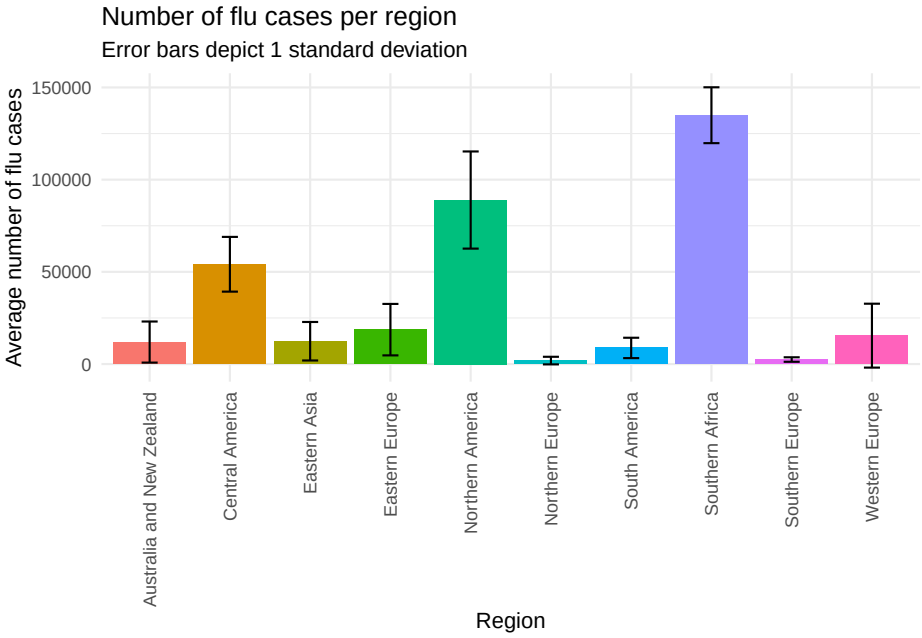

Figure 5.3: Bar plot showing the number of flu cases per region, with on the y-axis the yearly average number of flu cases.

```
SELECT year, country, cases
  FROM flu
WHERE country = 'South Africa'
  AND cases IS NOT NULL
ORDER BY RANDOM();
```

```
SELECT year, country, cases
  FROM flu
WHERE cases IS NOT NULL
ORDER BY RANDOM();
```

We can see that the number of weekly cases in South Africa is very high compared to other countries, causing it to stand out when looking at the bar graph. This will also be reflected in the average number of weekly cases:

```
#create a scatter plot of the flu data
flu_plot <- ggplot(flu_gapminder_normalized,
       aes(x = year, y = mean_weekly_flu, group = country,
           color = country))+
  geom_line()+
  geom_point()+
  labs(title = "Weekly flu cases per country",
```

Table 5.16: Displaying records 1 - 10

year	country	cases
2012	South Africa	2120
2012	South Africa	1926
2007	South Africa	3367
2011	South Africa	3719
2011	South Africa	2096
2013	South Africa	1854
2009	South Africa	3256
2012	South Africa	2734
2013	South Africa	3454
2014	South Africa	2852

Table 5.17: Displaying records 1 - 10

year	country	cases
2008	Spain	27
2013	Spain	29
2008	Australia	1396
2011	Sweden	3
2014	South Africa	2325
2006	Russia	338
2005	Switzerland	10
2006	Germany	332
2014	Japan	18
2009	Japan	379

```
       y = "Average number of flu cases per week\nper 100,000 population",
       x = "Year",
       color = "Country")+
  scale_x_continuous(breaks = seq(from = min(flu_gapminder_normalized$year),
                                  to = max(flu_gapminder_normalized$year),
                                  by = 1))+
  theme_minimal()

ggplotly(flu_plot) #visualize the number of flu cases per country in an interactive sc
```

Interactive scatter plot showing the normalized number of flu cases per country, with on the y-axis the average number of flu cases per week per 100,000 population and on the x-axis the year.

Northern American countries United States and Canada still stand out, having higher weekly averages than most countries, despite the numbers being normalized for this graph. South Africa does have the highest average number of flu cases per week per 100,000 population since data started being collected from there.

Now that the analysis and visualization is done, Rstudio can be disconnected from the database.

```
dbDisconnect(con)
```

Chapter 6

Parameterized report on COVID-19

6.1 Parameterized reports

Up until now, every chapter has been a static report. When using a dataset with, for example, a large number of groups, it would be nice to able to choose what groups exactly you want to visualize with just the click of a button. This is what parameterized reports are for. In this report I will be visualizing COVID-19 data (obtained from ECDC), more specifically the daily numbers of newly reported COVID-19 cases and deaths in EU countries.

6.2 The parameterized COVID-19 report

6.2.1 The parameters

The parameters I will be using are country, year and month. Using these parameters, I will generate two interactive plots: one for the newly reported number of cases and one for the newly reported number of deaths.

6.2.2 Rendering the report

The parameters are chosen when rendering the report, and are set as follows:

```
#generate the report with chosen parameters
rmarkdown::render("06_parameterizedcovid.Rmd", params = list(country = c("Germany", "France", "Ne
                                                             year = 2022,
                                                             month = 1:3))
```

For the parameterization to work, the YAML header of the markdown file needs to contain some defaults for the parameters, that are used when the parameter is not specified when rendering.

```
---
params:
  country: "Austria"
  year: 2022
  month: 10
---
```

```
library(tidyverse)
library(plotly)
library(scales)
library(here)
```

```
data <- read.csv(here("data_raw/data.csv")) #load the data
```

After the data is loaded, it can be filtered to reflect the given parameters. I will also be determining the axis breaks depending on how many years are plotted, to keep the x-axis readable.

```
data_filtered <- data %>% filter(countriesAndTerritories %in% params$country,
                                 year %in% params$year,
                                 month %in% params$month) #filter for the given parame

data_filtered <- mutate(data_filtered, "date" = paste(day, month, year, sep="/")) #add

data_filtered$date <- as.Date(data_filtered$date, format="%d/%m/%Y") #change the data

if(length(params$year) > 1){ #determine the x-axis breaks
  datebreaks <- "1 month"
}else{
  datebreaks <- "2 weeks"
}
```

Finally, the data can be plotted:

```
plot_cases <- ggplot(data_filtered,
                     aes(x = date, y = cases, group = countriesAndTerritories,
                         color = countriesAndTerritories))+
              geom_line()+
              geom_point(size = 1)+
              labs(title = "Number of newly reported COVID-19 cases over time by counti
                   y = "Number of COVID-19 cases",
                   x = "Date",
                   color = "Country")+
              scale_x_date(date_breaks = datebreaks, date_labels = "%d-%m-%y")+
              scale_y_continuous(labels = label_comma())+
```

```
              theme_minimal()+
              theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))

ggplotly(plot_cases) #generate an interactive plot showing the number of cases
```

Number of newly reported COVID-19 cases over time by country, with on the y-axis the number of COVID-19 cases and on the x-axis the date.

```
plot_deaths <- ggplot(data_filtered,
                      aes(x = date, y = deaths, group = countriesAndTerritories,
                          color = countriesAndTerritories))+geom_line()+
                geom_point(size = 1)+
                labs(title = "Number of newly reported COVID-19 deaths over time by country",
                     y = "Number of COVID-19 deaths",
                     x = "Date",
                     color = "Country")+
                scale_x_date(date_breaks = datebreaks, date_labels = "%d-%m-%y")+
                scale_y_continuous(labels = label_comma())+
                theme_minimal()+
                theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))

ggplotly(plot_deaths) #generate an interactive plot showing the number of deaths
```

Number of newly reported COVID-19 deaths over time by country, with on the y-axis the number of COVID-19 deaths and on the x-axis the date.

6.2.3 Using different parameters

Because the report is parameterized, it's simple to recreate the graph for different countries and time periods:

```
#generate the plots with different parameters
rmarkdown::render("06_parameterizedcovid.Rmd", params = list(country = c("Estonia", "Latvia", "Li
                                                             year = 2020:2021,
                                                             month = 1:12))
```

Number of newly reported COVID-19 cases over time by country, with on the y-axis the number of COVID-19 cases and on the x-axis the date, this time using different parameters.

Number of newly reported COVID-19 deaths over time by country, with on the y-axis the number of COVID-19 deaths and on the x-axis the date, this time using different parameters.

Chapter 7

R package vignette

7.1 Creating the package

I have created a package to help me with data analysis. My package {dsfbnine} was created with the help of the {devtools} (Hadley Wickham et al. 2022) and {usethis} (Hadley Wickham, Bryan, and Barrett 2022) packages. The general frame was made using function from these packages, like `usethis::create_package` and `devtools::document`.

The functions included in this package are specific to my own workflow, most were written by looking at my own duplicate code. The main use of the functions are to make my code tidier and help me work faster, since I often do things like plotting illness cases for example. The rest of this chapter will be the vignette that comes with the package.

7.2 Installing and loading the package

```
devtools::install_github("nineluijendijk/dsfbnine")
library(dsfbnine)
```

7.3 Package contents

7.3.1 Dataset measlesdata

The {dsfbnine} package contains some functions to help me with my data analysis and one cleaned dataset `measlesdata`. It contains data on measles cases and population numbers in EU countries from 2018 and 2019. The dataset loads as a dataframe with 744 rows and 8 variables. It is available via lazy-loading, meaning columns can be accessed as follows:

```
measlesdata$cases %>% head(n = 20)
```

```
##  [1]  7  5 15 17 12  6  1  3  6  0  4  1  1  6  9 11 24 22  3 11
```

7.3.2 Function dataSummarizer()

The function `dataSummarizer()` calculates the sum, mean, median and standard deviation of data, grouped by a given argument:

```
dataSummarizer(measlesdata, groupBy = c("CountryName", "year")) %>% head(n = 10)
```

```
## # A tibble: 10 x 6
## # Groups:   CountryName [7]
##    CountryName year  total  mean median standarddeviation
##    <chr>       <chr> <dbl> <dbl>  <dbl>             <dbl>
##  1 EU/EEA      2019  13207 1101. 1012.              783.
##  2 EU/EEA      2018  12352 1029.  959               687.
##  3 France      2018   2913  243.  136               252.
##  4 France      2019   2636  220.  166.              176.
##  5 Italy       2018   2517  210.  170               155.
##  6 Greece      2018   2293  191.   96.5             211.
##  7 Romania     2019   1706  142.  111               72.8
##  8 Italy       2019   1626  136.  162               106.
##  9 Poland      2019   1423  119.   82.5             121.
## 10 Bulgaria    2019   1235  103.   46.5             110.
```

7.3.3 Function illnessPlot()

The function `illnessPlot()` generates a graph showing illness numbers, specifically COVID-19 numbers from an ECDC dataset, used in Chapter 6 of my portfolio. The same code was used to generate 4 different plots, to fix this issue `illnessPlot()` was created.

```
coviddata <- read.csv(here::here("data/datacovid.csv")) #load the data
illnessPlot(coviddata, countries = c("Netherlands", "Belgium", "France"), years = 2020
```

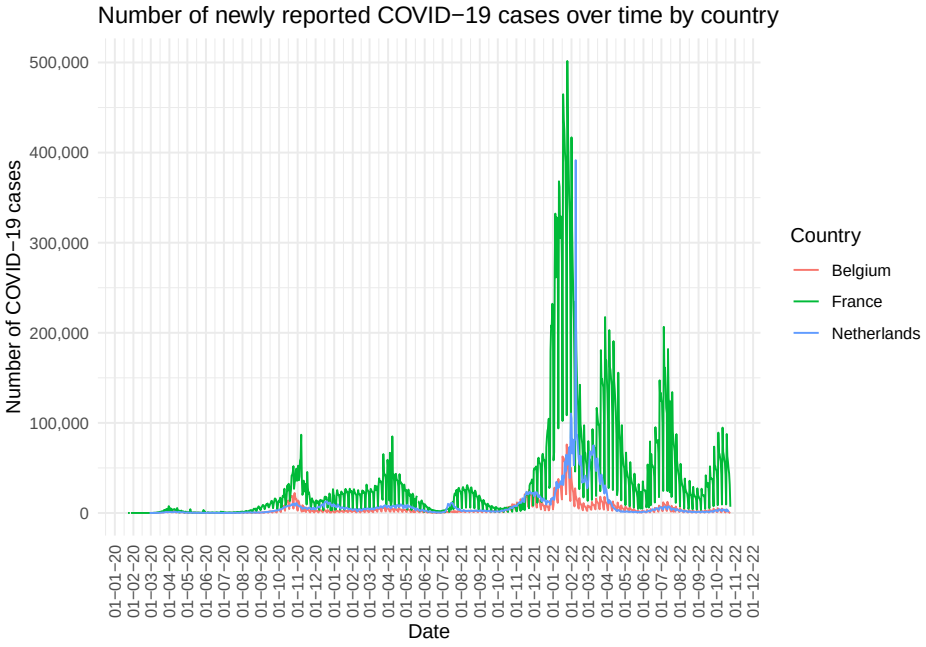

Figure 7.1: Example illness plot.

7.3.4 Function fastaImporter.R

The function `fastaImporter()` can be used to import FASTA sequences by entering their GenBank Identifiers. The imported sequences will be stored in an object of class "DNAbin".

```
dna <- fastaImporter(here::here("data/species.txt"))
dna
```

```
## 16 DNA sequences in binary format stored in a list.
##
## Mean sequence length: 16523.81
##    Shortest sequence: 16264
##     Longest sequence: 17062
##
## Labels:
## KF914213.1 Gorilla beringei graueri mitochondrion, complete ...
## NC_011120.1 Gorilla gorilla gorilla mitochondrion, complete ...
## GU170821.1 Homo sapiens isolate NS07 mitochondrion, complete...
## HM015213.1 Pan paniscus isolate PP25 mitochondrion, complete...
## NC_001643.1 Pan troglodytes mitochondrion, complete genome
## NC_002083.1 Pongo abelii mitochondrion, complete genome
## ...
```

```
##
## Base composition:
##     a     c     g     t
## 0.314 0.299 0.131 0.256
## (Total: 264.38 kb)
```

7.3.5 Function gotermAnalysis()

The function `gotermAnalysis()` was written specifically to help with my analysis of RNA-sequencing data, which is why there are many different parameters to be set: changing just 1 can completely change the results (seen in the code chunk below). The function performs a GO-term enrichment analysis. The output of this function can be used in the next function `gotermPlot()` as well.

```
DESeqresults <- readRDS(here::here("data/dge_results")) #load the data object of class

gotermAnalysis_results <- gotermAnalysis(DESeqresults)
```

7.3.6 Function gotermPlot()

The function `gotermPlot()` is an extension to the previous `gotermAnalysis()` function, generating a figure containing the most abundant GO-terms. To save computing time, the `gotermAnalysis()` function will not be ran. This means its output has to be imported for the `gotermPlot()` function:

```
gotermAnalysis_results <- readRDS(here::here("data/gotermAnalysis_results"))

gotermPlot(gotermAnalysis_results, topamount = 10, plot_title = "Top 10 upregulated GO
```

7.4 Additional files

Additional files can be found in the `/inst/extdata/` directory. It consist of files used to build the vignette, (raw) data for analysis/plotting, example files and an R script that shows how the dataset `measlesdata` was cleaned.

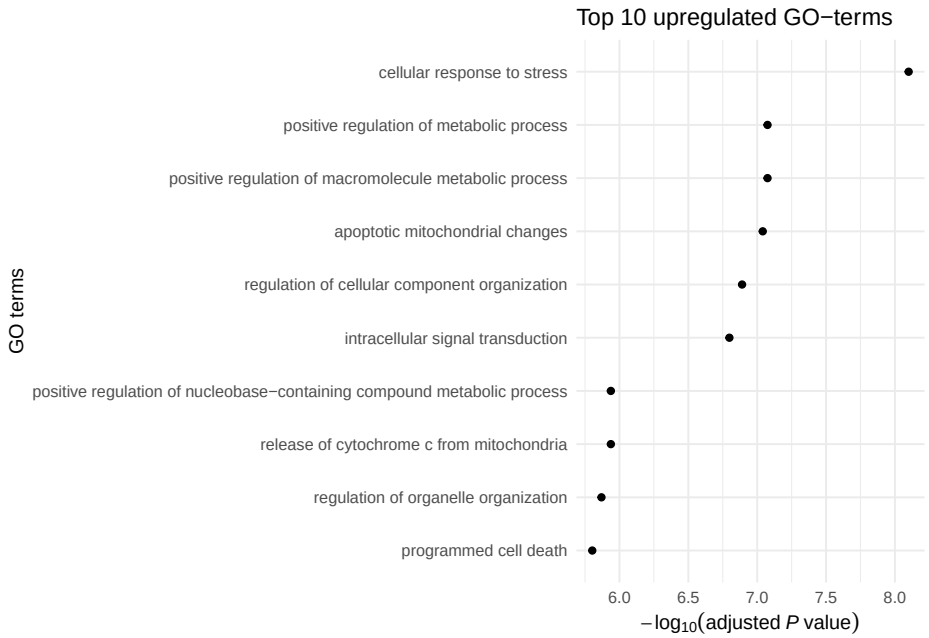

Figure 7.2: Example GO-term plot.

Chapter 8

Generating a phylogenetic tree in R

8.1 The plan

Starting September 2024 I want to be doing my master's in Biology at a research University. I would like to specialize in Evolutionary Biology. During my minor I attended classes at Radboud University together with other (pre-master) students in Biology - Adaptive Organisms to get a head start on learning about evolutionary biology, both because it's something I enjoy and to make the transition from Life Sciences to a master's program easier.

I would like to learn more about the differences between multiple primates and generate a phylogenetic tree. Prior to me starting on the actual code for this phylogenetic tree, I had to look into the different types of trees and how to generate them.

8.2 Preparatory research

I started by looking for R packages having to do with phylogenetic research. There were ample packages to help generate phylogenetic trees, but none of them seemed to have all the tools needed. Wanting to keep my research as reproducible as possible, my main priority was to keep every single part of the research in R, which is free and open source software (R Core Team 2014). To get an idea of how to get started, I read a couple of papers by researchers who've created phylogenetic trees. In these papers, most of the software used to generate the trees (and all the steps that come before) was not R (Pozzi et

al. 2014); (Perelman et al. 2011). Some of the programs were point and click as well (Suchard et al. 2018); (Vaidya, Lohman, and Meier 2011), making the research less reproducible.

After looking at some different R packages I decided {ape} (Paradis and Schliep 2019) was the best option, seeing as most packages for phylogenetic research are dependent on it (Jombart and Ahmed 2011). The first step in the analysis was to obtain the data from an online database. NCBI has their own command-line tool called NCBI Datasets (NCBI Staff 2022). The command-line tool was installed using Miniconda3 (Anaconda n.d.). The tool did work fine, but it wasn't exactly what I was looking for. This was also when it was decided to perform the analysis on the mitochondrial genomes of the primates, instead of the nuclear genomes, due to computational limitations.

8.3 Getting started

I ended up making a list of the species of interest, containing their GenBank (Benson et al. 2013) identifier, scientific name and their common name, and saving it in a text file called species.txt. This is the only part of the code that would need to be changed in case a species is to be added.

To keep the code tidy the packages {tidyverse} (Hadley Wickham et al. 2019) and {here} (Müller n.d.) are used. {BiocManager} (Ramos and Morgan n.d.) was also used to install packages.

8.4 Generating the first tree

8.4.1 Obtaining FASTA sequences

The file is read by R and filtered so that the comments (species' names) are removed, and only the GenBank identifier remains. Then, using the package {rentrez} (Winter 2017), the FASTA sequences are obtained and stored in a vector. The FASTA sequences are then stored in a file so they can be used as input for the {ape} function `read.FASTA`, where the sequences will be converted to a vector of the class DNAbin.

```
species <-scan(file = here("data/species.txt"), what = "character", sep = "\n", comment

fasta_seqs <- entrez_fetch(db = "nucleotide",
             id = species,
             rettype = "fasta") #retrieve the fasta sequences of the mitochondrial gen

write(fasta_seqs,
      file="temp")

grep("^*$", readLines("temp"), invert = TRUE, value = TRUE) %>% write(file = here("data
```

```
file.remove("temp")

dna <- read.FASTA(here("data/sequences.fasta"), type = "DNA") #store the sequences in a vector o
```

8.4.2 Performing multiple sequence alignment

Now that the dna has been stored as DNAbin, a multiple sequence alignment will be performed using Clustal Omega 1.2.3 (Sievers et al. 2011). The aligned DNA is saved as a file, so the alignment only has to be performed once (since it takes a while).

```
dna_aligned <- clustalomega(dna, exec = here("clustalo")) #use Clustal Omega 1.2.3 to perform mu
saveRDS(dna_aligned, here("data/dna_aligned.RDS")) #save as file so the code only has to run once
```

8.4.3 Importing the aligned dna

```
dna_aligned <- readRDS(here("data/dna_aligned.RDS")) #import the aligned dna
```

8.4.4 Calculating the genetic distances between the sequences and drawing a tree

The different dna sequences have been aligned and now the genetic distances between the sequences can be calculated using the Tamura and Nei 1993 model (TN93) (Tamura and Nei 1993). The distance matrix can be used as input to calculate the phylogenetic tree. The used algorithm is the BIONJ algorithm of Gascuel (Gascuel 1997). The generated tree is visualized as a cladogram using the package {ggtree} (Yu et al. 2017). It's visualized as a cladogram for now to keep it easy to read while it's being improved.

```
dna_dist <- dist.dna(dna_aligned, model = "TN93") #calculate genetic distances and store in a vec

treebionj <- bionj(dna_dist) #create a tree of the class "phylo" using the method neighbor joini

ggtree(treebionj, branch.length = "none", ladderize = F)+ #visualize the tree as a cladogram
  theme_tree()+
  geom_tiplab(size = 6, as_ylab = T)+
  labs(title = "Unrooted neighbor joining cladogram of\nmitochondrial genomes of 15 primates and
```

8.5 Different types of phylogenetic models

This first tree 8.1 was generated using an improved neighbor joining model, which is just one of the four most commonly used (Yoshida and Nei 2016). Neighbor joining only looks at the distance between the sequences, but not at smaller details in the sequences (Yoshida and Nei 2016). A different method is

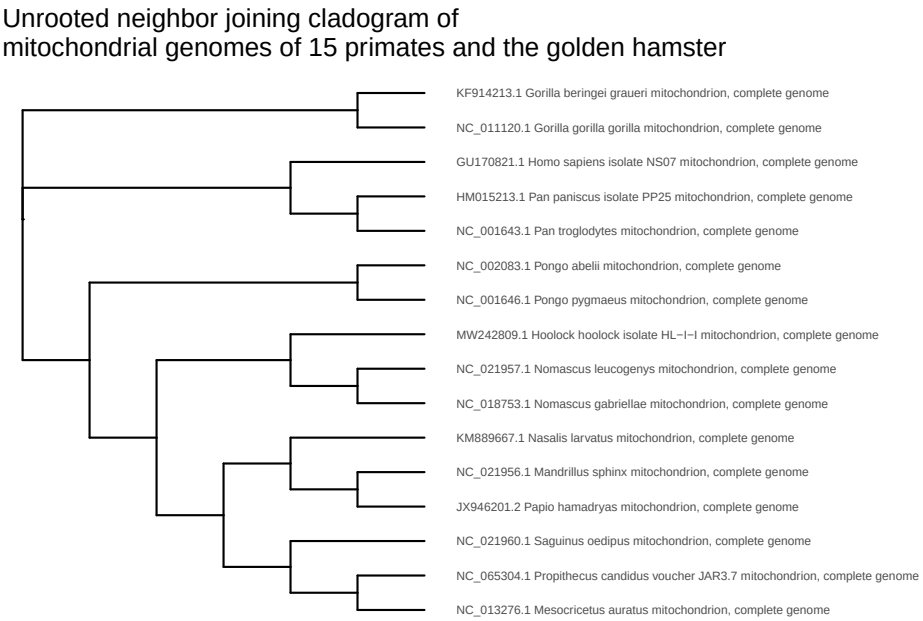

Figure 8.1: Unrooted neighbor joining cladogram of mitochondrial genomes of 15 primates and the golden hamster.

maximum likelihood, where the likelihood of the different possible states of the tree is calculated, and the tree with the highest likelihood is generated (Dhar and Minin 2016). Two other popular methods are Bayesian inference and maximum parsimony (Dhar and Minin 2016); (Yoshida and Nei 2016). Since maximum likelihood is the preferred method in publications (Dhar and Minin 2016), that is the method I will be using.

8.5.1 Finding the best model

The input for the {phangorn} `modelTest` function is an object of the class phyDat, so first the DNAbin object has to be converted:

```
dna_phydat <- as.phyDat(dna_aligned) #convert object to class phyDat
```

The model test is ran and the model with the lowest Akaike information criterion (AIC, calculated by the {phangorn} function `modelTest`), making it the best model (Cavanaugh and Neath 2019), is saved, again so this only has to be performed once as it takes quite long.

```
modeltest <- modelTest(dna_phydat) #run the model test
optmodel <- modeltest$Model[modeltest$AIC==min(modeltest$AIC)] #find the model with th
saveRDS(optmodel, here("data/optmodel")) #save as file so the code only has to run onc
```

```
optmodel <- readRDS(here("data/optmodel")) #import the data again
```

8.5.2 Generating a maximum likelihood tree

Using the neighbor joining tree, the aligned dna and the best model the maximum likelihood tree can be computed:

```
mlparsed <- pml(treebionj, dna_phydat) #generate the maximum likelihood tree
```

The `optim.pml` function from {phangorn} is used to optimize the different model parameters. The model found using `modelTest` is not an option in the `optim.pml` function, so the closest option is used.

```
optmodel <- gsub("\\+.*","", optmodel) #change the string so the optim.pml function can read it
mlparsedoptim <- optim.pml(mlparsed, optNni=TRUE, model = optmodel) #optimize the tree
```

8.5.3 Visualizing the maximum likelihood cladogram

```
treeml <- ladderize(mlparsedoptim$tree) #ladderize the tree

ggtree(treeml, branch.length = "none", ladderize = F)+ #visualize the tree as a cladogram
  theme_tree()+
  geom_tiplab(size = 6, as_ylab = T)+
  labs(title = "Unrooted maximum likelihood cladogram of\nmitochondrial genomes of 15 primates ar
```

8.5.4 Rooting the tree

The last step is to root the tree and clean up the tip labels (species names). It is important to root the tree, trees that aren't rooted correctly may be misleading (Kinene et al. 2016). There are multiple ways to root a tree, I will be using an "outgroup", a species that is genetically very different from the other species in the tree (Kinene et al. 2016).

```
rootedml <- root(treeml,
                 outgroup = "NC_013276.1 Mesocricetus auratus mitochondrion, complete genome",
                 resolve.root = TRUE) #rooting the tree

rootedml$tip.label <- gsub(".*\\.[12]", "", rootedml$tip.label) #clean up the labels
rootedml$tip.label <- gsub("(mitochondrion, | isolate | voucher).*genome", "", rootedml$tip.label

ggtree(rootedml, ladderize = F)+ #visualize the tree as a phylogenetic tree
  theme_tree()+
  geom_tiplab(size = 2.5, as_ylab = F)+
  xlim(0, 0.35)+
  labs(title = "Rooted maximum likelihood phylogenetic tree of\nmitochondrial genomes of 15 prima
```

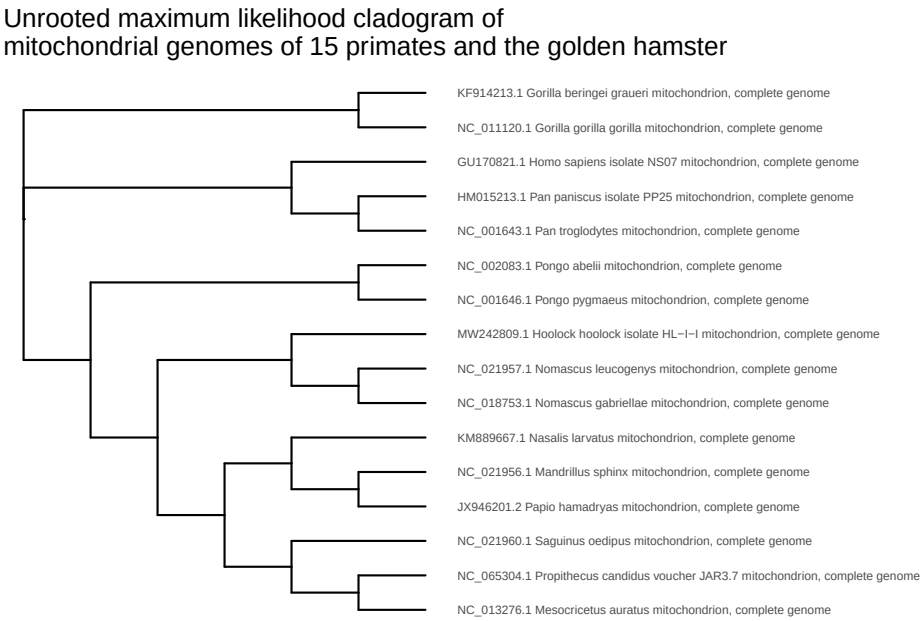

Figure 8.2: Unrooted maximum likelihood cladogram of mitochondrial genomes of 15 primates and the golden hamster

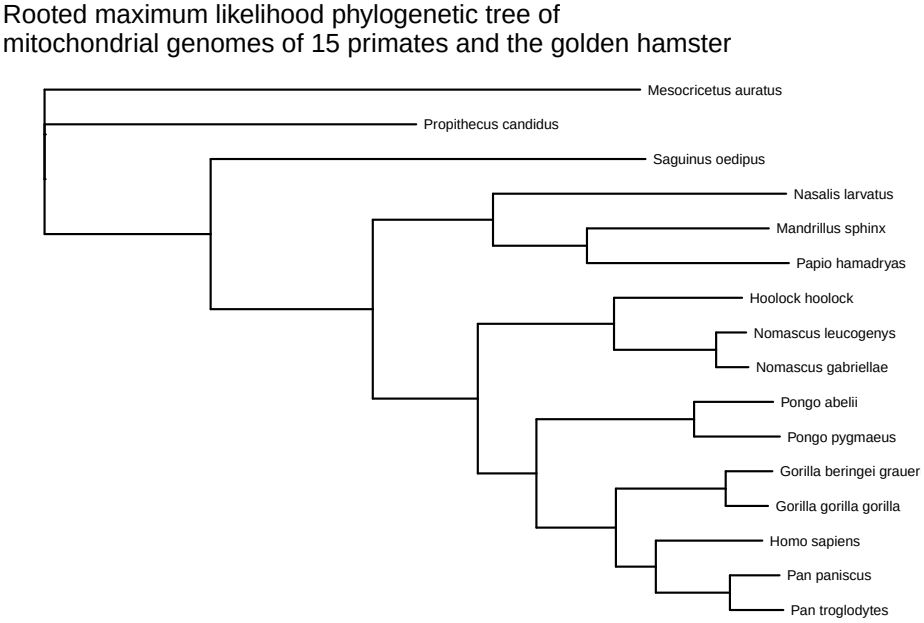

Figure 8.3: Rooted maximum likelihood phylogenetic tree of mitochondrial genomes of 15 primates and the golden hamster

The final tree 8.3 is visualized as a phylogenetic tree instead of a cladogram, with different branch lengths describing the ancestry of the primates. The cladogram only showed how the different species are related (Pearson et al. 2013).

Normally, more thought would go into selecting the best possible outgroup (Rota-Stabelli and Telford 2008). Since my goal was to learn how to generate a phylogenetic tree in R (within a reasonable timeframe) I chose an animal that I like and know is more genetically different from the primates than the primates between each other.

References

Anaconda. n.d. “Miniconda — Conda Documentation.” Accessed December 21, 2022. https://docs.conda.io/en/latest/miniconda.html.

Benson, Dennis A., Mark Cavanaugh, Karen Clark, Ilene Karsch-Mizrachi, David J. Lipman, James Ostell, and Eric W. Sayers. 2013. “GenBank.” *Nucleic Acids Research* 41 (Database issue): D36–42. https://doi.org/10.1093/nar/gks1195.

Cavanaugh, Joseph E., and Andrew A. Neath. 2019. “The Akaike Information Criterion: Background, Derivation, Properties, Application, Interpretation, and Refinements.” *WIREs Computational Statistics* 11 (3): e1460. https://doi.org/10.1002/wics.1460.

Coen A. Bernaards, and Robert I. Jennrich. 2005. “Gradient Projection Algorithms and Software for Arbitrary Rotation Criteria in Factor Analysis.” *Educational and Psychological Measurement* 65: 676--696.

Cokol, Murat. 2012. “Drugs and Their Interactions.” *Current Drug Discovery Technologies* 10 (December). https://doi.org/10.2174/1570163811310020003.

Dhar, Amrit, and V.N. Minin. 2016. “Maximum Likelihood Phylogenetic Inference.” *Encyclopedia of Evolutionary Biology*, December. https://doi.org/10.1016/B978-0-12-800049-6.00207-9.

Gascuel, O. 1997. “BIONJ: An Improved Version of the NJ Algorithm Based on a Simple Model of Sequence Data.” *Molecular Biology and Evolution* 14 (7): 685–95. https://doi.org/10.1093/oxfordjournals.molbev.a025808.

Gubler, Duane J. 2002. “The Global Emergence/Resurgence of Arboviral Diseases As Public Health Problems.” *Archives of Medical Research* 33 (4): 330–42. https://doi.org/10.1016/S0188-4409(02)00378-8.

“High-Throughput Screening (HTS).” n.d. Research. Accessed January 1, 2023. https://research.prinsesmaximacentrum.nl/en/core-facilities/high-throughput-screening.

Irizarry, Rafael A., and Amy Gill. 2021. “Dslabs: Data Science Labs.” https://CRAN.R-project.org/package=dslabs.

Jaaks, Patricia, Elizabeth A. Coker, Daniel J. Vis, Olivia Edwards, Emma F. Carpenter, Simonetta M. Leto, Lisa Dwane, et al. 2022. “Effective Drug Combinations in Breast, Colon and Pancreatic Cancer Cells.” *Nature* 603 (7899): 166–73. https://doi.org/10.1038/s41586-022-04437-2.

Jombart, Thibaut, and Ismaïl Ahmed. 2011. "Adegenet 1.3-1: New Tools for the Analysis of Genome-Wide SNP Data." *Bioinformatics* 27 (21): 3070–71. https://doi.org/10.1093/bioinformatics/btr521.

Kinene, T., J. Wainaina, S. Maina, and L.M. Boykin. 2016. "Rooting Trees, Methods For." *Encyclopedia of Evolutionary Biology*, 489–93. https://doi.org/10.1016/B978-0-12-800049-6.00215-8.

Lewandowska, Anna, Barbara Zych, Katalin Papp, Dana Zrubcová, Helena Kadučáková, Mária Šupínová, Serap Ejder Apay, and Małgorzata Nagórska. 2021. "Problems, Stressors and Needs of Children and Adolescents with Cancer." *Children* 8 (12): 1173. https://doi.org/10.3390/children8121173.

Michalak, Nicholas M., Oliver Sng, Iris M. Wang, and Joshua Ackerman. 2020. "Sounds of Sickness: Can People Identify Infectious Disease Using Sounds of Coughs and Sneezes? | Proceedings of the Royal Society B: Biological Sciences." June 10, 2020. https://doi.org/10.1098/rspb.2020.0944.

Müller, Kirill. n.d. "Here." Accessed December 21, 2022. https://here.r-lib.org/articles/here.html.

NCBI Staff, NCBI. 2022. "Now Available! Updated NCBI Datasets Command-Line Tools." NCBI Insights. October 12, 2022. https://ncbiinsights.ncbi.nlm.nih.gov/2022/10/12/ncbi-datasets-command-line-tools/.

Paradis, Emmanuel, and Klaus Schliep. 2019. "ape 5.0: an environment for modern phylogenetics and evolutionary analyses in R." *Bioinformatics (Oxford, England)* 35 (3): 526–28. https://doi.org/10.1093/bioinformatics/bty633.

Pearson, Talima, Heidie M. Hornstra, Jason W. Sahl, Sarah Schaack, James M. Schupp, Stephen M. Beckstrom-Sternberg, Matthew W. O'Neill, et al. 2013. "When outgroups fail; phylogenomics of rooting the emerging pathogen, Coxiella burnetii." *Systematic Biology* 62 (5): 752–62. https://doi.org/10.1093/sysbio/syt038.

Perelman, Polina, Warren E. Johnson, Christian Roos, Hector N. Seuánez, Julie E. Horvath, Miguel A. M. Moreira, Bailey Kessing, et al. 2011. "A Molecular Phylogeny of Living Primates." *PLOS Genetics* 7 (3): e1001342. https://doi.org/10.1371/journal.pgen.1001342.

Pozzi, Luca, Jason A. Hodgson, Andrew S. Burrell, Kirstin N. Sterner, Ryan L. Raaum, and Todd R. Disotell. 2014. "Primate Phylogenetic Relationships and Divergence Dates Inferred from Complete Mitochondrial Genomes." *Molecular Phylogenetics and Evolution* 75 (June): 165–83. https://doi.org/10.1016/j.ympev.2014.02.023.

Prinses Máxima Centrum. n.d. "Ons verhaal." Prinses Máxima Centrum. Accessed January 1, 2023. https://www.prinsesmaximacentrum.nl/nl/over-ons/ons-verhaal.

R Core Team. 2014. "R: A Language and Environment for Statistical Computing." *MSOR Connections*. https://www.semanticscholar.org/paper/R%3A-A-language-and-environment-for-statistical-Team/659408b243cec55de8d0a3bc51b81173007aa89b.

Ramos, Marcel, and Martin Morgan. n.d. "Installing BiocManager." Accessed December 21, 2022. https://cran.r-project.org/web/packages/BiocManager

/vignettes/BiocManager.html.

Ridge, Enda. 2014. "Guerrilla Analytics - 1st Edition." 2014. https://www.elsevier.com/books/T/A/9780128002186.

Ritz, Christian, Florent Baty, Jens C. Streibig, and Daniel Gerhard. 2015. "Dose-Response Analysis Using R." *PLOS ONE* 10 (12): e0146021. https://doi.org/10.1371/journal.pone.0146021.

Rota-Stabelli, Omar, and Maximilian J. Telford. 2008. "A Multi Criterion Approach for the Selection of Optimal Outgroups in Phylogeny: Recovering Some Support for Mandibulata over Myriochelata Using Mitogenomics." *Molecular Phylogenetics and Evolution* 48 (1): 103–11. https://doi.org/10.1016/j.ympev.2008.03.033.

Rounsaville, Bruce J., Nancy M. Petry, and Kathleen M. Carroll. 2003. "Single Versus Multiple Drug Focus in Substance Abuse Clinical Trials Research." *Drug and Alcohol Dependence* 70 (2): 117–25. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3662471/.

Schüz, Joachim, and Eve Roman. 2021. "Childhood Cancer: A Global Perspective." *Cancer Epidemiology*, Childhood Cancer: A Global Perspective, 71 (April): 101878. https://doi.org/10.1016/j.canep.2020.101878.

Sievers, Fabian, Andreas Wilm, David Dineen, Toby J. Gibson, Kevin Karplus, Weizhong Li, Rodrigo Lopez, et al. 2011. "Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega." *Molecular Systems Biology* 7 (October): 539. https://doi.org/10.1038/msb.2011.75.

SteliarovaFoucher, Eva, Charles Stiller, Peter Kaatsch, Franco Berrino, Jan-Willem Coebergh, Brigitte Lacour, and Max Parkin. 2004. "Geographical patterns and time trends of cancer incidence and survival among children and adolescents in Europe since the 1970s (the ACCISproject): an epidemiological study." *Lancet (London, England)* 364 (9451): 2097–2105. https://doi.org/10.1016/s0140-6736(04)17550-8.

Suchard, Marc A, Philippe Lemey, Guy Baele, Daniel L Ayres, Alexei J Drummond, and Andrew Rambaut. 2018. "Bayesian Phylogenetic and Phylodynamic Data Integration Using BEAST 1.10." *Virus Evolution* 4 (1): vey016. https://doi.org/10.1093/ve/vey016.

Sumner, Josh, Leah Haynes, Sarah Nathan, Cynthia Hudson-Vitale, and Leslie D. McIntosh. 2020. "Reproducibility and Reporting Practices in COVID-19 Preprint Manuscripts." medRxiv. https://doi.org/10.1101/2020.03.24.20042796.

Tamura, K, and M Nei. 1993. "Estimation of the Number of Nucleotide Substitutions in the Control Region of Mitochondrial DNA in Humans and Chimpanzees." *Molecular Biology and Evolution* 10 (3): 512–26. https://doi.org/10.1093/oxfordjournals.molbev.a040023.

Taylor, Richard. 1990. "Interpretation of the Correlation Coefficient: A Basic Review." *Journal of Diagnostic Medical Sonography* 6 (1): 35–39. https://doi.org/10.1177/875647939000600106.

Vaidya, Gaurav, David J. Lohman, and Rudolf Meier. 2011. "SequenceMatrix: Concatenation Software for the Fast Assembly of Multi-Gene Datasets with Character Set and Codon Information." *Cladistics* 27 (2): 171–80. https:

//doi.org/10.1111/j.1096-0031.2010.00329.x.

Wickham, Hadley. 2016. "Ggplot2: Elegant Graphics for Data Analysis." 2016. https://ggplot2.tidyverse.org.

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

Wickham, Hadley, Jennifer Bryan, and Malcom Barrett. 2022. "Usethis: Automate Package and Project Setup." https://usethis.r-lib.org, https://github.com/r-lib/usethis.

Wickham, Hadley, Jim Hester, Winston Chang, and Jennifer Bryan. 2022. "Devtools: Tools to Make Developing R Packages Easier." https://devtools.r-lib.org/, https://github.com/r-lib/devtools.

Wickham, H, J Ooms, and K Müller. 2022. "RPostgres: Rcpp Interface to PostgreSQL." https://rpostgres.r-dbi.org.

Winter, David J. 2017. "Rentrez: An R Package for the NCBI eUtils API" 9.

Yoshida, Ruriko, and Masatoshi Nei. 2016. "Efficiencies of the NJp, Maximum Likelihood, and Bayesian Methods of Phylogenetic Construction for Compositional and Noncompositional Genes." *Molecular Biology and Evolution* 33 (6): 1618–24. https://doi.org/10.1093/molbev/msw042.

Yu, Guangchuang, David K. Smith, Huachen Zhu, Yi Guan, and Tommy Tsan-Yuk Lam. 2017. "Ggtree: An r Package for Visualization and Annotation of Phylogenetic Trees with Their Covariates and Other Associated Data." *Methods in Ecology and Evolution* 8 (1): 28–36. https://doi.org/10.1111/2041-210X.12628.

Zheng, Shuyu, Wenyu Wang, Jehad Aldahdooh, Alina Malyutina, Tolou Shadbahr, Ziaurrehman Tanoli, Alberto Pessia, and Jing Tang. 2022. "SynergyFinder Plus: Toward Better Interpretation and Annotation of Drug Combination Screening Datasets." *Genomics, Proteomics & Bioinformatics*, January. https://doi.org/10.1016/j.gpb.2022.01.004.