Day 1: General info, R Basics

Niklas Schandry

April 2020

Contents

Preface

Personal note

If you have no background in any type of programming, and no understanding of statistics this course will possibly be challenging. Even if you have experience, programming learning curves are steep, the first days

are incredibly frustrating, most things will not make sense to you. You will probably try things and they will not work, the error message you get makes no sense, and you are 90% sure it is not your fault... However, in 99% of cases it does not work because you entered a command that is wrong. Keep in mind that your experience is normal. You can reach out to me, or your peers to help each other. I am trying to provide a very low-level intro in this document. If you know R, this may not be informative. In some cases I may use terms that may not be the most accurate term to convey concepts.

I highly recommend the use of cheatsheets to gain a quick overview over how things work and for reference. Most importantly (below are clickable links):

- The Rstudio cheatsheet
- base R
- dpylr
- ggplot2

Grading

You will be asked to input results into moodle. There are two types of tasks, those that require code and code output and those that require text. Please note the following: The nature of writing code is that you can solve your problem by doing a google or stackoverflow search and you will find *a* solution. I do not see anything wrong with this, this is how it is done a lot of the time. If you need help, make use of the chat on moodle to discuss with me or your peers. If you just copy and paste from google or stack overflow, I cannot prove it. It is your responsibility to answer the question correctly. However, if it becomes obvious to me that your solutions have all been copy & paste from the internet, without any extra effort that shows me that you understand the code you copied, I may ask you to repeat the task. This is especially the case when you copy & paste an answer that is incorrect.

I am aware that this course possibly has a steep learning curve. On some days, the practical parts will be rather short. My explicit expectation is that you will invest time to understand what you are doing, probably more time than actually doing it. Some of the tasks you get aim at checking your understanding by asking you to submit text. Of course, for these tasks copy & paste from the internet (or other sources) constitutes plagiarism and is not accepted. Again, I will be available for questions if you need clarifications.

R & RStudio Background

R is a programming language with a focus on statistics and data analysis. R is syntactically not always easy, and there has been a community effort to make R more usable and modernize the syntax. These efforts are largely collected in the *tidyverse* and are to some degree driven by RStudio Inc. the developers of the RStudio IDE. In this course, you will use programming concepts that are related to the tidyverse.

You will use the RStudioServer IDE during this course. This is hosted by the Leibniz Rechenzentrum (LRZ) and provides largely the same experience as using R on a desktop. The major advantage is that your whole Rsession runs on the LRZ high performance computing infrastructure, you are not limited by your home hardware. The nodes these sessions run on are probably better equipped than desktops or laptops. As a side note: Rstudio is also able to integrate R and python. This is great, but we will *not* make use of it, because I am unable to teach you python, and unless you have some programming experience you are probably unable to learn two programming languages at the same time.

Packages

Packages are collections of functions that are bundled. Packages in R are basically libraries (loaded via library()). Usually packages contain functions that work together, follow a common logic and are designed

for certain purpose. Install and load the tidyverse and patchwork, also load magrittr:

```
install.packages("tidyverse") # installs a package
install.packages("patchwork")
install.packages("Hmisc") # Needed to compute summary statistics with ggplot
library(tidyverse) # loads a package.
library(magrittr)
library(patchwork)
```

Things you will encounter in R

Functions

Functions take R objects (see below) and do something to them. What they do is documented in the documentation. Documentation for any R object can be retrieved using "?":

```
?sum
```

Data

Data is whatever you get out from your experiment. Data can be organized in many ways, there are philosophies centered around different data paradigms. I highly recommend to use data formats that are "tidy". The concept of tidiness in data science is explained here:

https://vita.had.co.nz/papers/tidy-data.pdf

Task: Tidy data

Explain tidy datasets in your own words.

Vectors

R stores basically anything in vectors. Vectors are generated using the c() function. You can think of vectors as one column of table, or a sequence of things.

```
## We assign using <-
numbers <- c(1:3) # : is a range
numbers
```

```
## [1] 1 2 3
```

```
abc <- c("a","b","c")
abc
```

```
## [1] "a" "b" "c"
```

Data frames, tibbles etc

Data Types

Often in programming, things have a "type". Understanding types is an important step in understanding the power of data frames in R, and their relatives like tibbles (tidyverse). Keep this in mind: Computers are dumb.

Look at this code:

```
a <- "2" # this a vector of length 1.
b <- "3"
a + b
```

Why does in not work? The answer is: types. By putting 2 and 3 in "", they are assumed as characters (text or strings). Trying to add two letters or words together makes no sense. Correctly, R refuses to do it.

```
a <- 2
b <- 3
a + b
```

```
## [1] 5
```

This shows why data types are important, they basically inform the computer what it can do with whatever you give it. Historically, tables in computers have one type, you can have a matrix that is only numbers, or only text, but not both. R uses dataframes, where each column can have a different type. By default, data.frames convert text (strings) to factors.

A factor is a special case of string often encountered in data science. Often your text-variables only have a limited number of possible values, e.g. your Treatment variable may only take "Treatment" or "Control". If this is a factor R will internally use e.g. 1 for "Treatment" and 2 for "Control" and will create a list that maps the value (1 or 2) to the factor *levels* ("Treatment" or "Control"). This behavior is not necessarily very good, mostly for user-side issues. For example, if you have simple typo in your column, like "Cnotrol", in one cell, this will create a new level, which you may not notice unless you look for it. In fact, modern variants like tibbles do not do this conversion of strings to factors automatically.

```
example_df <- data.frame(text = c("Word1", "Word2"), numbers = c(1,2))
```

Look at this table

```
example_df
```

```
##    text numbers
## 1 Word1       1
## 2 Word2       2
```

Compare with a tibble:

```
example_tibl <- tibble(text = c("Word1", "Word2"), numbers = c(1,2))
```

```
example_tibl
```

```
## # A tibble: 2 x 2
##   text  numbers
##   <chr>   <dbl>
## 1 Word1       1
## 2 Word2       2
```

Explore the structure using the str() function. str prints the columns (prefixed by $) and their type as well as the first couple of values in each column.

```
str(example_df)
```

```
## 'data.frame':    2 obs. of  2 variables:
##  $ text   : Factor w/ 2 levels "Word1","Word2": 1 2
##  $ numbers: num  1 2
```

```
str(example_tibl)
```

```
## tibble [2 x 2] (S3: tbl_df/tbl/data.frame)
##  $ text   : chr [1:2] "Word1" "Word2"
##  $ numbers: num [1:2] 1 2
```

What is the difference?

Syntax

Syntax in programming is how to actually write the programming language you are using. Unfortunately, R is quite old, and the syntax is far from modern. I will largely avoid using "base R" (how R works by default) for most of the course. As mentioned in the beginning of this document R has been modernized and this is collected in the tidyverse family. A key addition was a "pipe" functionality. Those of you familiar with bash may know "|" for bash piping. R pipe works the same way. For those not familiar with pipes, look at the code below.

This code does the following:

Generate a tibble. Make this tibble long and store in long_tbl. Then it is filtered for only Day1 and only the column Measurement is kept.

```
wide_tbl <- tibble("Day1" = c(1, 2, 5, 6, 71, 23, 5, 61, 5, 6, 71,34),
                   "Day2" = c(3, 8, 2, 6, 69, 10, 3, 44, 2, 8, 99, 102),
                   "Day3" = c(103, 5, 8, 55, 7, 6, 9, 9, 0, 3, 44, 0))
long_tbl <- pivot_longer(wide_tbl,
                         cols = contains("Day"),
                         names_to = "Day",
                         values_to = "Measurement")
long_tbl <- filter(long_tbl, Day == "Day1")
long_tbl <- select(long_tbl, Measurement)
long_tbl
```

```
## # A tibble: 12 x 1
##    Measurement
##          <dbl>
##  1           1
##  2           2
##  3           5
##  4           6
##  5          71
##  6          23
##  7           5
##  8          61
##  9           5
## 10           6
## 11          71
## 12          34
```

As you can see, a lot of times we are doing something to long_tbl and overwrite long_tbl with the result. Would it not be nice to do this more concise?

The pipe looks like this: %>% and can be placed in Rstudio with ctrl+shift+m or cmd+shift+m on macs. The pipe will take whatever is the output of what is to the left side of it and use it as the *first argument* of the next function (= the one on the right). If you want to explicitly specify where in the next function the output from the left side should go you can use "." to refer to the piped object in the next function.

This makes code easier to read. In addition, depending on the functions used it reads almost like a sentence.

```
long_tbl2 <- wide_tbl %>%
  pivot_longer(cols = contains("Day"),
               names_to = "Day",
               values_to = "Measurement") %>%
  filter(Day == "Day1") %>%
  select(Measurement)

long_tbl2
```

```
## # A tibble: 12 x 1
##    Measurement
##          <dbl>
##  1           1
##  2           2
##  3           5
##  4           6
##  5          71
##  6          23
##  7           5
##  8          61
##  9           5
## 10           6
## 11          71
## 12          34
```

To expand on the tidyverse a little: the tidyverse packages, and especially the pipe from the "magrittr" package sort of re-invent the R-languange and modernize the syntax a lot.

Below are some examples of how the pipe translates to base R syntax. This can be useful for users familiar with base R but not with the tidyverse.

We experiment with the iris dataset. Take a minute to read what is in this dataset.

```
?iris
```

```
data("iris")
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Column selection

Base R, these return vectors without names

```
iris$Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

```
iris[['Sepal.Length']]
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

Tidyverse variants

Using the dot

```
iris %>%
  .$Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

Using the "exposing pipe" (%$%)

```
iris %$%
  Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

Using dplyr select. This actually returns something different from the methods above. What is it?

```
iris %>%
  dplyr::select(Sepal.Length) %>%
  head(10)
```

```
##    Sepal.Length
## 1           5.1
## 2           4.9
## 3           4.7
## 4           4.6
## 5           5.0
## 6           5.4
## 7           4.6
## 8           5.0
## 9           4.4
## 10          4.9
```

This is a non-exhaustive set of examples for column selection.

Grouping

dplyr also provides ways to group data and perform grouped operations. This makes computing summaries etc extremely easy. Assume you want to calculate the mean of sepal length by species in the iris dataset

```
iris %>%
  group_by(Species) %>%
  summarise(mean_sepal_length = mean(Sepal.Length))
```

Species	mean_sepal_length
setosa	5.006
versicolor	5.936
virginica	6.588

You can also summarise all columns with the same function. Assume you are only interested in Sepals and want to calculate their average length and width per species.

```
iris %>%
  group_by(Species) %>%
  dplyr::select(starts_with("Sepal")) %>%
  summarise_all("mean")
```

```
## Adding missing grouping variables: `Species`
```

```
## # A tibble: 3 x 3
##   Species    Sepal.Length Sepal.Width
##   <fct>             <dbl>       <dbl>
## 1 setosa             5.01        3.43
## 2 versicolor         5.94        2.77
## 3 virginica          6.59        2.97
```

Tasks

Compute the median per species for Petal length and width.

Modify the code above to compute *medians* for all *Petal* measurements

Compute the mean and sd per species of all Measurements per species

Summarize the table in a way that you get a summary of each measured trait (Sepal.Length, Sepal.Width, Petal.Length, Petal.Width) per species. The summaries should include the mean and the sd, as shown in the table below. The formatting is not important.

A per-species summary
Remember tidy datasets.

Trait	mean	sd
setosa		
Petal.Length	1.462	0.1736640

Petal.Width	0.246	0.1053856
Sepal.Length	5.006	0.3524897
Sepal.Width	3.428	0.3790644
versicolor		
Petal.Length	4.260	0.4699110
Petal.Width	1.326	0.1977527
Sepal.Length	5.936	0.5161711
Sepal.Width	2.770	0.3137983
virginica		
Petal.Length	5.552	0.5518947
Petal.Width	2.026	0.2746501
Sepal.Length	6.588	0.6358796
Sepal.Width	2.974	0.3224966

Visualizations

I recommend you ignore the base R plotting functions and instead understand ggplot2 .

Read this chapter: https://r4ds.had.co.nz/data-visualisation.html

Then use the iris dataset.

```
data("iris")
```

```
head(iris)
```

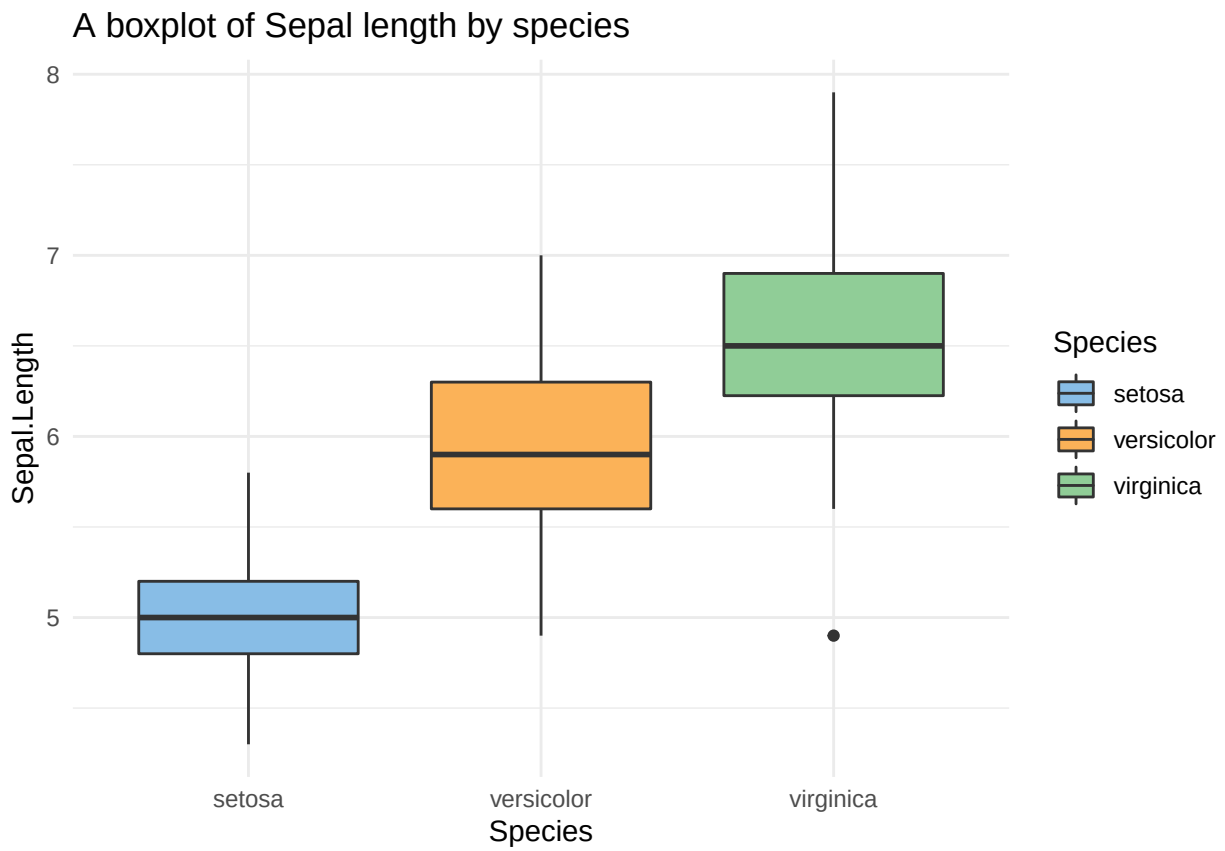
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Tasks

Boxplot of sepal length

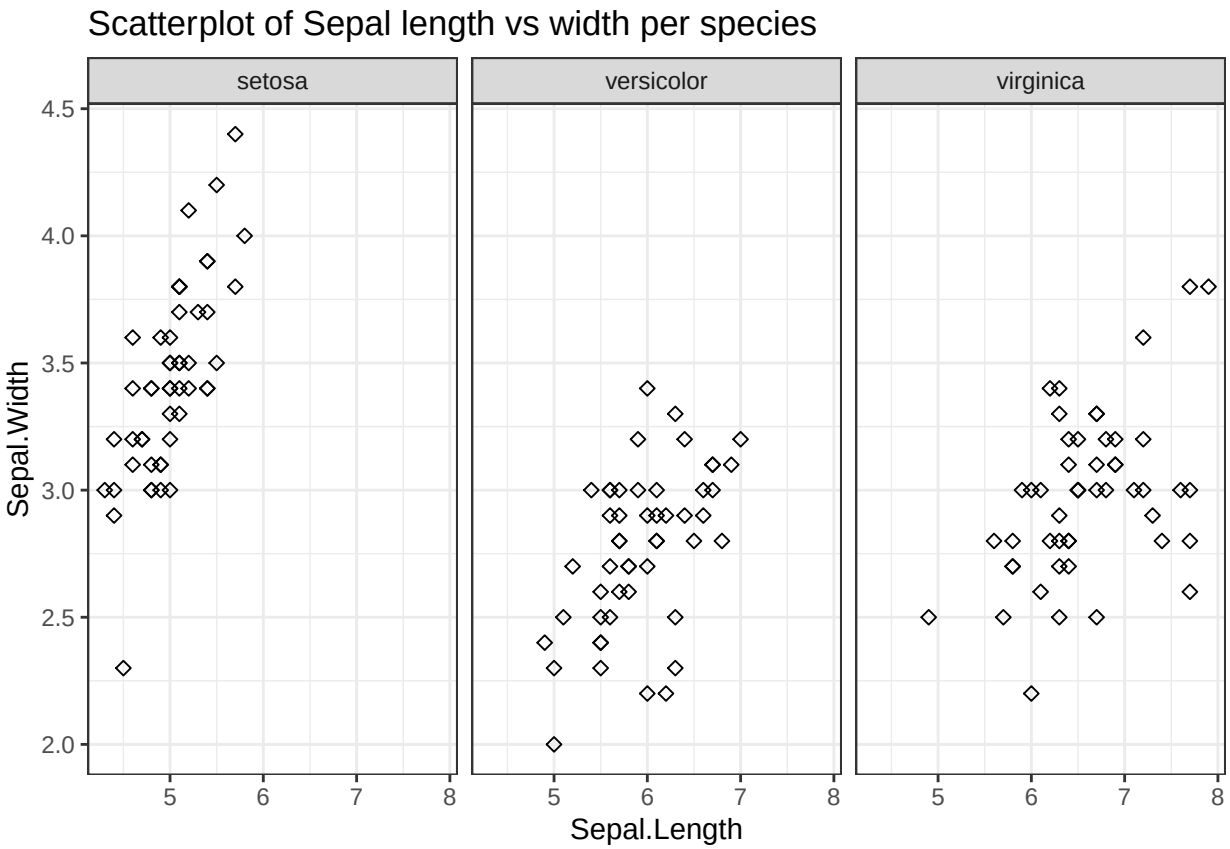
Produce a boxplot of Sepal length by species (Species on x and values on y). Explore the use of colors.

You should be able to produce something like this. Your plot may have different colors, but should show the same things.



Sepal Scatter plot with facetting

Next, create a scatter plot of Sepal.Length (x) versus Sepal.Width (y). Use facetting to produce one plot containing one subpanel per species. Again, your plot may look different.



Free plot

Use your creativity to create at least one more plot using a *different geom* (not boxplot or point).

Patchwork

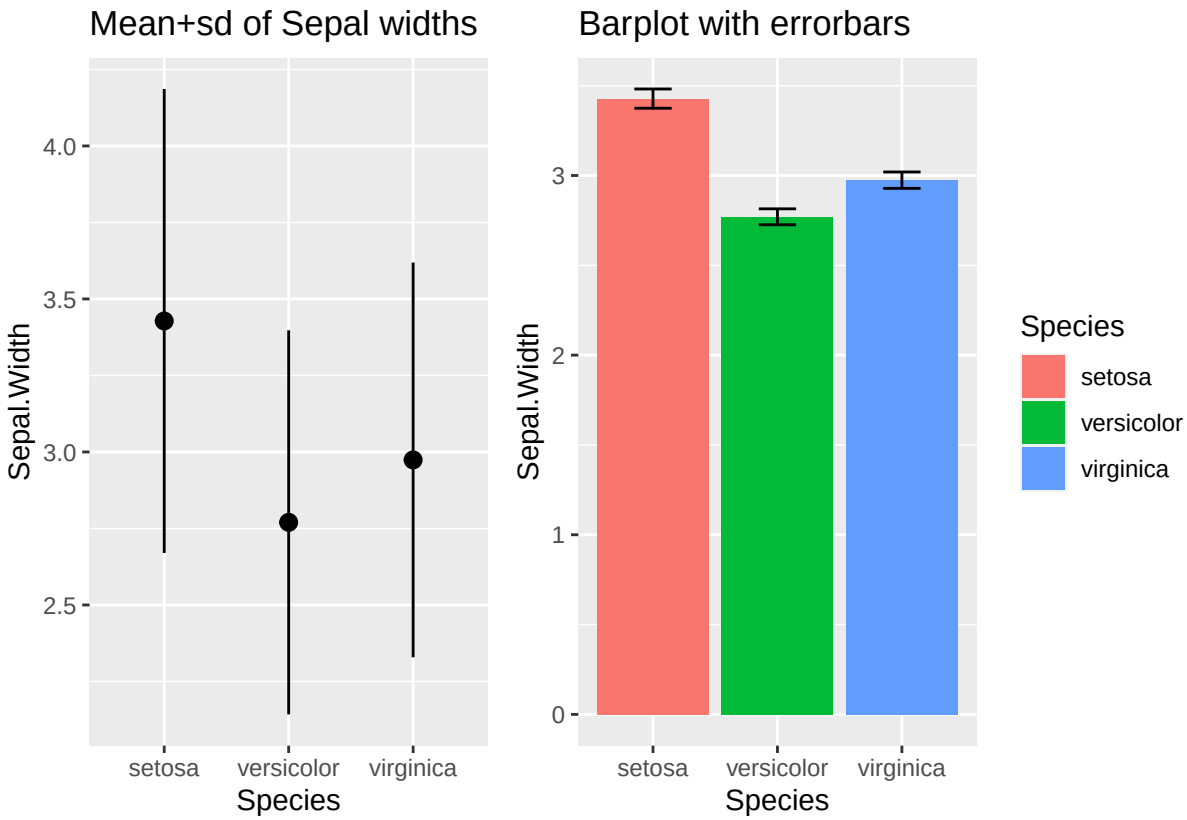
This is a rather recent package. It takes on the task of adding multiple plots into one Figure. This is notoriously hard and annoying. Patchwork builds on the ggplot philosophy of adding things.

Below I create 3 ggplots (gg1, gg2, gg3) and put them together with patchwork.

```
gg1 <- iris %>%
  ggplot(aes(x = Species, y = Sepal.Width)) +
  stat_summary(fun.data = "mean_sdl") +
  ggtitle("Mean+sd of Sepal widths")
gg2 <- iris %>%
  ggplot(aes(x = Species, y = Sepal.Width)) +
  stat_summary(geom = "bar", aes(fill = Species), fun.data = "mean_se") +
  stat_summary(geom = "errorbar", width =0.3, fun.data = "mean_se") +
  ggtitle("Barplot with errorbars")
gg3 <- iris %>%
  ggplot(aes(x = Petal.Width, y = Sepal.Width)) +
  geom_point(aes(color = Species)) +
  ggtitle("Dotplot of Petal width vs Sepal width")
```

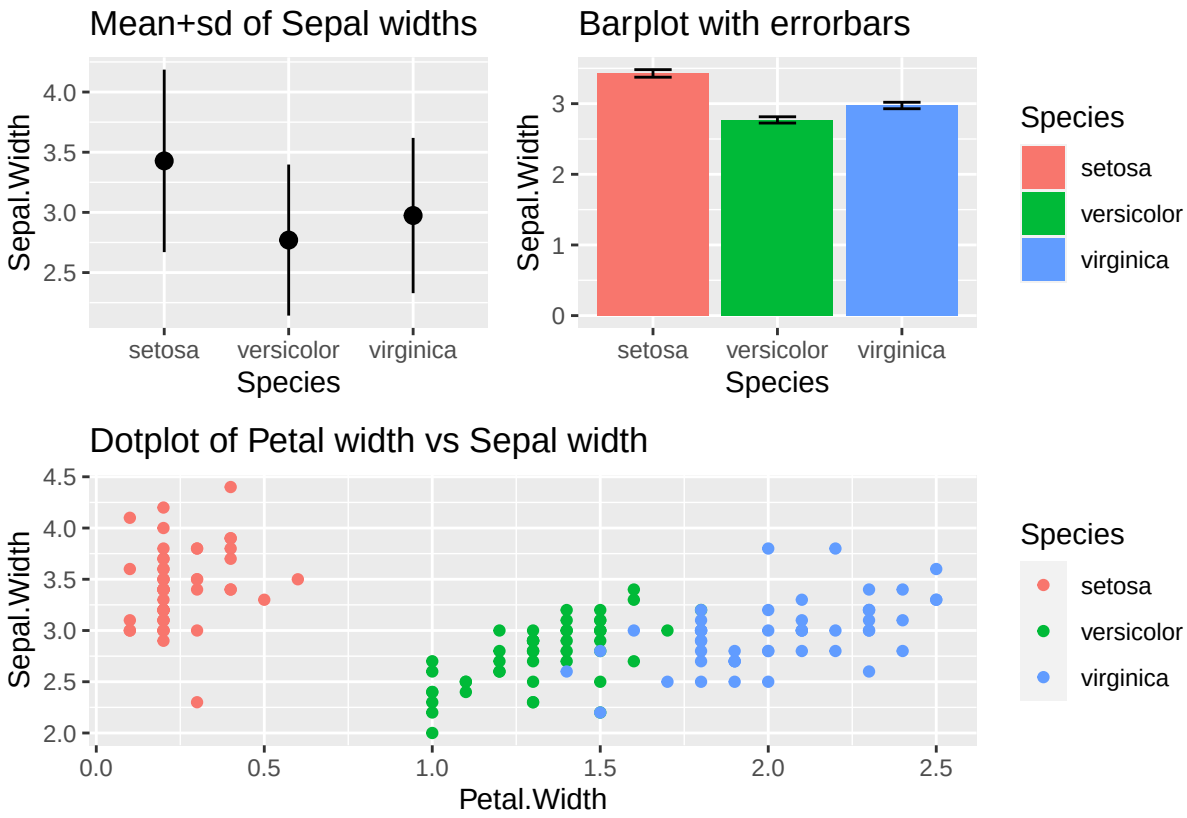
Side by side placement

```
gg1 + gg2
```



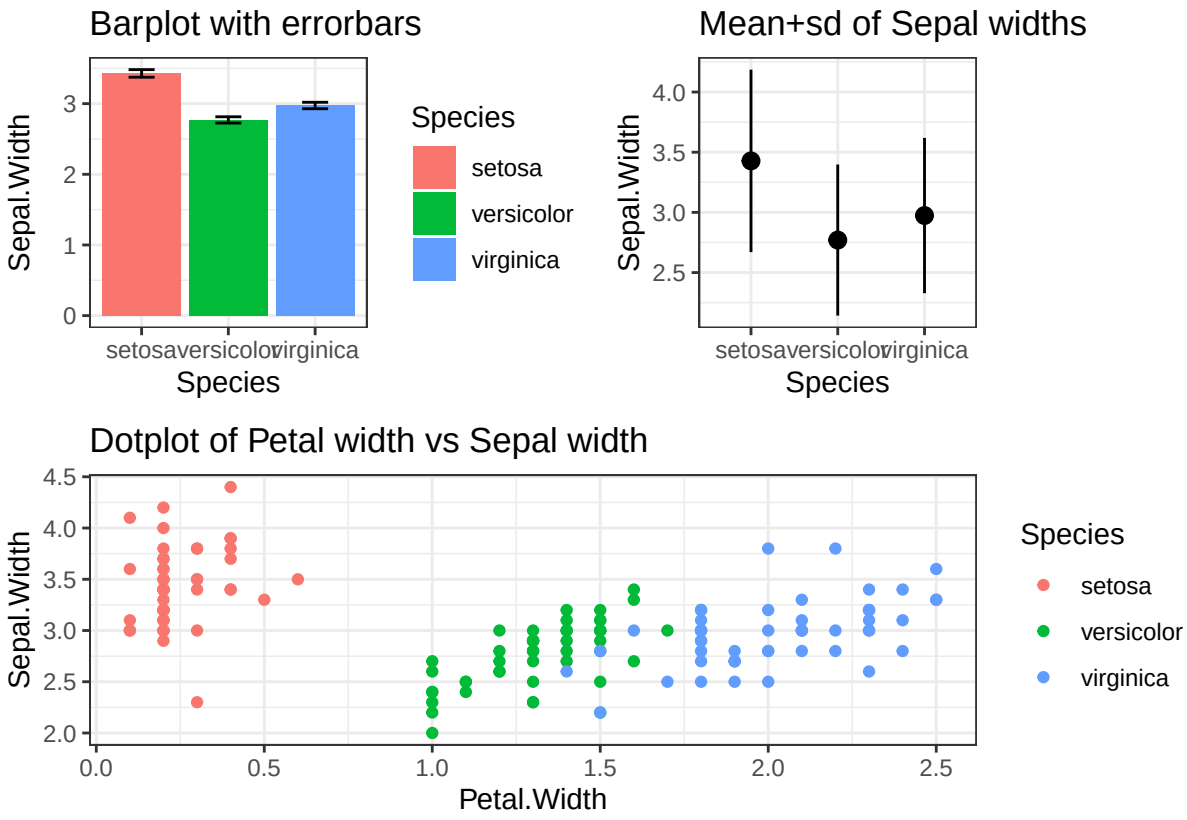
Placement of plots above / below

```
(gg1 + gg2) /
gg3
```



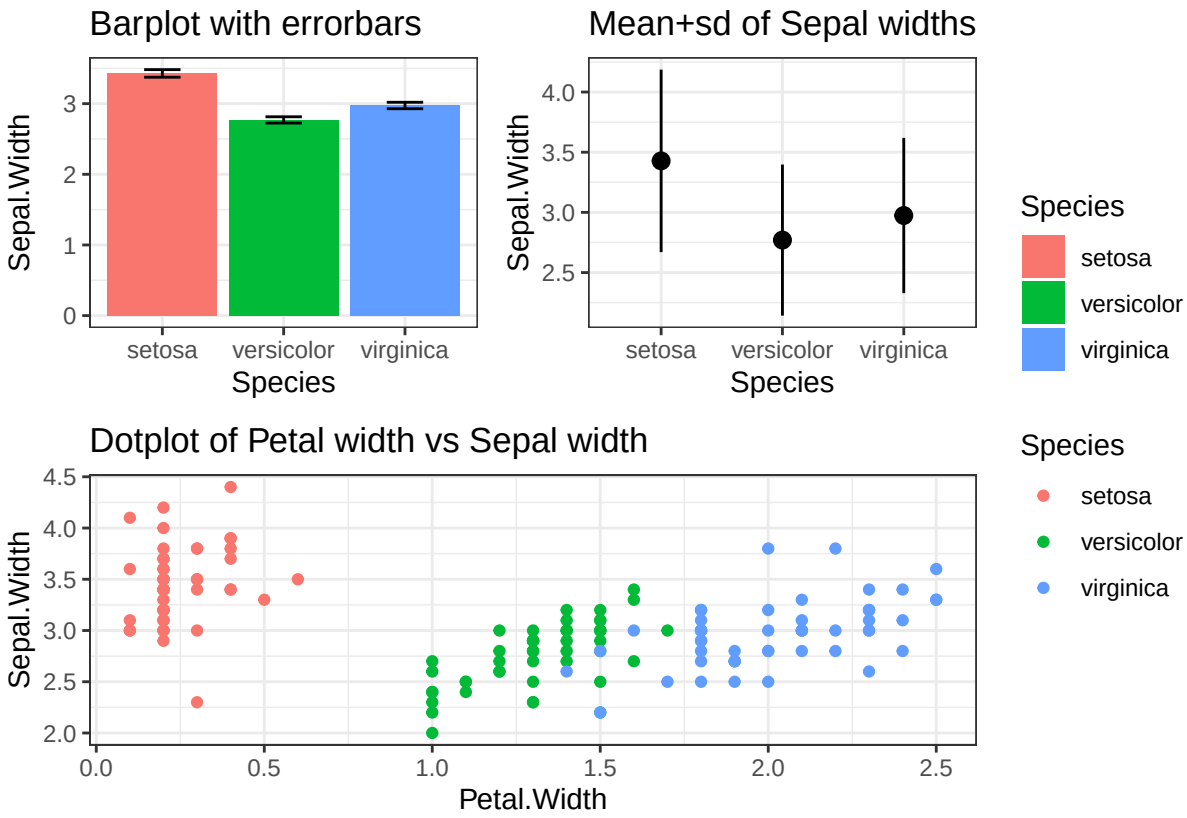
The whole patchwork of plots can be styled using "&" to connect things. Below we restyle the whole plot to use a different ggplot theme.

```
((gg2 + gg1) /
gg3) & theme_bw()
```



Collect those legends into one area:

```
((gg2 + gg1) /
gg3) + plot_layout(guides = 'collect') & theme_bw()
```



For more documentation, check https://patchwork.data-imaginist.com/

Task

Combine the plots you created during earlier (boxplot, scatter plot and your own plot) into a visually appealing figure.