Topics in data analytics

Intro to R

Bruna Wundervald

Contents

Chapter 1

Getting familiar with R & RStudio

1.1 Projects

Chapter 2

The `tidyverse`

The `tidyverse` is a collection o `R` packages designed to make data science as easy and clear as possible. It is mostly maintained by people who work at RStudio, but it also receives many collaborations from the community.

The term "collection" means that when you load the `tidyverse`, you actually load many packages at the same time, which are

- **`dplyr`: data manipulation**
- `tidyr`: tidy data
- **`ggplot2`: grammar of graphics**
- `readr`: reading rectangular data
- **`stringr`: functions for string data**
- **`forcats`: functions for factor data**
- **`purrr`: functional programming**
- `tibble`: modern data.frames

Such packages are built to work cohesively together. To install and load the `tidyverse`, just run

```r
install.packages("tidyverse")
library(tidyverse)
```

The `tidyverse` ecossystem intends to provide a toolkit for the full data science workflow in `R`, illustrated in Figure 1. The above mentioned packages are not the only ones in the ecossystem. In reality, there is a big tidy-wave happening now in the `R` community, so there are numerous new packages coming up everyday that work with and follow the same philosophy of the `tidyverse`.

The open book "R for Data Science", written by Hadley Wickham & Garrett Grolemund, is available and it can be really interesting for learning more about the `tidyverse` in practice and its philosophy.

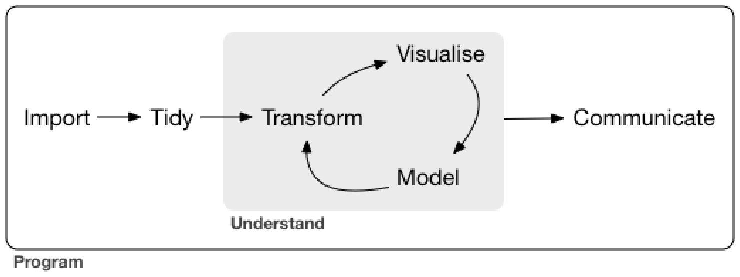


Figure 2.1: Data science workflow: from importing data to communication

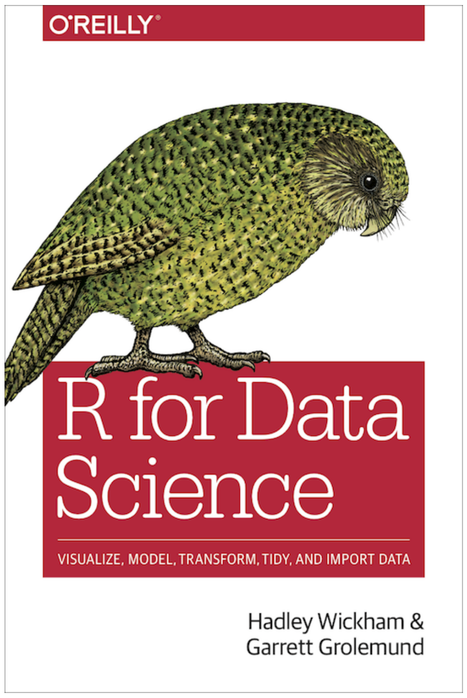


Figure 2.2: Book: R for Data Science, Hadley Wickham & Garrett Grolemund

Figure 2.3: The ‘pipe‘ operator in the ‘magrittr‘ package sticker.

2.1 The `pipe` operator

A key component of the `tidyverse` ecossystem is the `pipe` (or `%>%`) operator. The operator works by applying to the object that is on its left side, the function that comes on its right side. For example, we can find the mean of a vector by doing

```r
1:20 %>% mean()
```

```
[1] 10.5
```

This is useful because we can chain many operation layers in a logical order (the order that they actually happen) using the `%>%`, which makes the code much easier to write for who's writing it, and much more readable for who's trying to understand it. In the following we will see many more examples of the `%>%` being used.

Chapter 3

Reading data

In this section we will briefly approach a few ways of reading rectangular data into `R`. This task is usually accomplished with base/native functions and, when the data files are a bit more complex, with the help of other packages. In general data science, some common data file extensions are:

- .xlsx (Excel file)
- .csv (comma-separated files)
- .txt (text files)

3.1 the data is available at [etc]

We will examine a dataset on M&M sweets. The color counts and net weight (in grams) for a sample of 30 bags of M&M was recorded. The advertised net weight per bag is 47.9 grams.

3.2 Reading .xlsx files

```
mm1 <- readxl::read_xlsx("data/MandM.xlsx")
head(mm1)
```

```
# A tibble: 6 x 7
    Red Green  Blue Orange Yellow Brown Weight
  <dbl> <dbl> <dbl>  <dbl>  <dbl> <dbl>  <dbl>
1    15     9     3      3      9    19   49.8
2     9    17    19      3      3     8   49.0
```

```
3     14      8      6       8      19      4   50.4
4     15      7      3       8      16      8   49.2
5     10      3      7       9      22      4   47.6
6     12      7      6       5      17     11   49.8
```

3.3 Reading .csv files

```
mm2 <- read_csv("data/MandM.csv")
head(mm2)
```

```
# A tibble: 6 x 7
    Red Green  Blue Orange Yellow Brown Weight
  <dbl> <dbl> <dbl>  <dbl>  <dbl> <dbl>  <dbl>
1    15     9     3      3      9    19   49.8
2     9    17    19      3      3     8   49.0
3    14     8     6      8     19     4   50.4
4    15     7     3      8     16     8   49.2
5    10     3     7      9     22     4   47.6
6    12     7     6      5     17    11   49.8
```

3.4 Reading .txt files

```
mm3 <- read_table2("data/MandM.txt", col_names = FALSE)
names(mm3) <- c("Red", "Green", "Blue", "Orange",
                "Yellow", "Brown", "Weight")
head(mm3)
```

```
# A tibble: 6 x 7
    Red Green  Blue Orange Yellow Brown Weight
  <dbl> <dbl> <dbl>  <dbl>  <dbl> <dbl>  <dbl>
1    15     9     3      3      9    19   49.8
2     9    17    19      3      3     8   49.0
3    14     8     6      8     19     4   50.4
4    15     7     3      8     16     8   49.2
5    10     3     7      9     22     4   47.6
6    12     7     6      5     17    11   49.8
```

They all look the same!

And this is the summary of this dataset:

```r
summary(mm3)
```

```
      Red            Green            Blue            Orange      
 Min.   : 3.0   Min.   : 2.0   Min.   : 1.000   Min.   : 0.000  
 1st Qu.: 6.5   1st Qu.: 6.0   1st Qu.: 4.000   1st Qu.: 4.000  
 Median : 9.0   Median : 7.0   Median : 6.500   Median : 6.000  
 Mean   : 9.6   Mean   : 7.4   Mean   : 7.233   Mean   : 6.633  
 3rd Qu.:12.0   3rd Qu.: 9.0   3rd Qu.: 9.750   3rd Qu.: 9.000  
 Max.   :20.0   Max.   :17.0   Max.   :19.000   Max.   :13.000  
     Yellow           Brown           Weight     
 Min.   : 3.00   Min.   : 4.00   Min.   :46.22  
 1st Qu.: 8.25   1st Qu.: 8.00   1st Qu.:48.29  
 Median :13.50   Median :12.50   Median :49.07  
 Mean   :13.77   Mean   :12.47   Mean   :49.21  
 3rd Qu.:18.00   3rd Qu.:17.75   3rd Qu.:50.17  
 Max.   :26.00   Max.   :20.00   Max.   :52.06  
```

Chapter 4

Data cleaning & manipulation with `dplyr`

The `dplyr` package is arguably the most important/useful one of the `tidyverse` group of packages. It provides us with the "Grammar of Manipulation" that make the most common data manipulation tasks easy & consistently written. The term "Grammar" comes from the fact that the `tidyverse` functions, in general, are built to work as "verbs" in a sentence.

4.1 Manipulation

The core functions of the `dplyr` are:

- `mutate()`: adding and changing variables
- `select()`: pick or drop variables
- `filter()`: filter the data according to some defined rule
- `summarise`: reduces the data to a single value (such as the mean, maximum and minimum, etc)
- `arrange()`: ordering data
- `group_by()`: perform the above operations *taking into account* one or more grouping variables

These functions, again, comprise basically the main data manipulation tasks we always end up doing when dealing with a new dataset. In the following, we have a few examples of the functions in action:

```
# Finding summary measures of the package weight
mm3 %>%
  summarise(mean_weight = mean(Weight),
            sd_weight = sd(Weight)) %>%
  mutate(lb_mean = mean_weight - 1 * sd_weight,
         ub_mean = mean_weight + 1 * sd_weight)
```

```
# A tibble: 1 x 4
  mean_weight sd_weight lb_mean ub_mean
        <dbl>     <dbl>   <dbl>   <dbl>
1        49.2      1.52    47.7    50.7
```

If the advertised net weight per bag is 47.9 grams, can we easily test if the data agrees with that?

```
mm3 %>%
    summarise(mean_weight = mean(Weight),
              sd_weight = sd(Weight),

              # We can calculate the p-value of a t-test for the
              # mean being equal to 47.9 and save it as a
              # summary variable:
              p.value = t.test(Weight, mu = 47.9)$p.value
              ) %>%
  # Now, is this p-value lower than 0.05 (95% level of significance)?
  mutate(is_mean_equal = ifelse(p.value < 0.05,
                                "Evidence against",
                                "Evidence in favour"))
```

```
# A tibble: 1 x 4
  mean_weight sd_weight   p.value is_mean_equal
        <dbl>     <dbl>     <dbl> <chr>
1        49.2      1.52 0.0000535 Evidence against
```

We had checked before that the orange M&M's were less common. Could that be related with a change in the mean weight?

```
# We can either separate the data:
mm3 %>%
  filter(Orange <= 4) %>%
      summarise(mean_weight = mean(Weight),
                sd_weight = sd(Weight),
                p.value = t.test(Weight, mu = 47.9)$p.value)
```

```
# A tibble: 1 x 3
  mean_weight sd_weight p.value
        <dbl>     <dbl>   <dbl>
1        49.4      1.66  0.0287
```

```
mm3 %>%
  filter(Orange > 4) %>%
      summarise(mean_weight = mean(Weight),
                sd_weight = sd(Weight),
                p.value = t.test(Weight, mu = 47.9)$p.value)
```

```
# A tibble: 1 x 3
  mean_weight sd_weight p.value
        <dbl>     <dbl>   <dbl>
1        49.1      1.50 0.00108
```

```
# Or create a grouping variable:

mm3 %>%
  mutate(orange_low = ifelse(Orange <= 4, "Low", "Normal")) %>%
  group_by(orange_low) %>%
  summarise(mean_weight = mean(Weight),
            sd_weight = sd(Weight),
            p.value = t.test(Weight, mu = 47.9)$p.value)
```

```
# A tibble: 2 x 4
  orange_low mean_weight sd_weight p.value
  <chr>            <dbl>     <dbl>   <dbl>
1 Low               49.4      1.66 0.0287
2 Normal            49.1      1.50 0.00108
```

```
# Or use some other heuristic, such as filtering by the 10 lowest
# orange values

mm3 %>%
  arrange(Orange) %>%
  slice(1:10) %>%
  summarise(mean_weight = mean(Weight),
            sd_weight = sd(Weight),
            p.value = t.test(Weight, mu = 47.9)$p.value)
```

```
# A tibble: 1 x 3
  mean_weight sd_weight p.value
        <dbl>     <dbl>   <dbl>
1        49.4      1.57  0.0138
```

Those are more or less simple operations to find some statistics about the data. However, the `dplyr` has its own website where many more information and resources can be found: https://dplyr.tidyverse.org/

Besides that, the `dplyr` functions also have many handy extensions such as the:

- `_all()`: performs an operation is all columns
- `_at`: performs an operation at some specific columns
- `_if` performs an operation *if* a condition is satisfied

4.2 Cleaning

Very frequently, before actually starting our analysis, we first need to clean our data. This means we need to fix possible errors, adding or removing information, formatting it, always trying to be the most thorough possible in order to avoid. Let us consider now a second dataset from a typical clinical trial with continuous and characted variables, that will require a bit more of cleaning. A few other packages will be used in combination with the `dplyr`, especially the `stringr` and `forcats`.

```r
# Just reading the data
pt <- read.table("data/Patients.csv", header = TRUE, sep = ",")

# Showing the types and first rows of the data.frame
pt %>% glimpse()
```

```
Observations: 31
Variables: 8
$ PATNO  <fct> 1, 2, 3, 4, XX5, 6, 7, , 8, 9, 10, 11, 12, 13, 14, 2, 3, ...
$ GENDER <fct> M, F, X, F, M, , M, M, F, M, f, M, M, 2, M, F, M, F, F, M...
$ VISIT  <fct> 11/11/1998, 11/13/1998, 10/21/1998, 01/01/1999, 05/07/199...
$ HR     <int> 88, 84, 68, 101, 68, 72, 88, 90, 210, 86, NA, 68, 60, 74,...
$ SBP    <int> 140, 120, 190, 200, 120, 102, 148, 190, NA, 240, 40, 300,...
$ DBP    <int> 80, 78, 100, 120, 80, 68, 102, 100, NA, 180, 120, 20, 74,...
$ DX     <fct> 1, X, 3, 5, 1, 6, , , 7, 4, 1, 4, , 1, , X, , 3, 2, , 1, ...
$ AE     <fct> 0, 0, 1, A, 0, 1, 0, 0, 0, 1, 0, 1, 0, , 1, 0, 0, 1, 0, 0...
```

```r
# Fixing the names of the columns
names(pt) <- str_to_lower(names(pt))

# Finding frequency tables per gender
pt %>%
  count(gender)
```

```
# A tibble: 6 x 2
  gender     n
  <fct>  <int>
1 ""         1
2 2          1
3 f          2
4 F         12
5 M         14
6 X          1
```

The first problem was found: the gender column is not standardized, as we have both lower and upper cases and an empty level. This is easily fixable with the help of functions from the `stringr` package:

```
pt <- pt %>%
  # Transform lower cases to upper cases in gender
  mutate(gender = str_to_upper(gender)) %>%
  # Replacing the empty or strange information for something more useful
  mutate(gender = ifelse(gender %in%
                         c("", "2", "X"), "Not Available", gender))

# View the frequency table again
pt %>%
  count(gender)
```

```
# A tibble: 3 x 2
  gender            n
  <chr>         <int>
1 F                14
2 M                14
3 Not Available     3
```

```
# Now it makes more sense!
```

However, we still can spot a few problems. For instance, the variable names `visit` was read as factor, while we now that it should be interpreted as a date. We can use the `lubridate` package to fix in situations like this.

```
# Transforming the variable into a date type
pt <- pt %>%
  mutate(visit = as.Date(visit, "%m/%d/%Y"),
         day = lubridate::day(visit),
         month = lubridate::month(visit),
         month_year = zoo::as.yearmon(visit))
```

```
# Check that it is transformed
class(pt$visit)
```

```
[1] "Date"
```

```
# How many observations we have each month?
pt %>% count(month)
```

```
# A tibble: 13 x 2
   month     n
   <dbl> <int>
 1     1     2
 2     2     1
 3     3     1
 4     4     1
 5     5     2
 6     6     2
 7     7     1
 8     8     2
 9     9     1
10    10     4
11    11     6
12    12     1
13    NA     7
```

–> Exercises: https://teachingr.com/content/the-5-verbs-of-dplyr/the-5-verbs-of-dplyr-exercise.html

Chapter 5

Plotting data with ggplot2

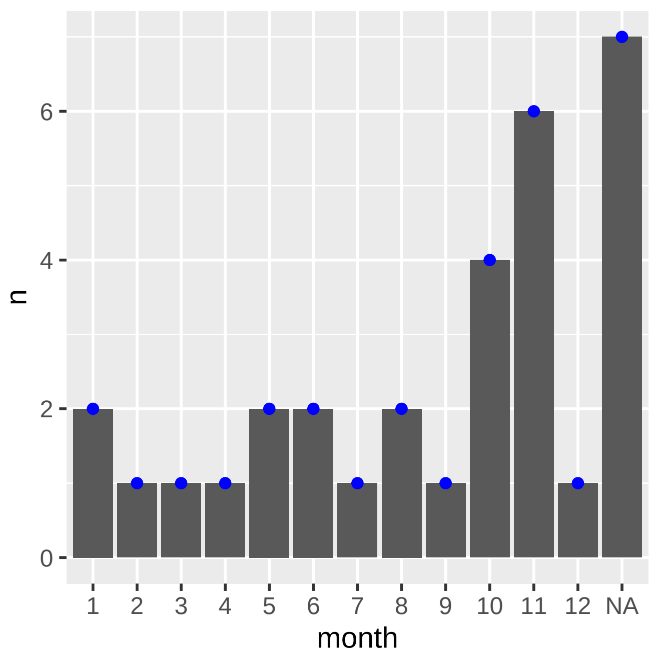
After preparing the data, we can start creating some plots to actually extract some information from it (or find even more things to be fixed). The `ggplot2` package is based on The Grammar of Graphics, and provides us with the tools to create whatever graph we need.

As defined by the creator of the package, Hadley Wickham, a plot is composed by 4 things:

- Layers: data, aesthetics (**aes**), geometric objects (**geom**), transformations, etc
- Scales
- Coordinates
- Facets

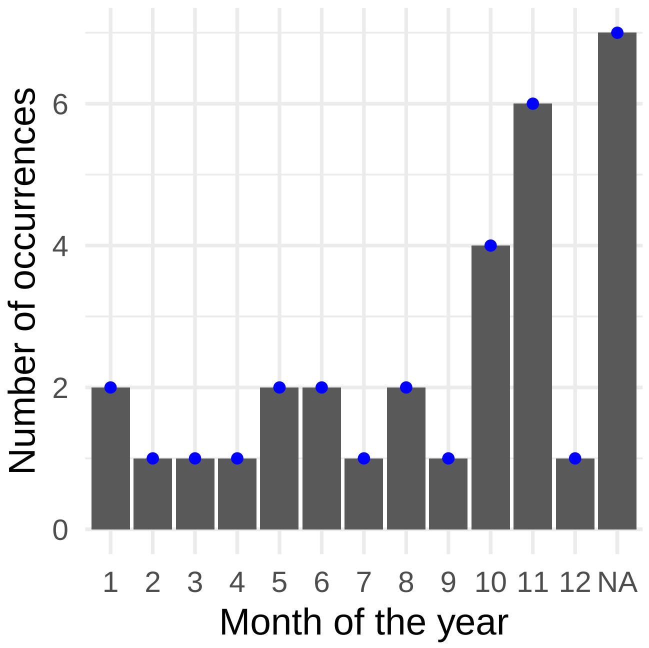
So the `ggplot2` syntax creates plots by especifically using a combination of these 4 elements. This might not make much sense now but it is actually quite easy to understand. For instance, we can create a barplot of the month count we saw above with:

```r
count_month <- pt %>%
  count(month) %>%
  mutate(month = as.factor(month))

p <- count_month %>%
  # In the aes() function we map the variables from the data
  # into the x, y and other dimensions of the plot
  ggplot(aes(x = month, y = n)) +
  # The geom_ layers define what type of graphic we want
  geom_bar(stat = "identity") +
  # The geom_ can be accumulated, by simply adding more layers to the
  # plot
  geom_point(colour = "blue")

p
```



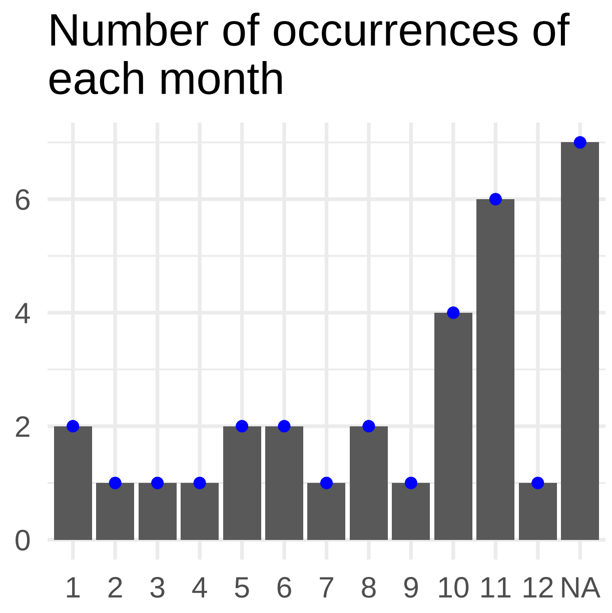
How can we make this nicer? Maybe changing colours, labels and other details:

```
p +
  # Adding the variable labelling
  labs(x = "Month of the year", y = "Number of occurrences") +
  # Using a built-in theme with font size 14
  theme_minimal(14)
```



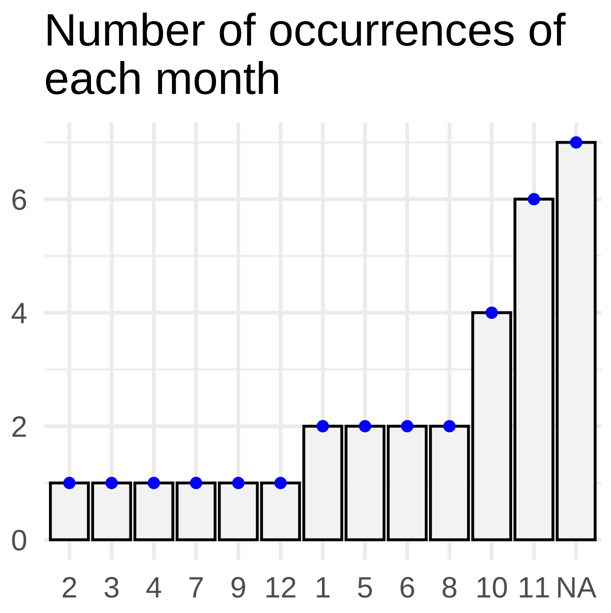
Or we can just suppress all the labs by an informative title:

```
p +
  geom_point(colour = "blue") +
  labs(title =  "Number of occurrences of \neach month",
       x = "", y = "") +
  theme_minimal(14)
```



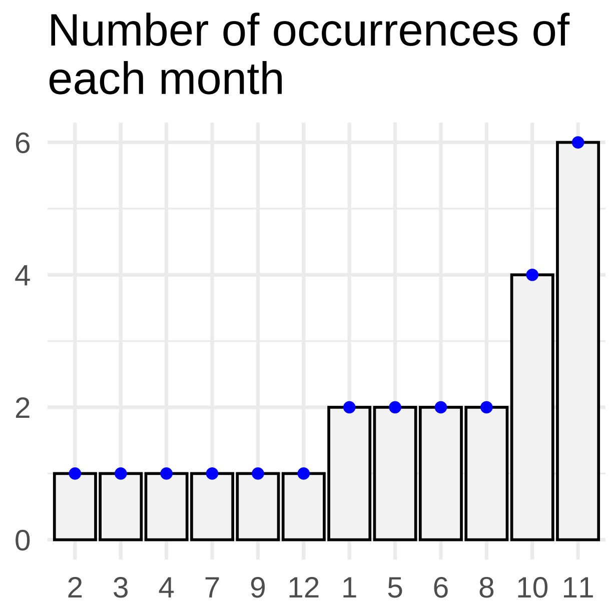
Can we change the order of the bars? (but note that it won't be in the month order anymore!)

```r
p <- count_month %>%
  mutate(month = as.factor(month)) %>%
  # In the aes() function we map the variables from the data
  # into the x, y and other dimensions of the plot
  ggplot(aes(x = reorder(month, n), y = n)) +
  # The geom_ layers define what type of graphic we want
  geom_bar(stat = "identity", colour = "black", fill = "grey95") +
  # The geom_ can be accumulated, by simply adding more layers to the
  # plot
  geom_point(colour = "blue") +
  labs(title =  "Number of occurrences of \neach month",
       x = "", y = "") +
  theme_minimal(14)
p
```



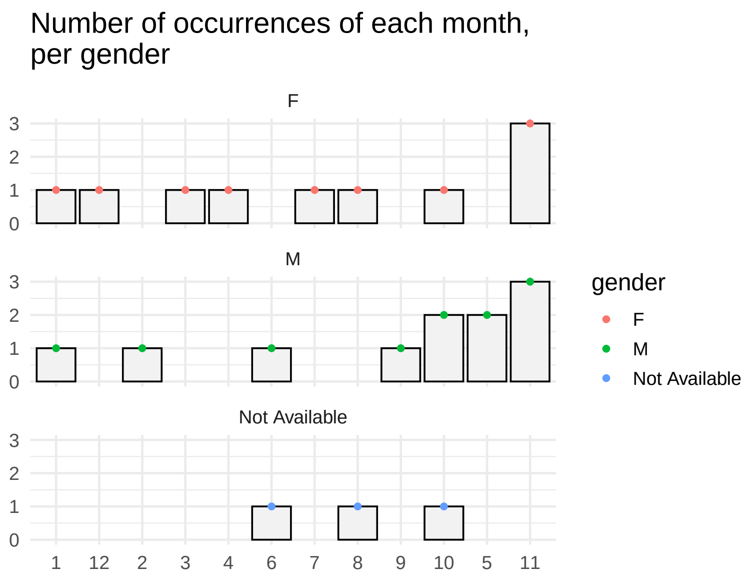
The NA class is the most frequent one, but we don't know what it means. We can just remove it, for example:

```r
p <- count_month %>%
  mutate(month = as.factor(month)) %>%
  na.omit() %>%
  # In the aes() function we map the variables from the data
  # into the x, y and other dimensions of the plot
  ggplot(aes(x = reorder(month, n), y = n)) +
  # The geom_ layers define what type of graphic we want
  geom_bar(stat = "identity", colour = "black", fill = "grey95") +
  # The geom_ can be accumulated, by simply adding more layers to the
  # plot
  geom_point(colour = "blue") +
  labs(title =  "Number of occurrences of \neach month",
       x = "", y = "") +
  theme_minimal(14)
p
```



Could we make the same plot per gender?

```r
count_month_gender <- pt %>%
  group_by(gender) %>%
  count(month)  %>%
  mutate(month = as.factor(month)) %>%
  na.omit()

p <- count_month_gender %>%
  # In the aes() function we map the variables from the data
  # into the x, y and other dimensions of the plot
  ggplot(aes(x = reorder(month, n), y = n)) +
  # The geom_ layers define what type of graphic we want
  geom_bar(stat = "identity", colour = "black", fill = "grey95") +
  # The geom_ can be accumulated, by simply adding more layers to the
  # plot
  facet_wrap(~gender, ncol = 1) +
  geom_point(aes(colour = gender)) +
  labs(title =  "Number of occurrences of each month, \nper gender",
       x = "", y = "") +
  theme_minimal(14)

p
```



–> Exercises: https://r4ds.had.co.nz/data-visualisation.html#exercises

Chapter 6

Functional Programming

6.1 Modelling

Chapter 7

Time Series

Chapter 8

Text Mining

Chapter 9

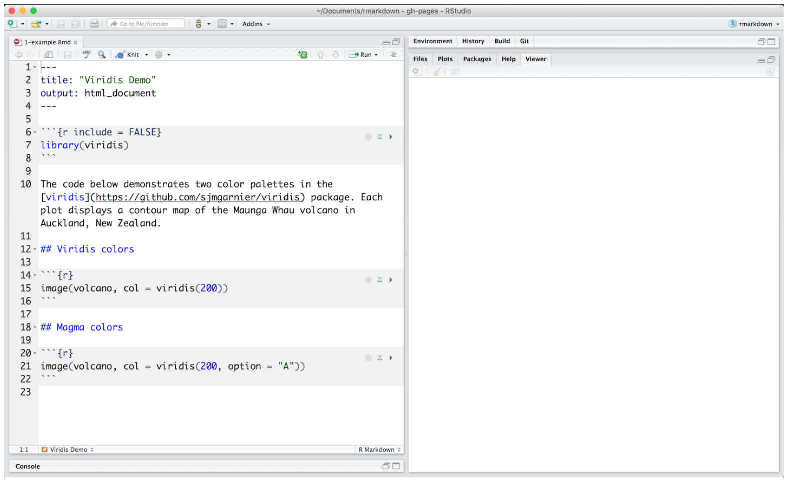
`R markdown`

An `R Markdown` file is a plain text file that has the extension .Rmd, and is a full authoring framework for data science. You can use a single `R Markdown` file to:rmark

- save and execute code
- generate high quality reports (papers, products, etc)

An `.Rmd` file contains the elements:

- An (optional) YAML header surrounded by `---`s
- R code chunks surrounded by “`s (also works for other languages like`Python`and`Julia‘)
- Text mixed with figures, code, etc
- Titles ad subtitles created by hashtags
- The results of code chunks
- Latex, equations, bold and italic text (a complete formatting tool)



~~~
---
title: "Viridis Demo"
output: html_document
---

```{r include = FALSE}
library(viridis)
```

The code below demonstrates two color palettes in the
[viridis](https://github.com/sjmgarnier/viridis) package. Each
plot displays a contour map of the Maunga Whau volcano in
Auckland, New Zealand.

## Viridis colors

```{r}
image(volcano, col = viridis(200))
```

## Magma colors

```{r}
image(volcano, col = viridis(200, option = "A"))
```

~~~

Figure 9.1: R Markdown

Chapter 10

References