Illustration: Amrei Binzer-Panchal

Basic Biostatistics and Bioinformatics

Session 1: Introduction to R

Swedish University of Agricultural Sciences, Alnarp

13 November 2023

Basic Biostatistics and Bioinformatics

A seminar series on fundamentals

Organised by *SLUBI* and *Statistics at SLU*

Presentation of background and a practical exercise

Upcoming topics

- 27 November. Linux Basics
- 11 December. PCA
- 15 January. Introduction to Markdown
- 29 January Population Structure

Topic suggestions are welcome

SLUBI

- SLU bioinformatics center
- Weekly online drop-in (Wednesdays at 13.00)
- slubi@slu.se, https://www.slubi.se
- Alnarp: Lizel Potgieter (Dept. of Plant Breeding)

Statistics at SLU

- SLU statistics center
- Free consultations for all SLU staff
- statistics@slu.se
- Alnarp: Jan-Eric Englund and Adam Flöhr (Dept. of Biosystems and Technology)

☰

Today’s Presentation

R & RStudio

Data handling

- Filter
- Select
- Transform

Plotting

Some RStudio features

Troubleshooting

Exercise session

Introduction

Open software used for statistical analysis

Code-based interface

1976	S created by John Chambers, Bell Labs
1991	R created by R. Gentleman and R. Ithaka, University of Aukland, NZ
1995	Goes under free license. First public release
2000	First stable beta release, R 1.0.0
2004	Release of R 2.0.0
2013	Release of R 3.0.0
2020	Release of R 4.0.0

R & RStudio

R itself is the programming language

The installation of the language comes with a basic interface

Most users work in more advanced interfaces, the most common of which is *RStudio*

Installation of R & RStudio

R. https://www.r-project.org/

CRAN in left frame > Select any mirror > Select depending on operating system

RStudio. https://posit.co/

Products > Select open version > Select RStudio Desktop > Select free version

Base R & Packages

R is extended by creating new functions

Collections of functions are called *packages*

The basic installation comes with a set of packages

New packages can be installed from CRAN (the *Comprehensive* R *Archive Network*)

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

Working in R. Objects

In R, information is stored as named *objects*

Objects are stored by writing a name followed by the *assign arrow* `<-`

Objects can be very simple, like a single number

```r
x <- 3
x # Simple writing the name of an object will print it
```

```
[1] 3
```

A bit more complex, like a *vector* (an ordered sequence of numbers)

```r
x <- c(3, 14, 159)
x
```

```
[1]   3  14 159
```

And more complex still, like a collection of vectors in a *data frame*

```r
dat <- data.frame(x = c(1,2,3), y = c("a", "b", "c"))
dat
```

```
  x y
1 1 a
2 2 b
3 3 c
```

Functions

Objects are created and changed using *functions*

Function take some input and produce an output

Functions typically have a set of arguments, allowing the user to control its behaviour

Called using the function name followed by the input and arguments in brackets

```r
x <- c(3, 14, 159)
sum(x) # Calculates the sum of the vector
```

```
[1] 176
```

The output of a function can be printed directly or stored as a new object

```r
s <- sum(x)
s
```

```
[1] 176
```

NA is used for missing values

```r
x <- c(3, 14, 159, NA)
sum(x)
```

```
[1] NA
```

```r
sum(x, na.rm = TRUE) # Setting the argument na.rm to TRUE removes the NAs
```

```
[1] 176
```

Code structure and piping

Many different ways to structure code with several steps

Simple example: (i) given a vector, (ii) transform with the logarithm, (iii) take the sum.

1. Store each step and use in next line

```r
dat <- c(3, 14, 159)
dat <- log(dat)
sum(dat)
```

```
[1] 8.806574
```

2. Use functions within functions

```r
sum(log(c(3, 14, 159)))
```

```
[1] 8.806574
```

3. Use piping (`%>%` or `|>`) to send output into the next function

```r
c(3, 14, 159) %>%
  log() %>%
  sum()
```

```
[1] 8.806574
```

The pipe takes the output of the left function and sends it as input to the right function

RStudio shortcut: ctrl + shift + M

Data import

Base R and add-on packages include multiple functions to import data

The specific choice of function depends on the data type

The file path can be specified relative a *working directory* - the base folder of the current R session

```r
dat_tv <- read.table("Data/IMDb_Economist_tv_ratings.csv", header = T, sep = ",", dec = ".")
```

```r
library(readxl)
dat_tv <- read_excel("Data/IMDb_Economist_tv_ratings.xlsx")
dat_tv
```

```
# A tibble: 2,266 × 7
   titleId   seasonNumber title       date                av_rating share genres
   <chr>            <dbl> <chr>       <dttm>                  <dbl> <dbl> <chr>
 1 tt2879552            1 11.22.63    2016-03-10 00:00:00      8.49  0.51 Drama…
 2 tt3148266            1 12 Monkeys  2015-02-27 00:00:00      8.34  0.46 Adven…
 3 tt3148266            2 12 Monkeys  2016-05-30 00:00:00      8.82  0.25 Adven…
 4 tt3148266            3 12 Monkeys  2017-05-19 00:00:00      9.04  0.19 Adven…
 5 tt3148266            4 12 Monkeys  2018-06-26 00:00:00      9.14  0.38 Adven…
 6 tt1837492            1 13 Reasons… 2017-03-31 00:00:00      8.44  2.38 Drama…
 7 tt1837492            2 13 Reasons… 2018-05-18 00:00:00      7.51  2.19 Drama…
 8 tt0285331            1 24          2002-02-16 00:00:00      8.56  6.67 Actio…
 9 tt0285331            2 24          2003-02-09 00:00:00      8.70  7.13 Actio…
10 tt0285331            3 24          2004-02-09 00:00:00      8.72  5.88 Actio…
# ℹ 2,256 more rows
```

Transforming and adding columns

Columns can be transformed or added by using functions on existing columns

1. Using `$` and the assign arrow `<-`

```r
dat_tv$year <- year(dat_tv$date)
```

2. Using piping and `mutate`

```r
dat_tv <- dat_tv %>%
  mutate(year = year(date))
```

Note that we still have to assign in order to store the new data frame

Selecting

Selecting takes a subset of *columns*

The show names are in the third column, called `title`

1. Using `$` and the name

```r
dat_tv$title
```

2. Using `[ ]` and an index

```r
dat_tv[ , 3] # First index left empty because we want all rows
```

3. Using piping and `select`

```r
dat_tv %>%
  select(title)
```

Filtering

Filtering takes a subset of *rows*

We pick out seasons with an average rating (`av_rating`) above 9

1. Using `[]` and a logical statement

```r
dat_tv[dat_tv$av_rating > 9,]
```

2. Using piping and `filter`

```r
dat_tv %>%
  filter(av_rating > 9)
```

Sorting

Sorting gives a re-ordering of the data

We order by average rating

1. Using `[]` and `order`

```r
dat_tv[order(dat_tv$av_rating, decreasing = T), ]
```

Note the use the argument `decreasing` to get the highest ratings first

2. Using piping and `arrange`

```r
dat_tv %>%
  arrange(-av_rating)
```

Note the use of `-` to get the highest rating first

Results

```r
dat_tv %>%  # Take the TV data, then ...
  filter(title == "Midsomer Murders") %>% # filter for title being Midsomer Murders, then ...
  arrange(-av_rating) %>% # sort by average rating in decreasing order, then ...
  select(title, seasonNumber, av_rating, genres) # select a subset of columns
```

```
# A tibble: 19 × 4
   title            seasonNumber av_rating genres
   <chr>                   <dbl>     <dbl> <chr>
 1 Midsomer Murders            2      7.92 Crime,Drama,Mystery
 2 Midsomer Murders            3      7.76 Crime,Drama,Mystery
 3 Midsomer Murders            9      7.74 Crime,Drama,Mystery
 4 Midsomer Murders            1      7.73 Crime,Drama,Mystery
 5 Midsomer Murders            5      7.66 Crime,Drama,Mystery
 6 Midsomer Murders            6      7.63 Crime,Drama,Mystery
 7 Midsomer Murders            7      7.63 Crime,Drama,Mystery
 8 Midsomer Murders           12      7.61 Crime,Drama,Mystery
 9 Midsomer Murders           16      7.59 Crime,Drama,Mystery
10 Midsomer Murders           17      7.54 Crime,Drama,Mystery
11 Midsomer Murders            4      7.53 Crime,Drama,Mystery
12 Midsomer Murders           10      7.51 Crime,Drama,Mystery
```

Aggregating

Aggregating calculates a summary value over a subset of values

We calculate mean rating across seasons

1. Using the `aggregate` function

```r
aggregate(av_rating ~ title, dat_tv, FUN = mean)
```

2. Using piping, `group_by` and `summarise`

```r
dat_tv %>%
  group_by(title) %>%
  summarise(mean_rating = mean(av_rating))
```

```
# A tibble: 868 × 2
   title                    mean_rating
   <chr>                          <dbl>
 1 11.22.63                        8.49
 2 12 Monkeys                      8.83
 3 13 Reasons Why                  7.97
 4 24                              8.58
 5 24: Legacy                      7.20
 6 24: Live Another Day            8.90
 7 39814                           8.27
 8 666 Park Avenue                 7.47
 9 7th Heaven                      7
10 8 Simple Rules                  8.17
# ℹ 858 more rows
```

Plots

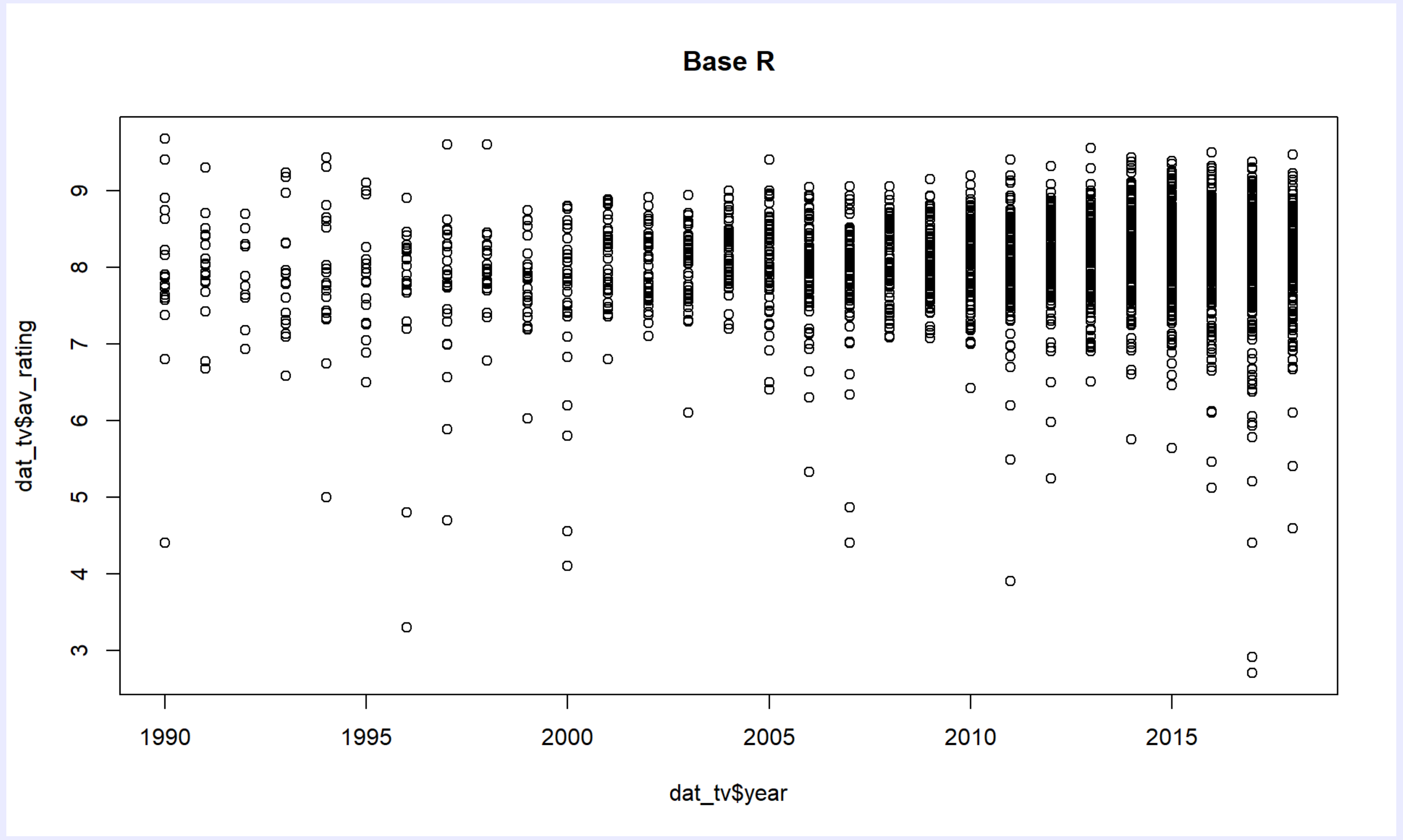
Base R includes functions to make plots

Highly customizable

Can add elements using functions like `points()`, `lines()` and `text()`

Average rating by year

```r
plot(dat_tv$year, dat_tv$av_rating, main = "Base R")
```



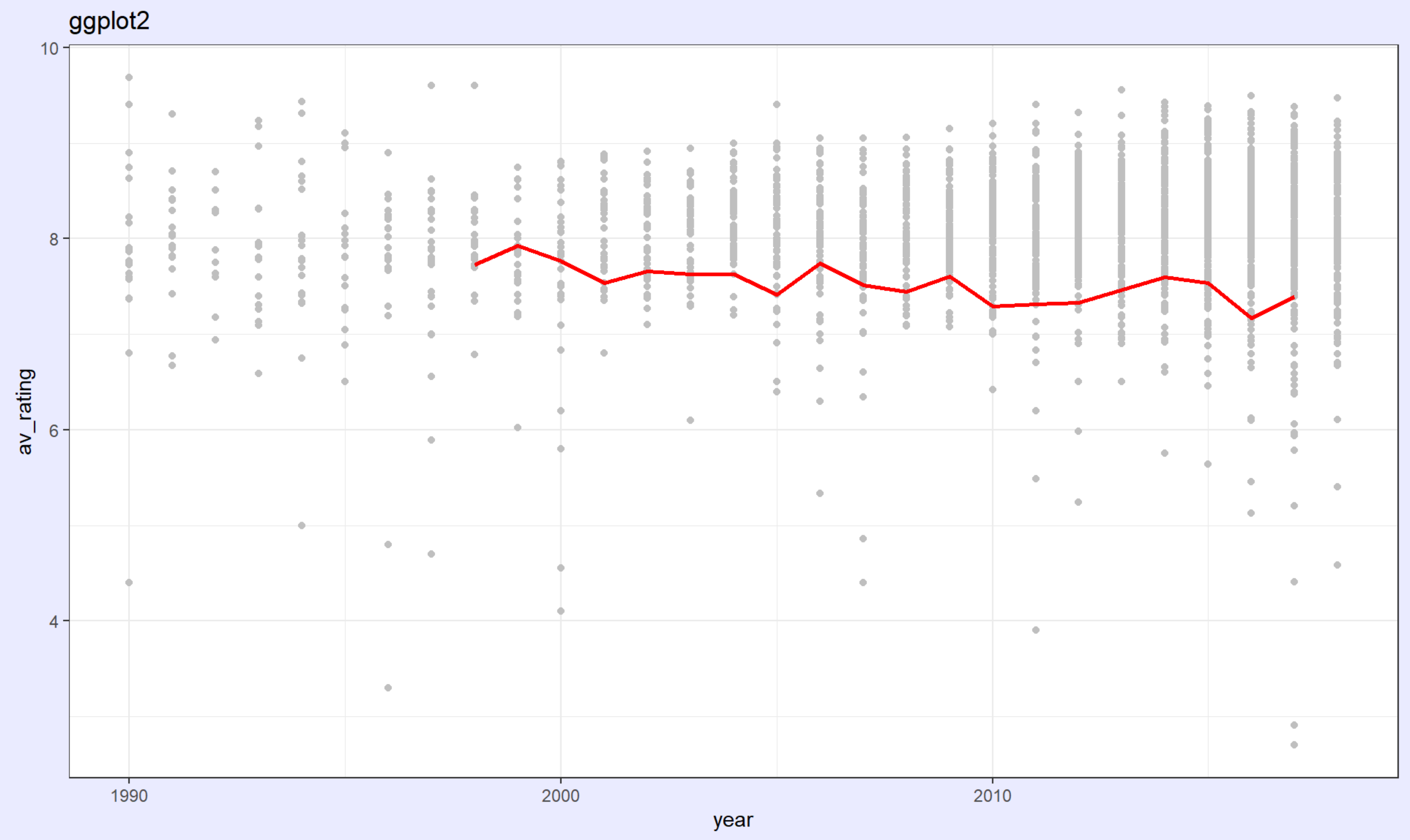
Plots. `ggplot2`

Many packages extend the plotting functionality, most notably `ggplot2`

Variables are specified in a special `aes()` function (aestethics)

Plot elements are added as `geom`s (geometries)

```r
library(ggplot2)
ggplot(dat_tv, aes(year, av_rating)) +
  geom_point(col = "grey75") +
  geom_line(data = dat_tv %>% filter(title == "Midsomer Murders"),
            col = "red", size = 1) +
  ggtitle("ggplot2")
```



RStudio

The RStudio window is divided into several frames

Code is run in the *Console* frame

Scripts

One usually writes code in a script (a separate text file)

File > *New file* > R *Script*

Lines from the script are run in the console by clicking the *Run* button or ctrl + enter

Scripts are saved with the file extention .R, but are just basic text files which can be opened in any text editor

Divide into sections and comment specific rows using #

RStudio. Projects

File > New project…

Creates a new folder for storing scripts, data, and output

Keeps the material contained, making it easier to keep track of changes and to share

Opening the project automatically sets the working directory to the project folder

Perfect to keep track of scripts written during a course

Something about style

R does not read empty space

Divide into several steps and add spaces and line breaks to make readable code

```r
x <- c(3, 14, 159)
y <- c(5, 2, 12)

plot(x, y, col = "red", pch = 3)
```

is more readable than

```r
plot(c(3,14,159),c(5,2,12),col="red",pch=3)
```

RMarkdown / Quarto

RMarkdown and Quarto allows one to mix written text and R code

The file can then be rendered into some standard format

File > *New file* > *R Markdown*… or *Quarto Document*…, then select file type

Possible output file types include html, pdf, and word

Troubleshooting

Every function has a help page

Either search in the *Help* frame in RStudio or run `?function_name`

```r
?plot
```

Always read error messages carefully

Use Google and Stack Overflow

Resources

R website: http://r-project.org

RStudio website: https://posit.co/

RStudio cheat sheets: https://posit.co/resources/cheatsheets/

Grolemund & Wickham, R for Data Science, https://r4ds.had.co.nz/

Wickham, Navarro & Pedersen, https://ggplot2-book.org/

Illustration: Amrei Binzer-Panchal

The End. Stick around for practical exercise