Statistical Computing with R

Lecture 5: loops in `R`; `for` loops; NA, NaN, Inf and NULL; data visualization (part 1)

Mirko Signorelli
: mirkosignorelli.github.io
: statcompr [at] gmail.com

Mathematical Institute
Leiden University

Master in Statistics and Data Science (2023-2024)

Universiteit Leiden

Announcements

- Assignment 1
  1. will be published tomorrow
  2. 2 exercises that cover course material up to lecture 6 (= next week)
  3. deadline for submissions: October 15
- Struggling with `R` or the assignment?
  1. Next question hour: October 10
  2. please sign up at least 24 hours before it
  3. more info on dedicated Brighspace page

Recap

Lecture 4:

- logical operators
- `if` statements
- organizing your files
- R Markdown
- differences between `R` scripts and `R` Markdown *(self-study)*

Today:

- more on file naming and organizing your files *(self-study)*
- loops in `R`
- `for` loops
- special values (`NA`, `NaN`, `Inf` and `NULL`)
- data visualization: part 1
- charts to visualize univariate frequency distributions

More on file naming and organizing your files (SELF-STUDY)

Loops in `R`

`for` loops

Special values: NA, NaN, Inf and NULL

Data visualization (part 1)

Frequencies: categorical variables

Frequencies: quantitative variables

More on file naming and organizing your files

- In lecture 2 we talked about good and bad naming conventions for your files

Saving a script

- R scripts can be saved using File > Save / Save as, or using the *floppy disk* icons
- Try to give your scripts a (short) name that will make sense when you (/ someone else) come back to it!

Dos and don'ts:

- script1.R, script2.R ✗
- model fitting.R ✗
- model_fitting.R ✓
- 1_data_import.R, 2_data_cleaning.R, 3_model_estimation.R ✓

More on file naming and organizing your files

- In lecture 2 we talked about good and bad naming conventions for your files
- In lecture 4 we discussed the importance of organizing your files in a well-structured way

Folders and subfolders (cont'd)

Within each project folder, you may:

1. save your R scripts and R Markdown files directly in the project folder
2. create subfolders where different input and output files are saved (see next slide!)

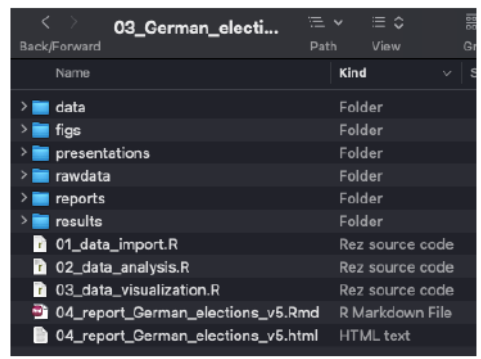


Figure 2: example of the use of subfolders to organize files within a project

More on file naming and organizing your files

- In lecture 2 we talked about good and bad naming conventions for your files
- In lecture 4 we discussed the importance of organizing your files in a well-structured way

Folders and subfolders (cont'd)

Within each project folder, you may:

1. save your R scripts and R Markdown files directly in the project folder
2. create subfolders where different input and output files are saved (see next slide!)

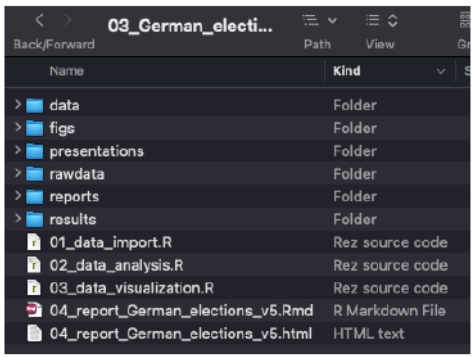


Figure 2: example of the use of subfolders to organize files within a project

- Here's a useful presentation where you can read more on these two topics (self-study)

More on file naming and organizing your files (SELF-STUDY)

Loops in R

`for` loops

Special values: NA, NaN, Inf and NULL

Data visualization (part 1)

Frequencies: categorical variables

Frequencies: quantitative variables

"Looping"

- Programming often involves the repetition of identical / similar operations
- These repetitive tasks are often referred to as looping / iterating / cycling
- Three types of loops: `for ( )`, `while ( )` and `repeat`
  1. today: `for` loops
  2. lecture 7: `while` and `repeat`

More on file naming and organizing your files (SELF-STUDY)

Loops in `R`

`for` loops

Special values: NA, NaN, Inf and NULL

Data visualization (part 1)

Frequencies: categorical variables

Frequencies: quantitative variables

The `for` loop

- `for` loops can be used to specify that a list of instructions should be executed a fixed number of times
- Syntax of a `for` loop:

```
for (index in vector) {
  # instruction #1
  # instruction #2
  # ...
}
```

- Major PRO: `for` loops are often intuitive ☺ → we basically can tell `R` to do what we would do if we were doing computations manually
- Major CON: `for` loops are ⚠ often not the most efficient way to code a repetitive task ⚠. Better alternatives are often possible (we will see some later in this course)
- However, sometimes the `for` loop may be the only / the most logical way to implement an algorithm

Example: computing a finite sum

- How can we compute $\sum_{x=5}^{70} \sqrt{x}$?
- If we were to do it by hand:
  1. Take 5 $\rightarrow \sqrt{5} = 2.236$
  2. Take 6 $\rightarrow \sqrt{6} = 2.449$
  3. Compute $\sqrt{5} + \sqrt{6} = 2.236 + 2.449 = 4.685$
  4. And so on, until we reach 70
- The same, but in R:

```
total = 0
for (x in 5:70) total = total + sqrt(x)
total
```

```
## [1] 388.2755
```

Do we really need a `for` loop?

PRO TIP:

- Every time you write a `for` loop, ask yourself 2 questions:
  1. do I actually need to write a loop, or can I do without?
  2. is a `for` loop the best I can do, or can I use better alternatives? (Examples: `apply`, `sapply`, `replicate`, `foreach`; all covered in later classes)
- Trade-off simplicity-efficiency
- Back to our example: it's pretty easy to compute $\sum_{x=5}^{70} \sqrt{x}$ avoiding the `for` loop:

```
sum(sqrt(5:70))
```

```
## [1] 388.2755
```

- Less code & faster = more efficient!

Your turn

Exercises

The Fibonacci sequence is a sequence of numbers with initial values 0 and 1, where each number is equal to the sum of the previous two. Formally:

$$x_k = x_{k-1} + x_{k-2} \text{ for } k > 2,$$

where $x_1 = 0$ and $x_2 = 1$.

1. Write a `for` loop to compute the first 20 elements in the sequence
2. Write a function that computes the sum of the first n elements in the sequence. Use it to compute the sum of the first 30 elements in the sequence

Solutions

```r
# Ex 1
x = c(0, 1, rep(NA, 18))
for (i in 3:20) x[i] = x[i-1] + x[i-2]
x
```

```
##  [1]    0    1    1    2    3    5    8   13   21   34
## [11]   55   89  144  233  377  610  987 1597 2584 4181
```

```r
# Ex 2
fibo = function(n) {
  x = c(0, 1, rep(NA, n-2))
  for (i in 3:n) x[i] = x[i-1] + x[i-2]
  sum(x)
}
fibo(30)
```

```
## [1] 1346268
```

NA: missing data in R

- When dealing with real data, information may sometimes be missing
- In `R`, missing data are coded as `NA` (= not available)
- `is.na` can be used to check which elements of a vector are missing

```
v = c(17, 42, NA, 28)
is.na(v)
```

```
## [1] FALSE FALSE  TRUE FALSE
```

NaN = not a number

- You know it: you just can't divide 0 by 0
- Formally, 0 / 0 is "undefined"
- In R, this is coded as NaN (= Not a Number!)

```
b = 0 / 0
b
```

```
## [1] NaN
```

Figure 1: An NaN protester at a demonstration in Hyde Park, London, in May 2020

Inf

- Infinite values are stored as `Inf`

```
v = 0:3
2 / v
```

```
## [1]       Inf 2.0000000 1.0000000 0.6666667
```

NULL

- NULL represents a value that does not exist. It can be thought as a vector of length 0:

```
c(4, NULL, 4)
```

```
## [1] 4 4
```

- Initializing a vector equal to null creates an empty vector:

```
v = c(56, Inf, 23)
v = NULL
v
```

```
## NULL
```

```
v[1:3] = 4:6
v
```

```
## [1] 4 5 6
```

NULL (cont'd)

- Setting a variable to NULL in a data frame removes the variable:

```
df1 = data.frame(name=c("Mark", "Margaret", "Wang", "Pedro"),
      age=c(25, 45, 32, 19),
      country=c('Germany', 'Australia', 'China', ' Mexico'),
      job=c('waiter', 'chef', 'plumber', 'student'))
df1$country = NULL
df1
```

```
##       name age     job
## 1     Mark  25  waiter
## 2 Margaret  45    chef
## 3     Wang  32 plumber
## 4    Pedro  19 student
```

More on file naming and organizing your files (SELF-STUDY)

Loops in `R`

`for` loops

Special values: NA, NaN, Inf and NULL

Data visualization (part 1)

Frequencies: categorical variables

Frequencies: quantitative variables

Data visualization

Let's face two harsh facts:

1. statistics ain't a popular subject ☹
2. the man in the street does not like numbers, let alone tables ☹☹

Data visualization

Let's face two harsh facts:

1. statistics ain't a popular subject ☹
2. the man in the street does not like numbers, let alone tables ☹☹

So... How can we communicate simple(-ish) statistical concepts to people who would pay little attention to our numbers and tables?

Data visualization

Let's face two harsh facts:

1. statistics ain't a popular subject ☹
2. the man in the street does not like numbers, let alone tables ☹☹

So... How can we communicate simple(-ish) statistical concepts to people who would pay little attention to our numbers and tables?

- Data visualization ("dataviz") is a powerful way to communicate results to a non-technical audience
- `R` offers a wide range of functions and packages for dataviz
- Broad subject - we start today with the basics; more will follow in lecture 6 and in block 2!

More on file naming and organizing your files (SELF-STUDY)

Loops in `R`

`for` loops

Special values: NA, NaN, Inf and NULL

Data visualization (part 1)

Frequencies: categorical variables

Frequencies: quantitative variables

Categorical data

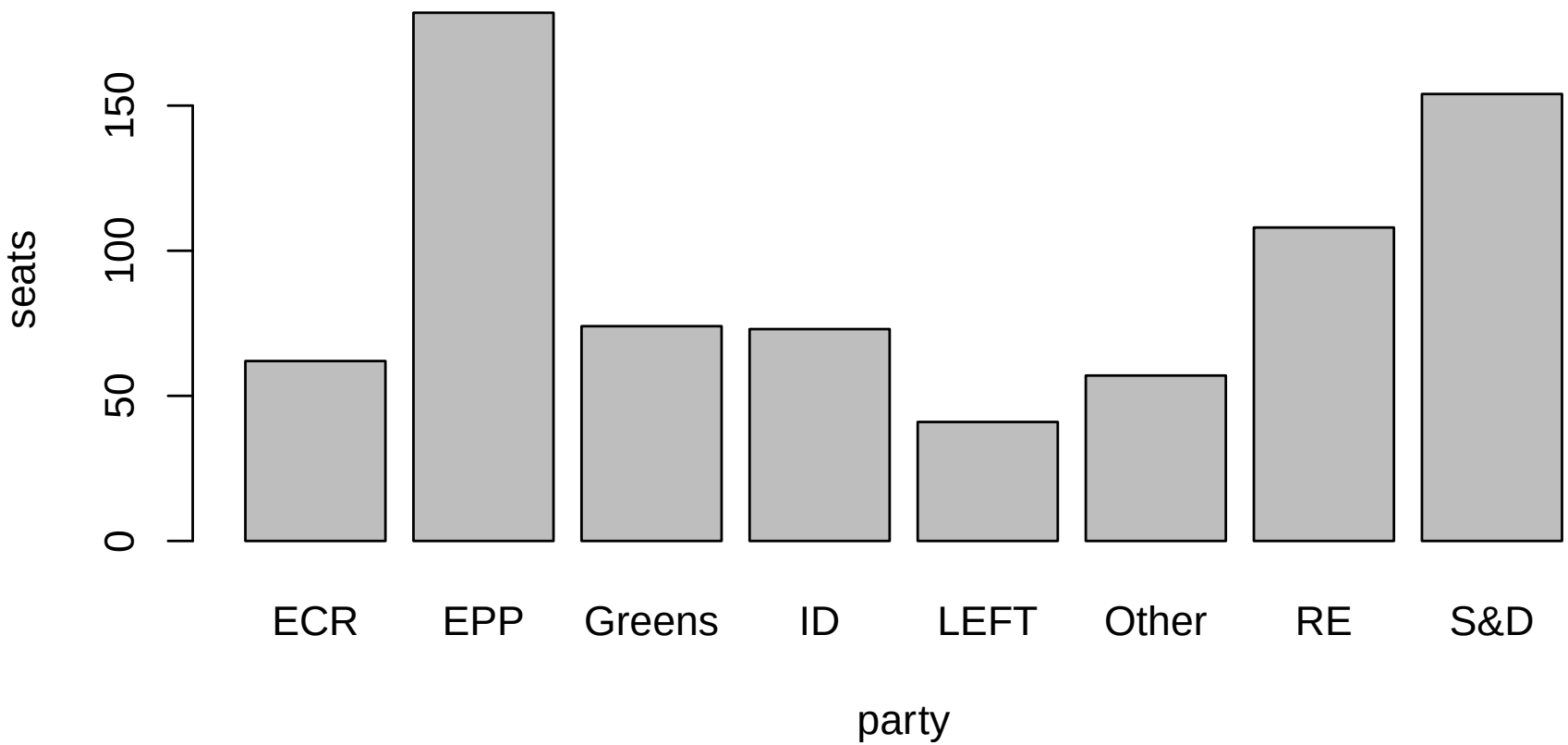
- Let's consider the distribution of seats in the European Parliament:

```
ep2019 = data.frame(
  party = c('EPP', 'S&D', 'RE', 'Greens',
            'ID', 'ECR', 'LEFT', 'Other'),
  seats = c(182, 154, 108, 74, 73, 62, 41, 57))
```

- How can we visualize the distribution of seats by party?
- Some options:
  1. Bar plot
  2. Pie chart
  3. Waffle chart

Bar plot

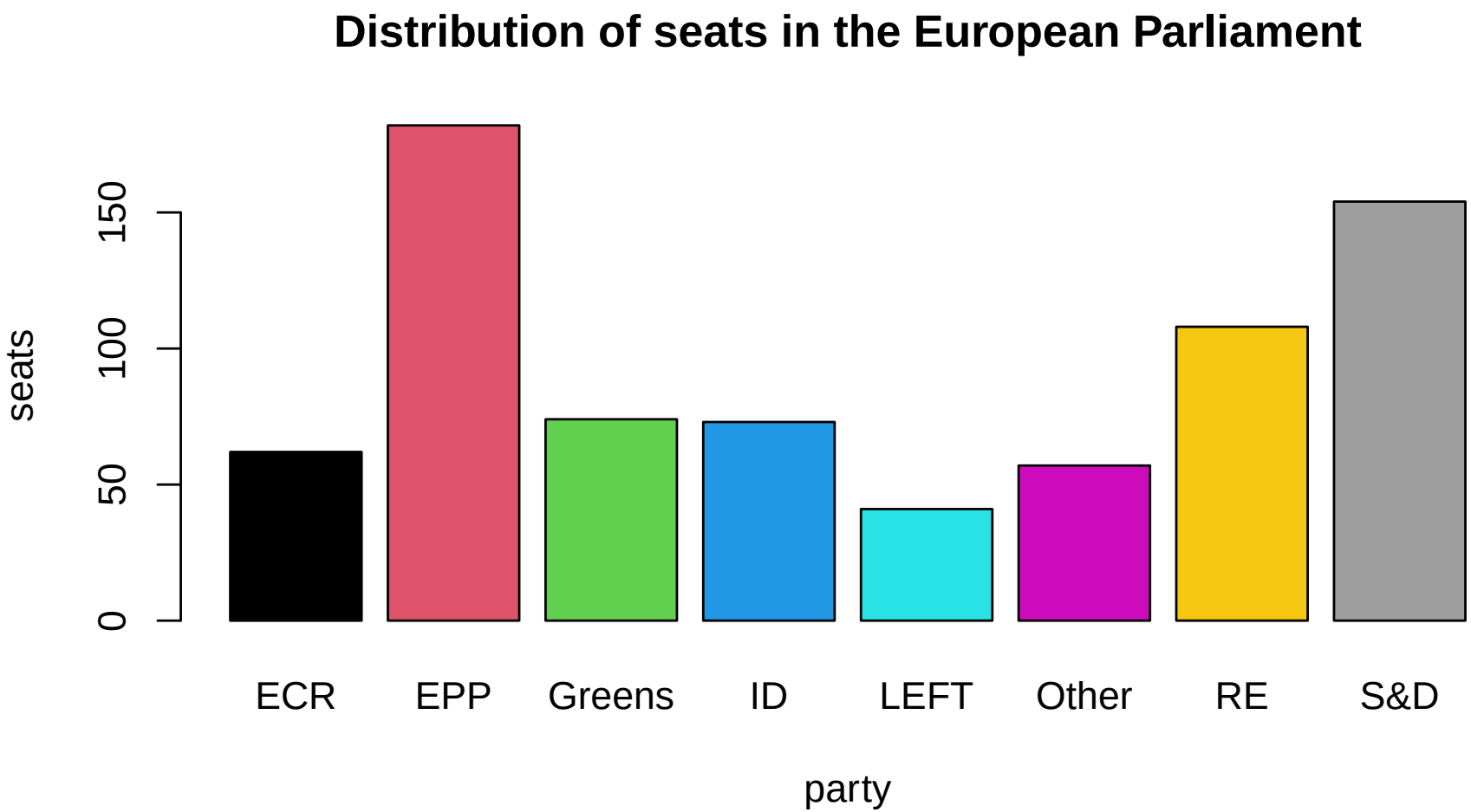
```
barplot(seats ~ party, data = ep2019)
```



- This is a bit too basic
- The next slides show how to improve this chart step by step

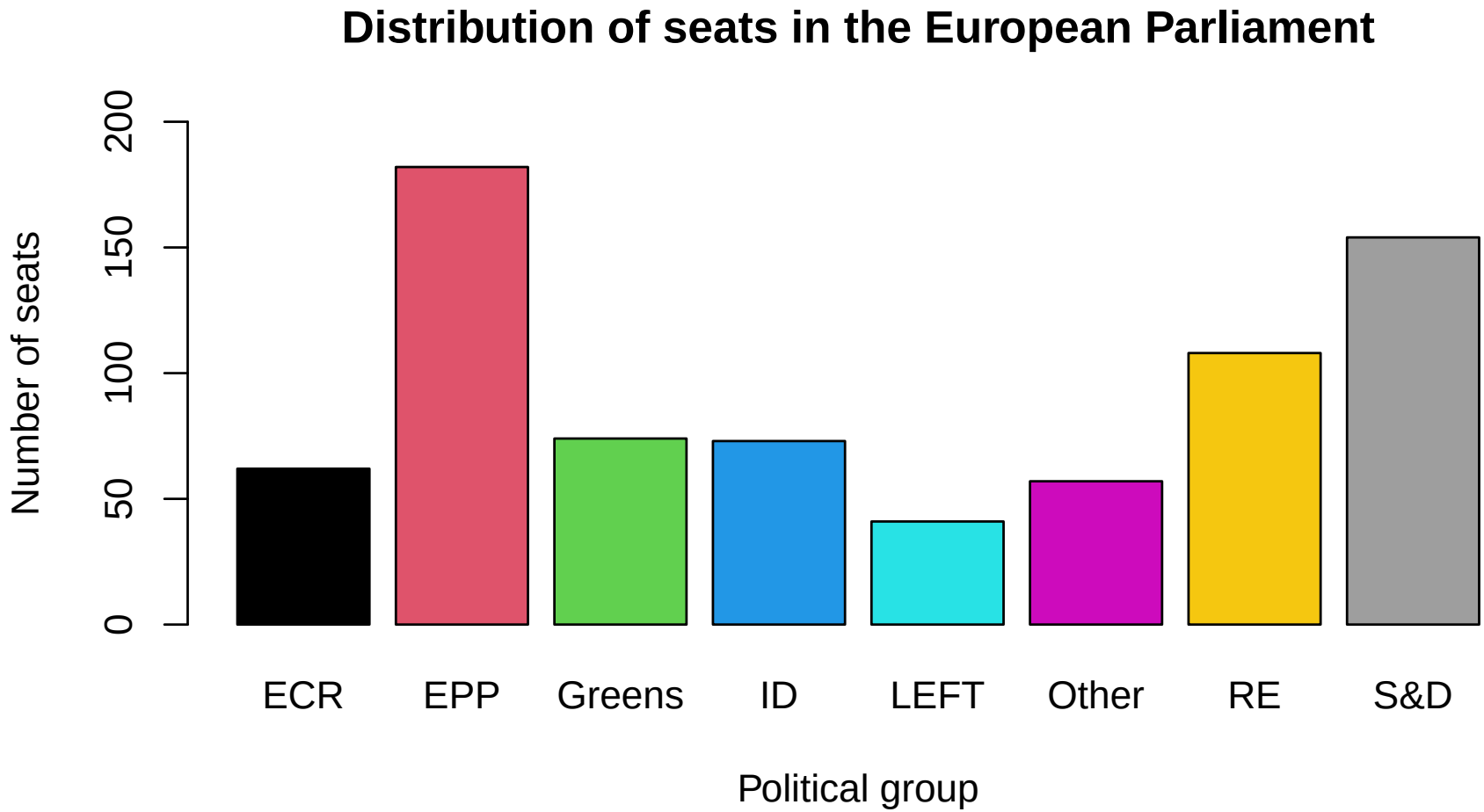
How to add colours and title

```
title = 'Distribution of seats in the European Parliament'
barplot(seats ~ party, data = ep2019,
        col = 1:8, main = title)
```



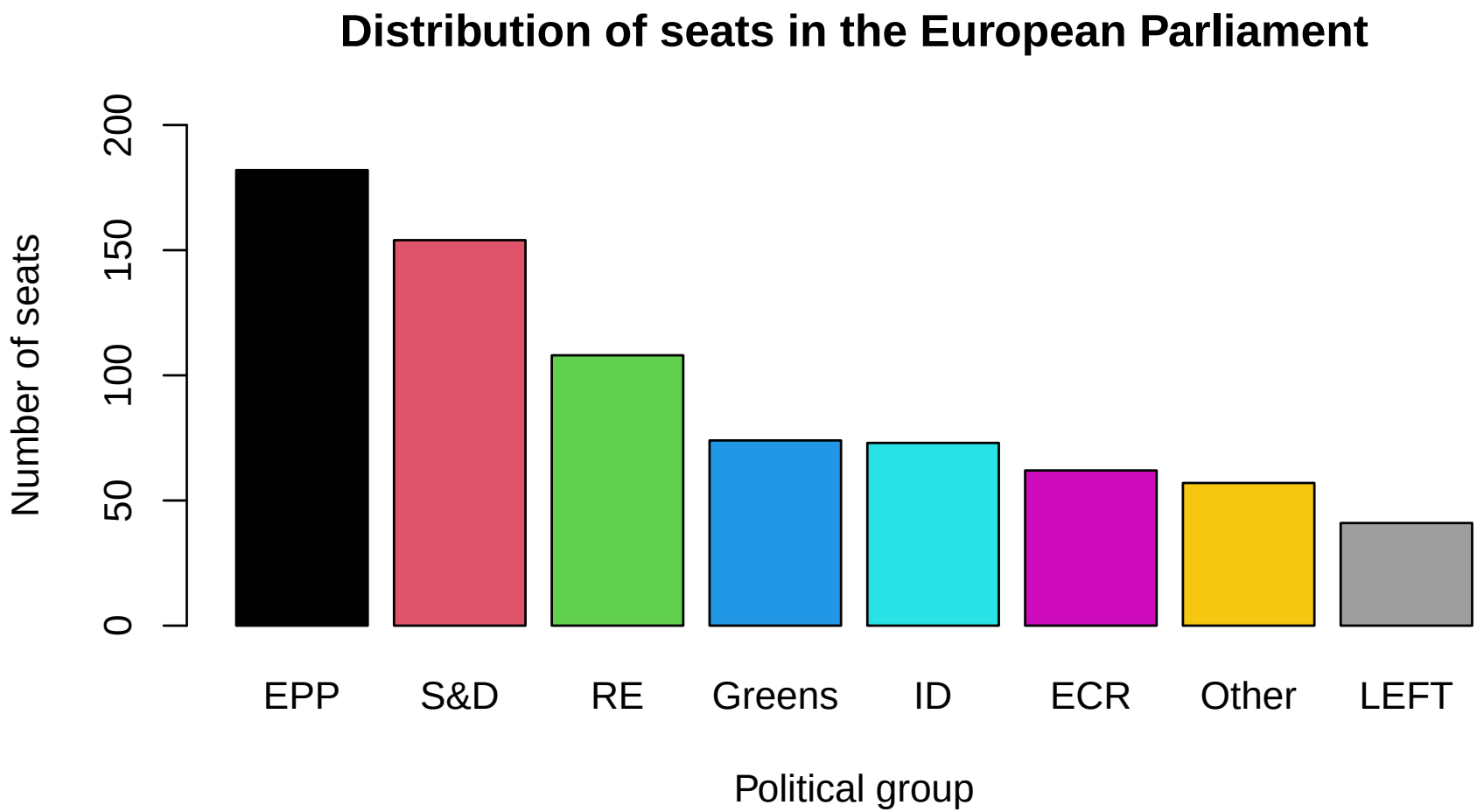
How to fix axes labels and values

```
barplot(seats ~  party, data = ep2019,
        col = 1:8, ylim = c(0, 200), main = title,
        xlab = 'Political group', ylab = 'Number of seats')
```



Ordering bars by value

```
ep_sorted = ep2019[order(ep2019$seats, decreasing = T), ]
barplot(ep_sorted$seats, names.arg = ep_sorted$party,
        col = 1:8, ylim = c(0, 200), main = title,
        xlab = 'Political group', ylab = 'Number of seats')
```



More about colours

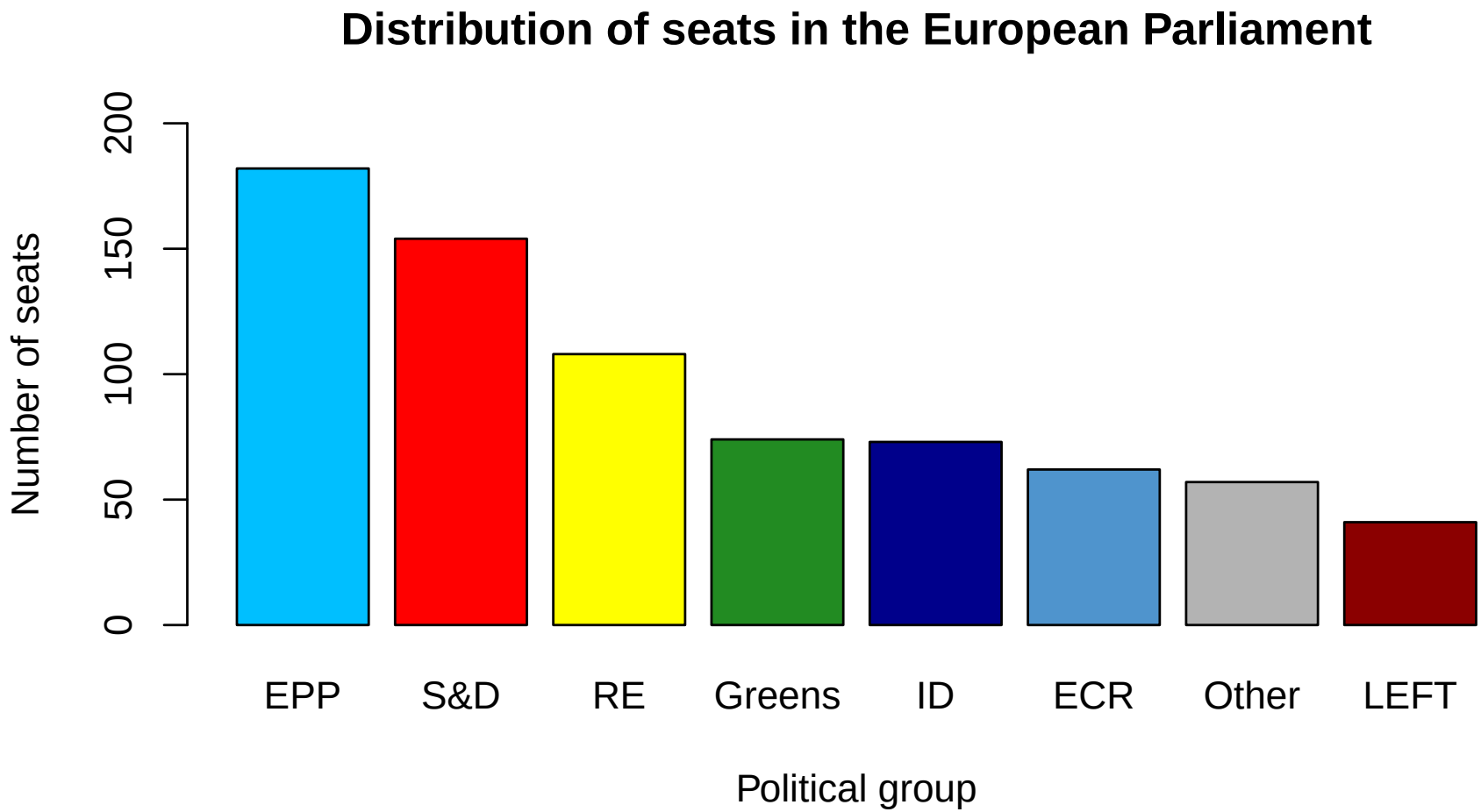
- `col = 1:8` is a quick and dirty way to add colours to a chart
- 657 colours in R → type `colours()` in the console
- A handy overview: list of colours in R
- Let's match colours to political group:

```
ep_sorted$color = c('deepskyblue', 'red', 'yellow',
                    'forestgreen', 'darkblue',
                    'steelblue3',  'gray70', 'red4')
head(ep_sorted, 5)
```

```
##    party seats       color
## 1    EPP   182 deepskyblue
## 2    S&D   154         red
## 3     RE   108      yellow
## 4 Greens    74 forestgreen
## 5     ID    73    darkblue
```

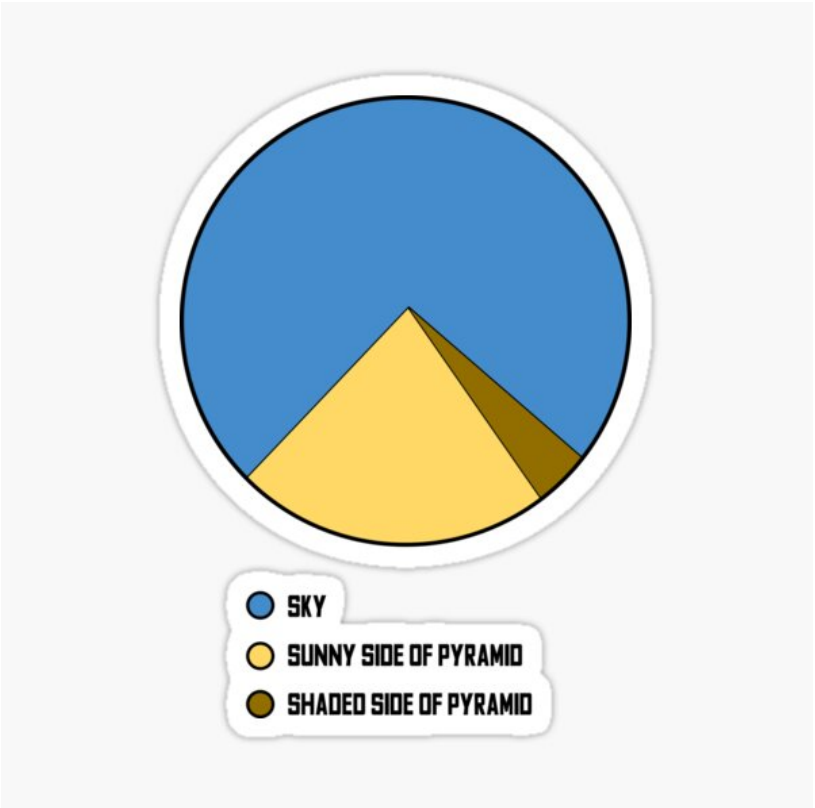
The end result

```
barplot(ep_sorted$seats, names.arg = ep_sorted$party,
        col = ep_sorted$color, ylim = c(0, 200), main = title,
        xlab = 'Political group', ylab = 'Number of seats')
```



Pie charts

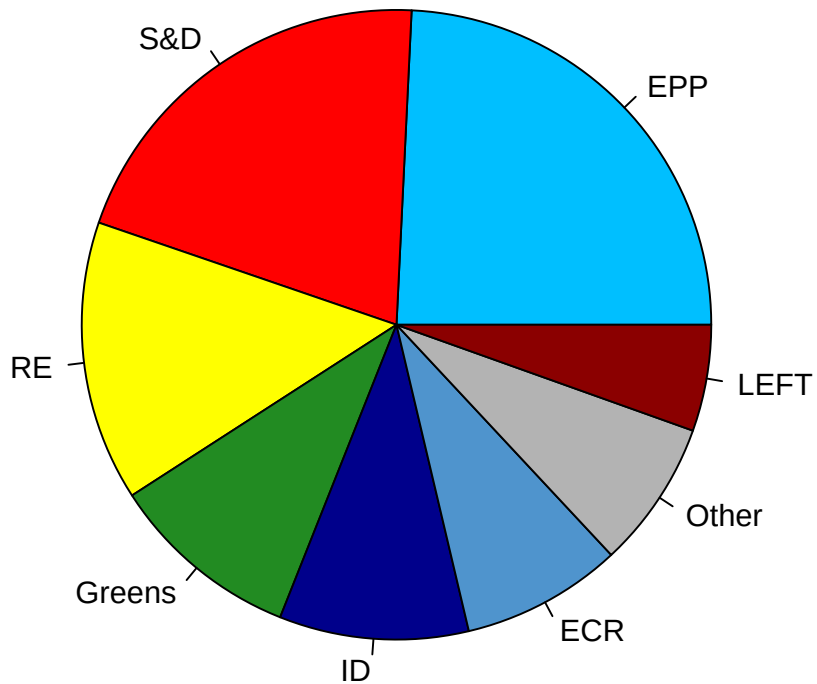
- A few pros:
  1. rather intuitive
  2. most people are familiar with pie charts (and with slices of pies!)
  3. easy-ish to read with few (3-4) groups
- A few cons:
  1. low sensitivity of human eye to small differences in angles (can you really distinguish 8% from 11% in a pie chart?)
  2. hard to read with many groups



⚠ The use of pie charts is controversial: many argue against its use ⚠

Pie chart

```
pie(ep_sorted$seats, labels = ep_sorted$party,
    col = ep_sorted$color, main = title)
```



Waffle plot (R package waffle)

- To create a waffle plot, let's first install the R package waffle(and dependencies)

```
install.packages("waffle")
```

```
library(waffle)
```

- As you can see from ?waffle, ⚠ this function takes as input a named vector, not a data frame! ⚠

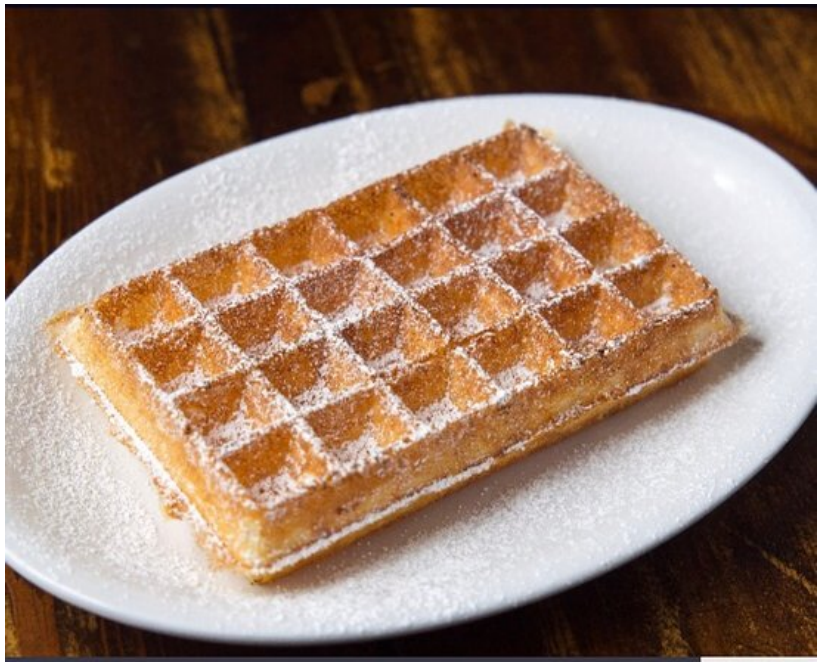
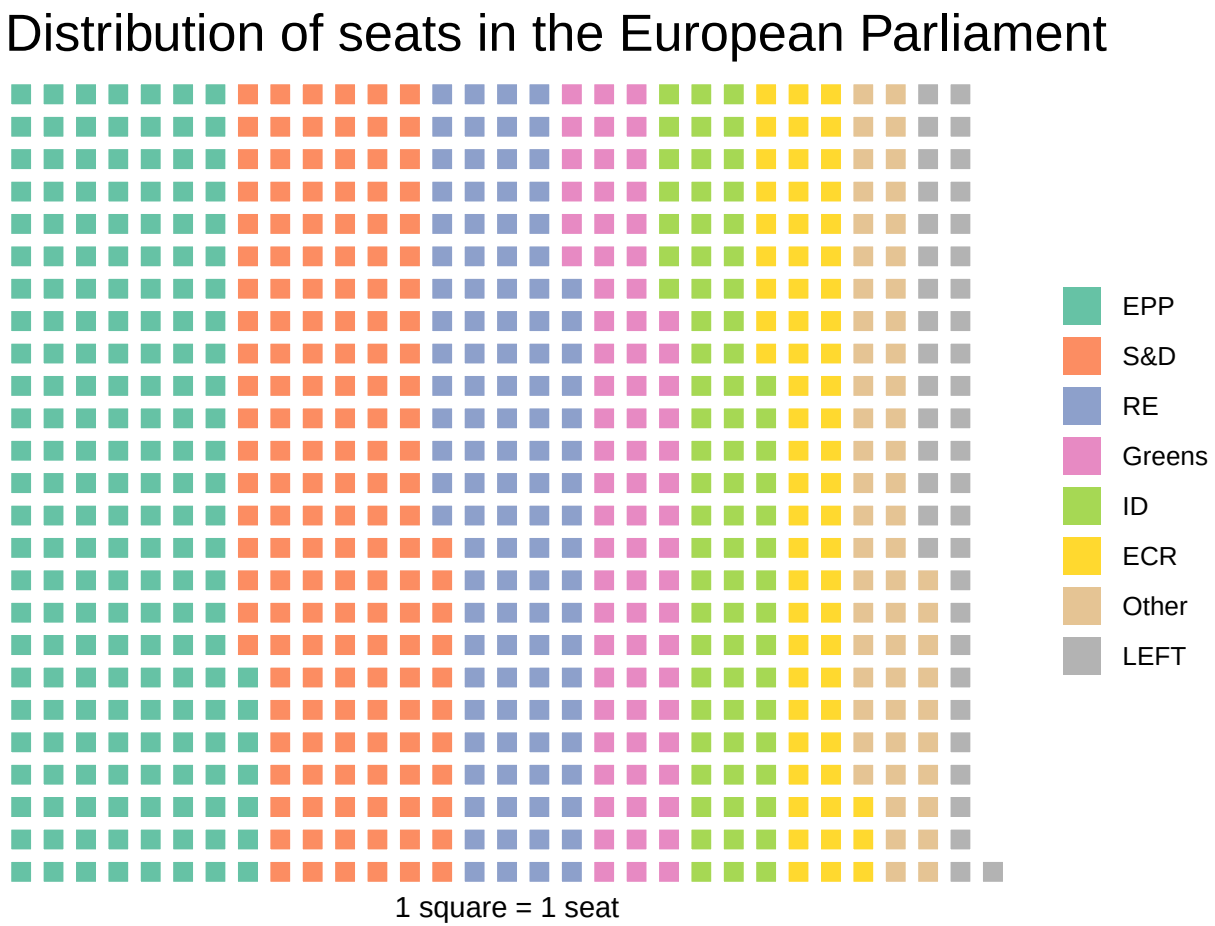


Figure 2: A Bruxelles waffle

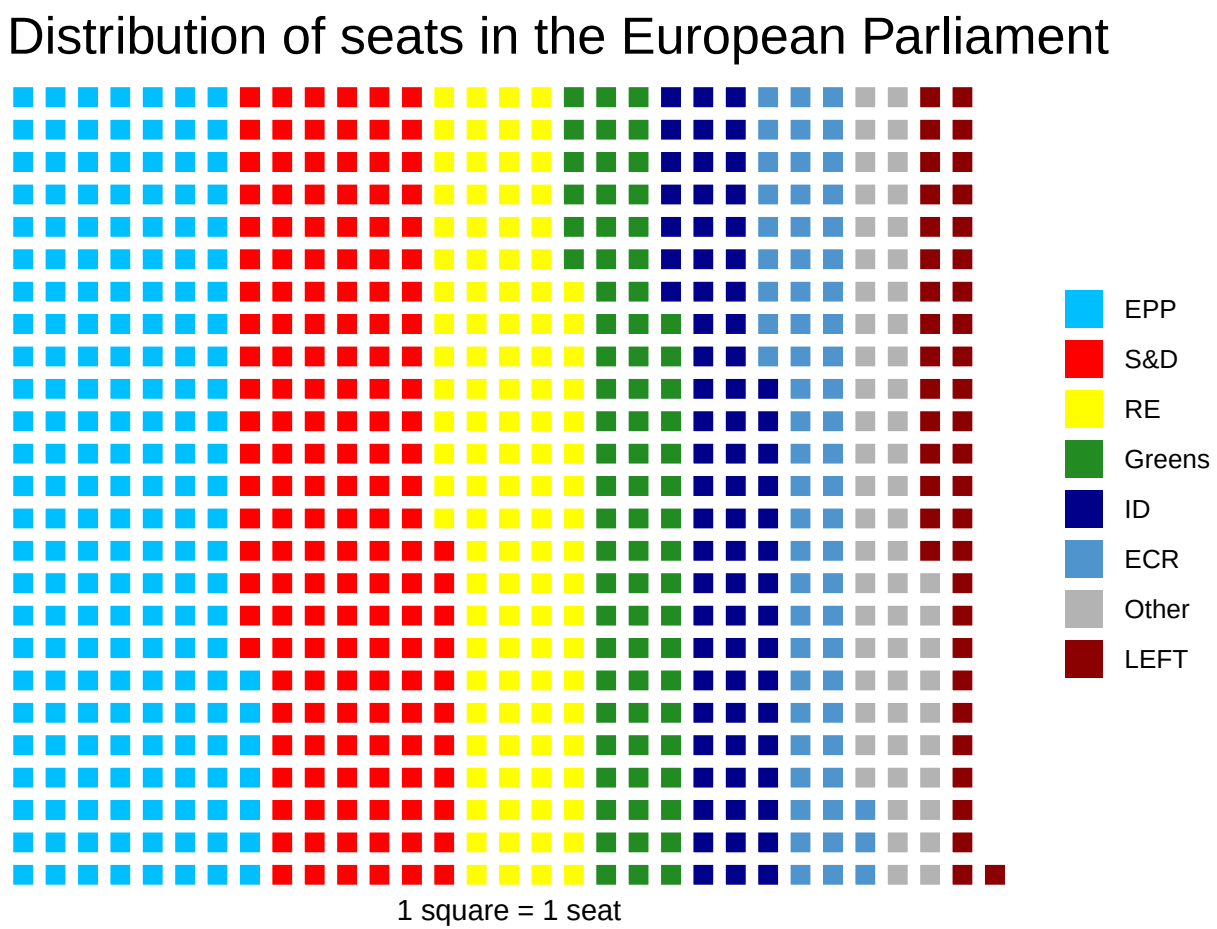
Waffle plot (default colours)

```
x = ep_sorted$seats
names(x) = ep_sorted$party
waffle(x, rows = 25, xlab = '1 square = 1 seat',
       title = title)
```



Waffle plot (custom colours)

```
waffle(x, rows = 25, colors = ep_sorted$color,
       xlab = '1 square = 1 seat', title = title)
```



Your turn

Exercises

In which continent were you born?

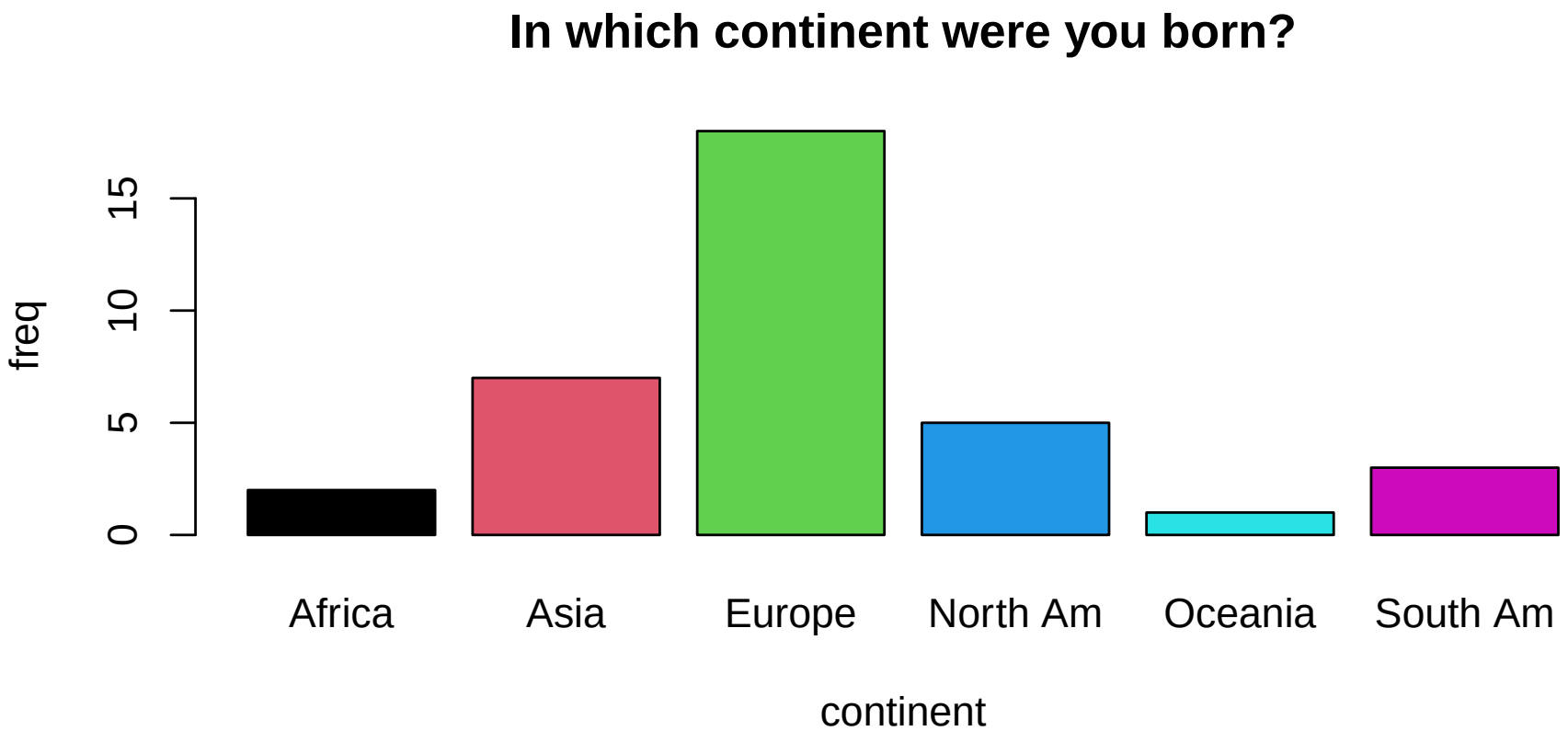
1. Let's gather data on birth by continent in the classroom!
2. Store the gathered data in a data frame
3. Visualize the data with a bar plot, a pie chart and a waffle plot

Tally:

Continent	Frequency
Europe	
Asia	
North America	
South America	
Africa	
Oceania	

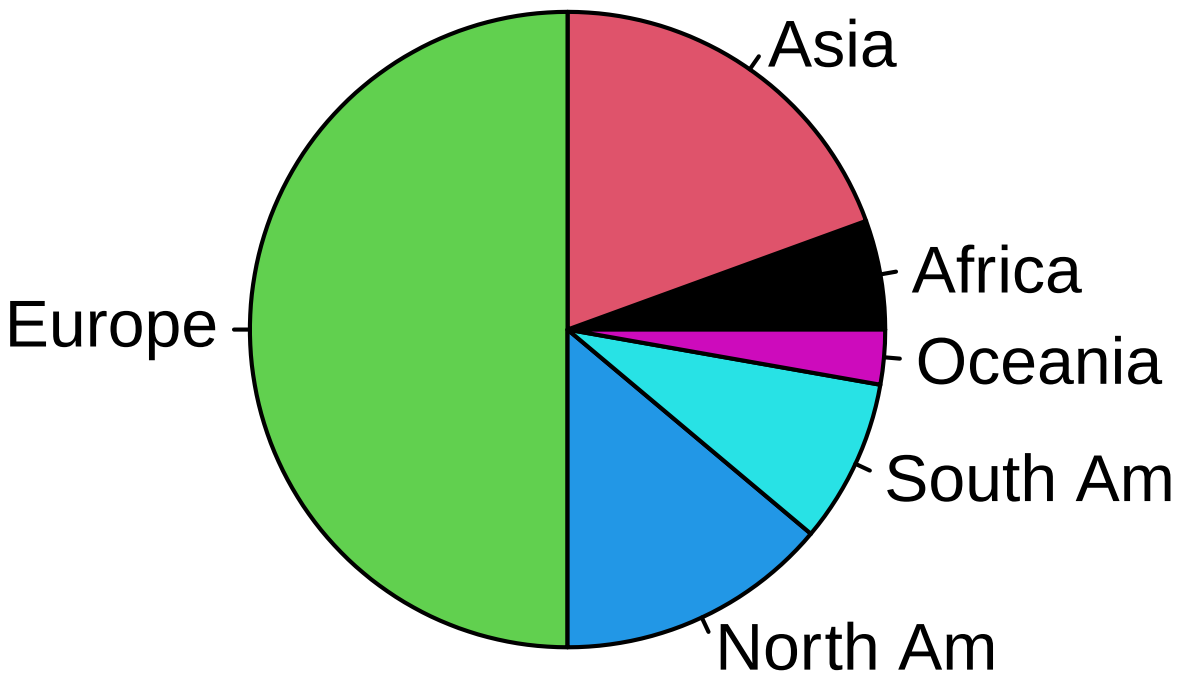
Solution

```
df = data.frame(continent = c('Africa', 'Asia',
    'Europe', 'North Am', 'South Am', 'Oceania'),
    freq = c(2, 7, 18, 5, 3, 1), xlim = c(0, 20))
barplot(freq ~  continent, data = df, col = 1:6,
        main = 'In which continent were you born?')
```



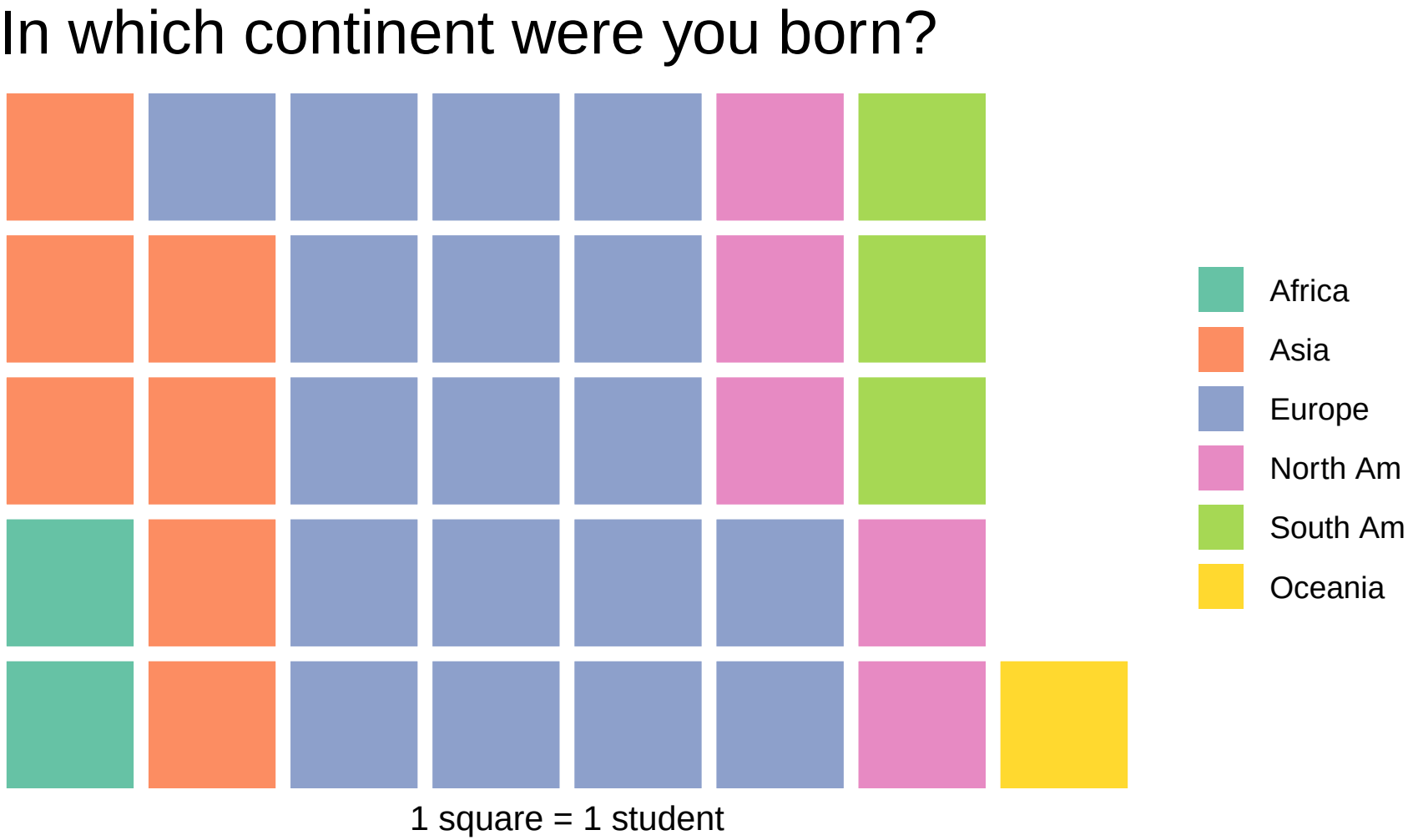
Solution (cont'd)

```
pie(df$freq, labels = df$continent, col = 1:6,
    main = 'In which continent were you born?', cex.main = 1)
```



Solution (cont'd)

```
freq = df$freq
names(freq) = df$continent
waffle(freq, rows = 5, xlab = '1 square = 1 student',
       title = 'In which continent were you born?')
```



More on file naming and organizing your files (SELF-STUDY)

Loops in `R`

`for` loops

Special values: NA, NaN, Inf and NULL

Data visualization (part 1)

Frequencies: categorical variables

Frequencies: quantitative variables

Quantitative data

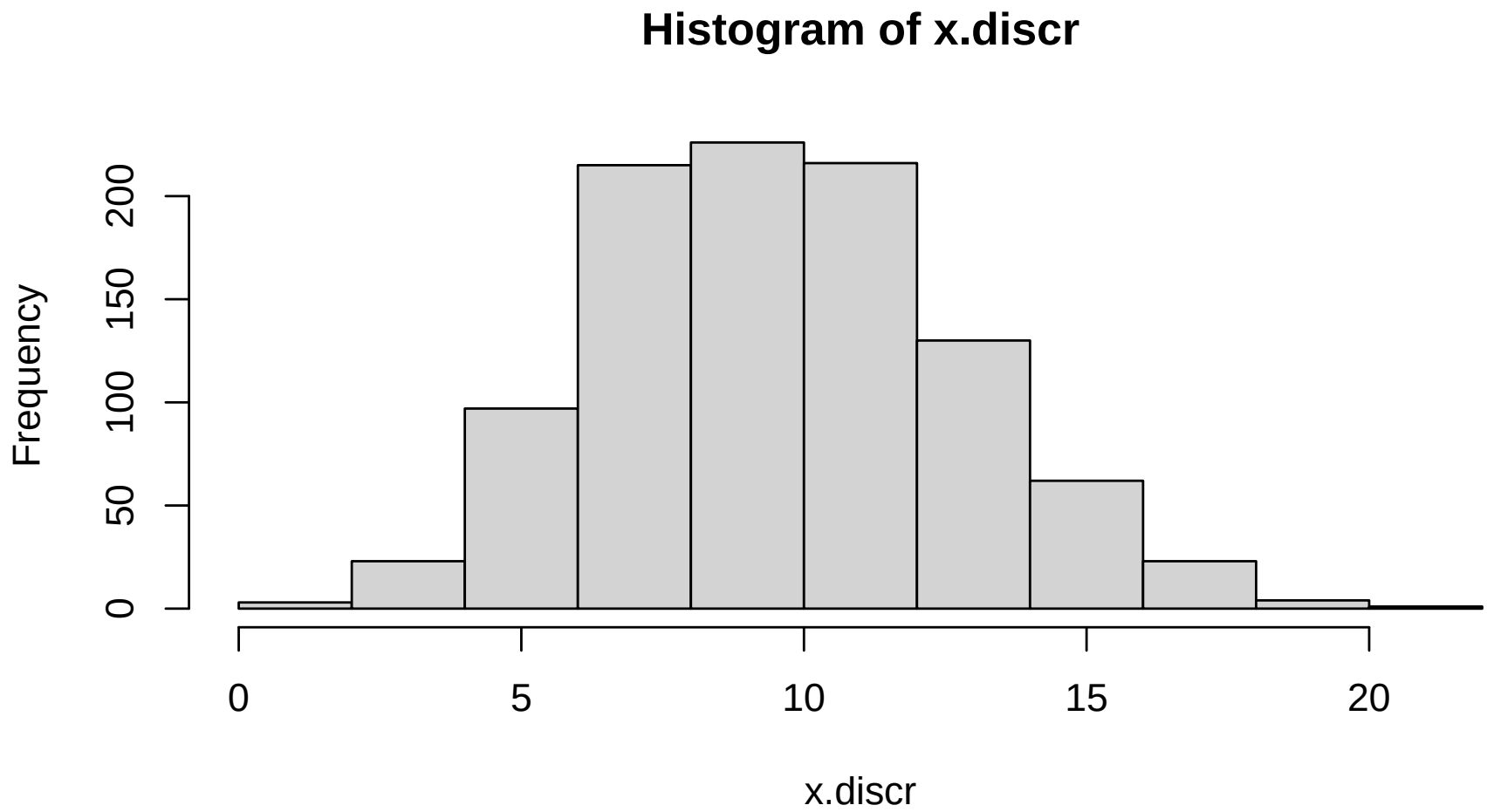
- Histograms are the most widely used way to visualize quantitative variables
- However, they may not always be the best option

Two useful alternatives:

- plots displaying the empirical pmf of X if X is discrete
- density plots if X is continuous

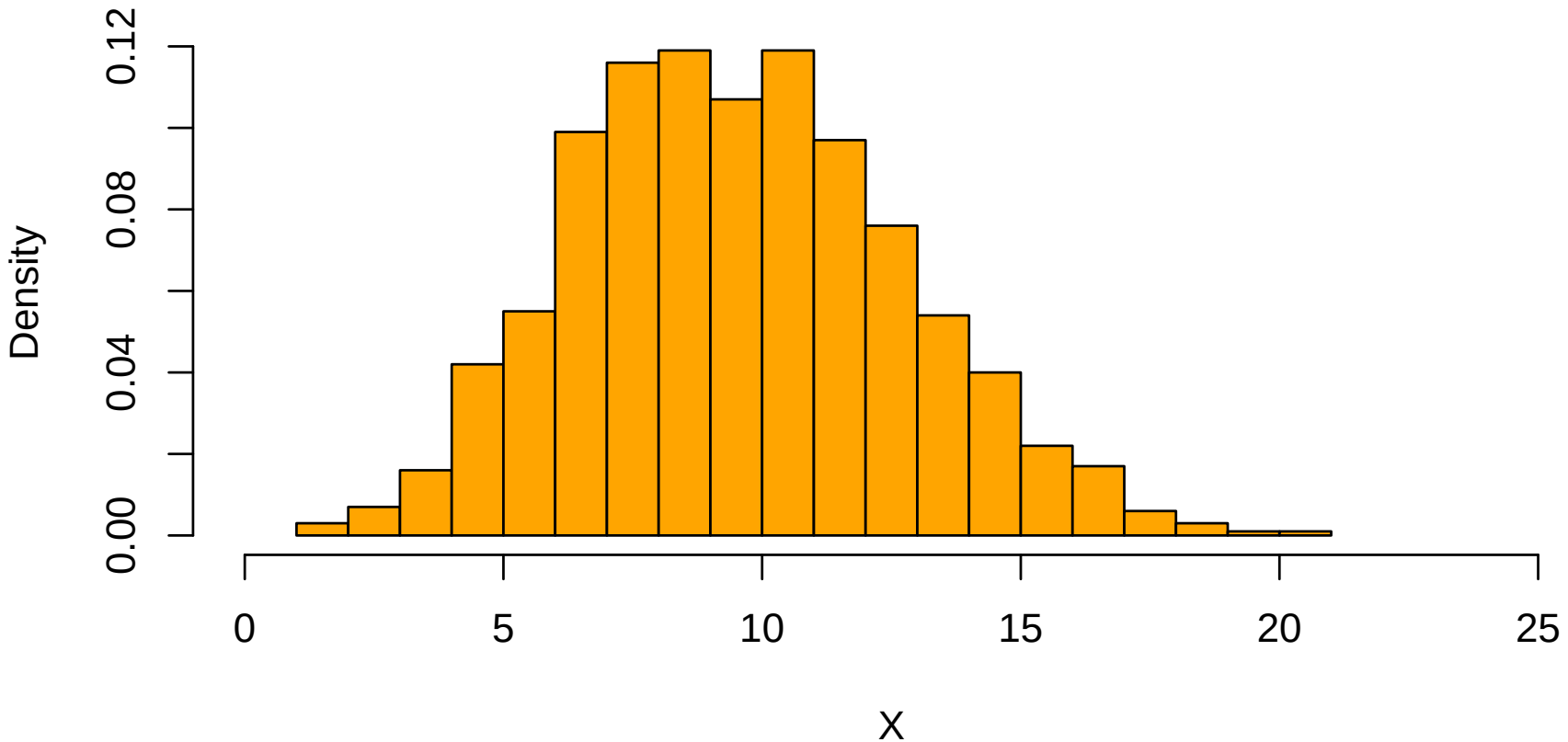
Discrete variables: histogram

```r
set.seed(5)
x.discr = rpois(1000, lambda = 10)
hist(x.discr)
```



Discrete variables: histogram (cont'd)

```
hist(x.discr, 20, freq = F, col = 'orange',
     xlim = c(0, 25), xlab = 'X', main = '')
```



Discrete variables: empirical pmf plot

- To create the plot in the next slide, you first need to install the package ptmixed
- ptmixed is a CRAN package that requires other packages as dependencies; ⚠ one of these dependencies, tweeDEseq, is not from CRAN, but from Bioconductor! ⚠
- Installation option 1 (easier): pretend that ptmixed itself is on Bioconductor

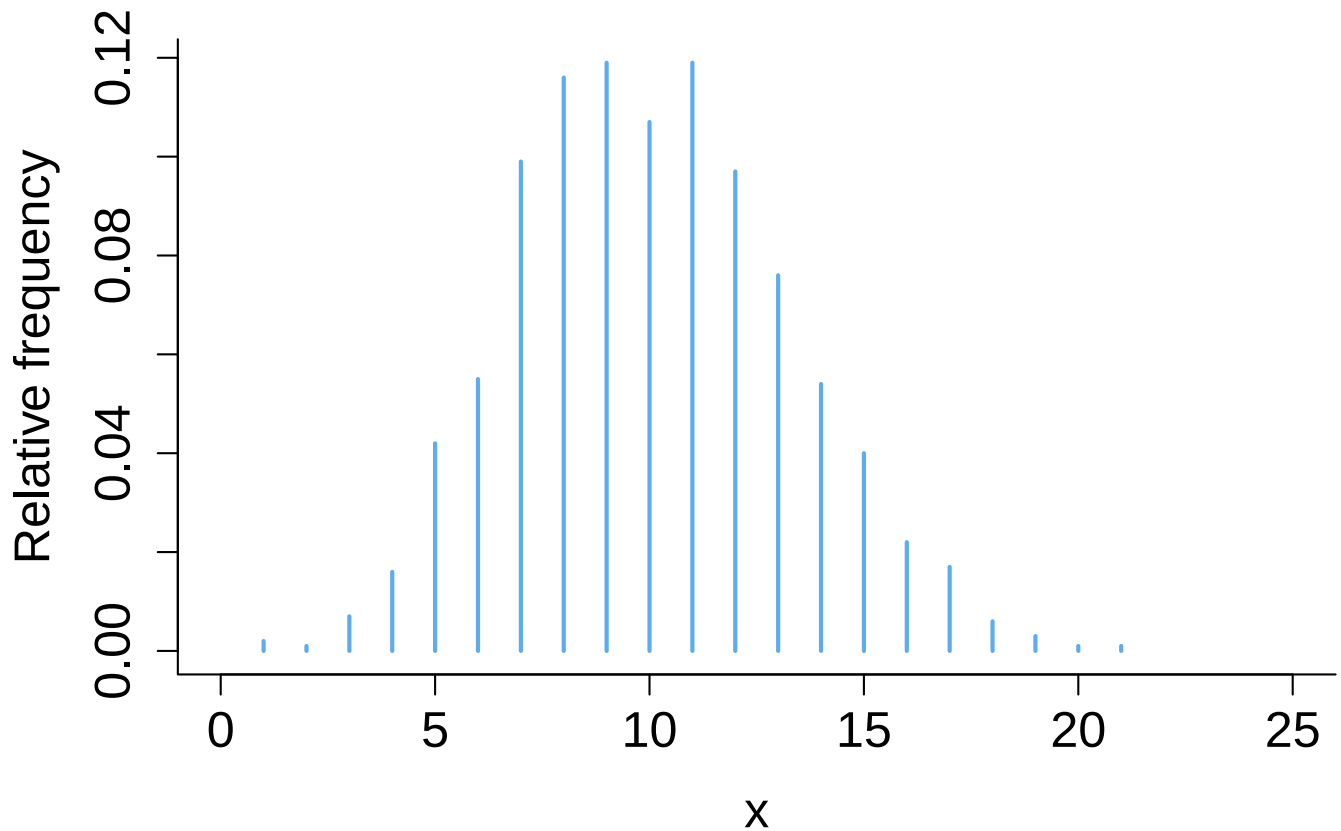
```
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install('ptmixed')
```

- Installation option 2 (orthodox, but more cumbersome):

```
# step 1: install tweeDEseq from Bioconductor
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install('tweeDEseq')
# step 2: install ptmixed and CRAN dependencies
install.packages('ptmixed')
```

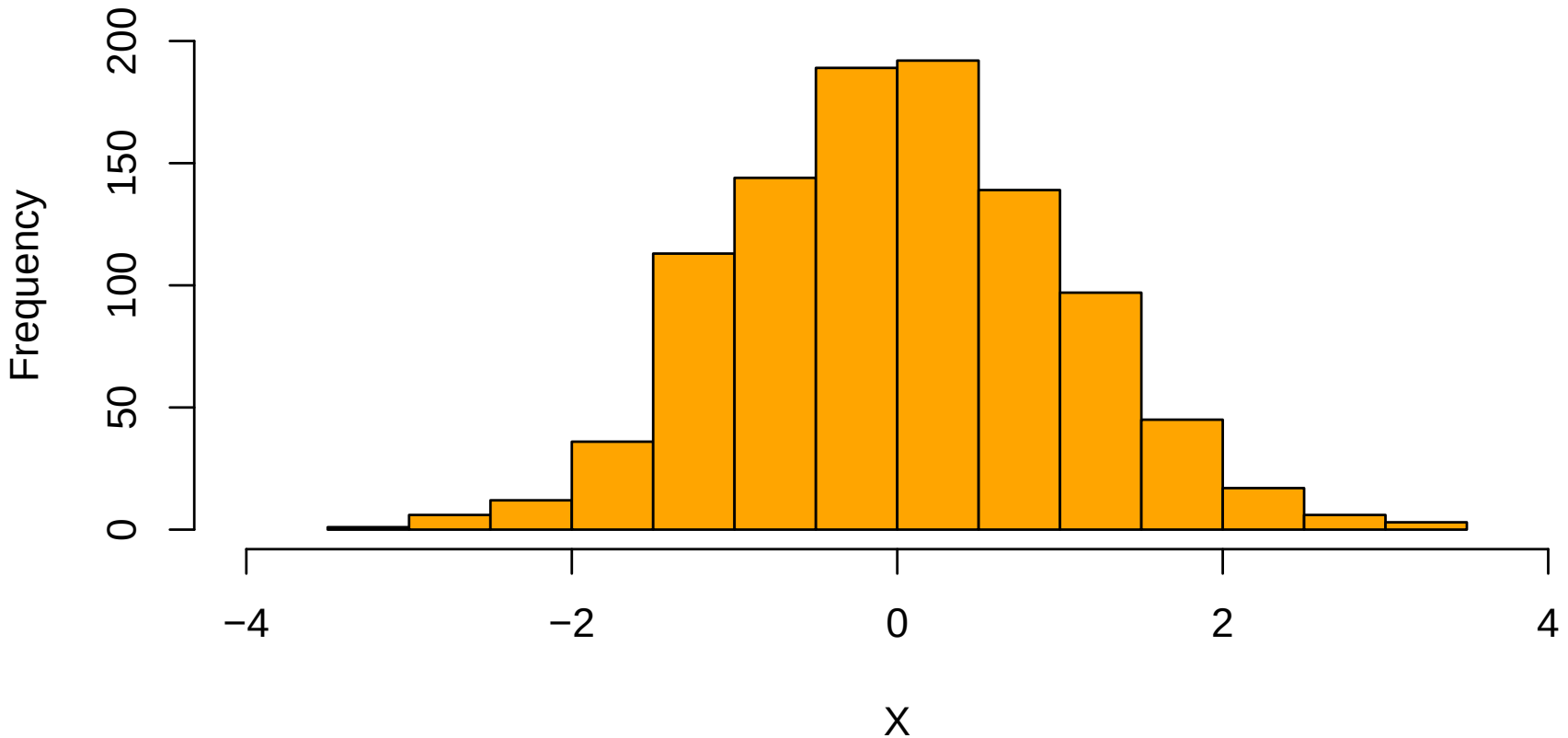
Discrete variables: empirical pmf plot

```
library(ptmixed)
pmf(x.discr, absolute = F, cex.axis = 1.5,
    xlim = c(0, 25), lwd = 2.5, col = 'steelblue2')
```



Continuous variables: histogram

```
x.cont = rnorm(1000)
hist(x.cont, 20, col = 'orange',
     xlim = c(-4, 4), ylim = c(0, 200),
     xlab = 'X', main = '')
```



Continuous variables: density plot

```r
par(bty = 'l')
plot(density(x.cont), 'density estimator of X')
polygon(density(x.cont), col = "steelblue2")
```