Data Visualization

Carolina Fellinghauer[*]
Bernd Fellinghauer[†]

Contents

[*]carolina.fellinghauer@uzh.ch
[†]febernd@gmail.com


numerical
purpose
visual
metric
analyzing
effective
data
science
organizing
interpreting
presenting
colors
collecting
information
categorical
statistics
decisions

Getting started

Background

Data visualization allows to explore data graphically for patterns and detailed behaviors. Typically complex data contents can be summarized in tables and visualized with plots. Visual perception is an important part of the data visualization as graphical displays have to be decoded by the viewer unambiguously. The course will not go into questions of design, color theories, or the choice and reasons regarding what to display and how. This course is intended for a one day tutorial and demo about the functions and techniques to write R-code for visual analysis in research. It is not comprehensive, but still the quantity of functions, packages and options to be used is extensive enough to create many useful plots.

Regarding good visualization guidelines:

- Cleveland, W.S. (1994). *The Elements of Graphing Data*. New Jersey, Hobart Press.
- Cleveland, W.S. (1993). *Visualizing Data*. New Jersey, Hobart Press.
- Few, S. (2009). *Now you see it: Simple Visualization Techniques for Quantitative Analysis*. Oakland, CA: Analytics Press.
- Few, S. (2012). *Show Me the Numbers: Designing Tables and Graphs to Enlighten, 2nd. Ed.*. Burlingame, CA: Analytics Press.
- Tufte, E.R. (2001). *The Visual Display of Quantitative Information, 2nd Ed.*. Cheshire, Connecticut: Graphics Press.

Regarding data visualization with R:

- Murrel, P. (2006). *R Graphics*. Boca Raton, FL: Chapman & Hall, Taylor and Francis Group.
- Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. New York, NY: Springer-Verlag.
- Rahlf, T. (2017). *Datenvisualisierung mit R: 111 Beispiele* (2. erweiterte Auflage). Bonn, Deutschland: SpringerSpektrum

Setting paths

First, the paths have to be set to where the course data and outputs will be saved. The necessary R-libraries are loaded in the sections where they are used.

```
# Setting data path & a directory for the output.

path_general = "C:/path_to_your_course_folder/"
path_data=paste(path_general, "Data/", sep="") # Path to data
dir.create(paste(path_general, "Output/", sep=""), showWarnings=FALSE) # Put output folder
path_output=paste(path_general, "Output/", sep="") # Path to output folder


# Read in the WHO-MDS data sample
Smpl_who=read.csv(file=paste(path_data, "WHO_MDS_large.csv", sep=""),
                  header=TRUE, sep=",")

dim(Smpl_who)

[1] 500  15

str(Smpl_who)

'data.frame':   500 obs. of  15 variables:
 $ X           : num  63 4474 6241 8484 3717 ...
 $ ID          : int  1 2 3 4 5 6 7 8 9 10 ...
 $ sex         : Factor w/ 2 levels "female","male": 2 1 1 2 1 2 1 2 1 1 ...
 $ age         : int  58 70 40 60 57 54 51 37 30 26 ...
 $ stand_up    : int  1 1 1 1 2 1 3 1 1 1 ...
 $ getting_out : int  1 1 1 1 3 1 2 2 1 1 ...
 $ go_somewhere: int  1 1 1 1 4 1 2 1 1 1 ...
 $ dexterity   : int  1 1 1 1 1 1 1 1 1 1 ...
 $ energy      : int  1 1 1 1 1 1 3 2 3 1 ...
 $ relating    : int  1 1 1 1 1 1 2 1 1 2 ...
 $ stress      : int  NA 1 1 1 4 1 4 1 2 3 ...
 $ memory      : int  1 2 1 1 4 1 2 1 1 1 ...
 $ relax       : int  1 1 1 1 1 1 3 1 1 1 ...
 $ anxiety     : int  2 2 2 2 1 1 2 2 2 1 ...
 $ health      : num  9.82 44.03 0 9.82 45.35 ...
```

The data

A random sample of N=500 adults from the Model Disability Survey (from WHO) in Chile is used to show some useful functions and commands for visual analysis in R. The variables have been selected having in mind practical and realistic data-situations and not with the idea to illustrate a research question.

The following variables have been retained:

- **ID** – an ID for each person = row number
- **sex** – 1 for male, 2 for female
- **stand up** – standing up from sitting
- **getting out** – getting out of the house
- **go somewhere** – getting where you want to go
- **dexterity** – manipulating small objects, e.g. open jars
- **energy** – feeling tired and not having enough energy
- **relating** – relating with persons that you do not know
- **memory** – remembering important things
- **relax** – relaxing and enjoying life
- **stress** – managing stress
- **anxiety** – having anxiety: 1=yes, 2=no
- **health** – Rasch based health (problems) score 0-100, 0=no problem

The variables **stand up** to **stress** are coded:

- 1 = no problem
- 2 = mild problem
- 3 = moderate problem
- 4 = severe problem
- 5 = extreme problem

Coloring

Coloring in R is a broad topic. Available colors can be called by simply entering *colors()* in the console. R has 657 ready build in colors. Beyond this, thematic palettes can be called in specific R-packages, so that the less-inspired user can find choices of color schemes that match well together. Otherwise, if the right color is still not found, it is possible to program one's own color.

The following link displays the standard colors in R with their labels:

http://www.stat.columbia.edu/ tzheng/files/Rcolor.pdf

```
# List of first 100 colors in R.
colors()[1:100]
```

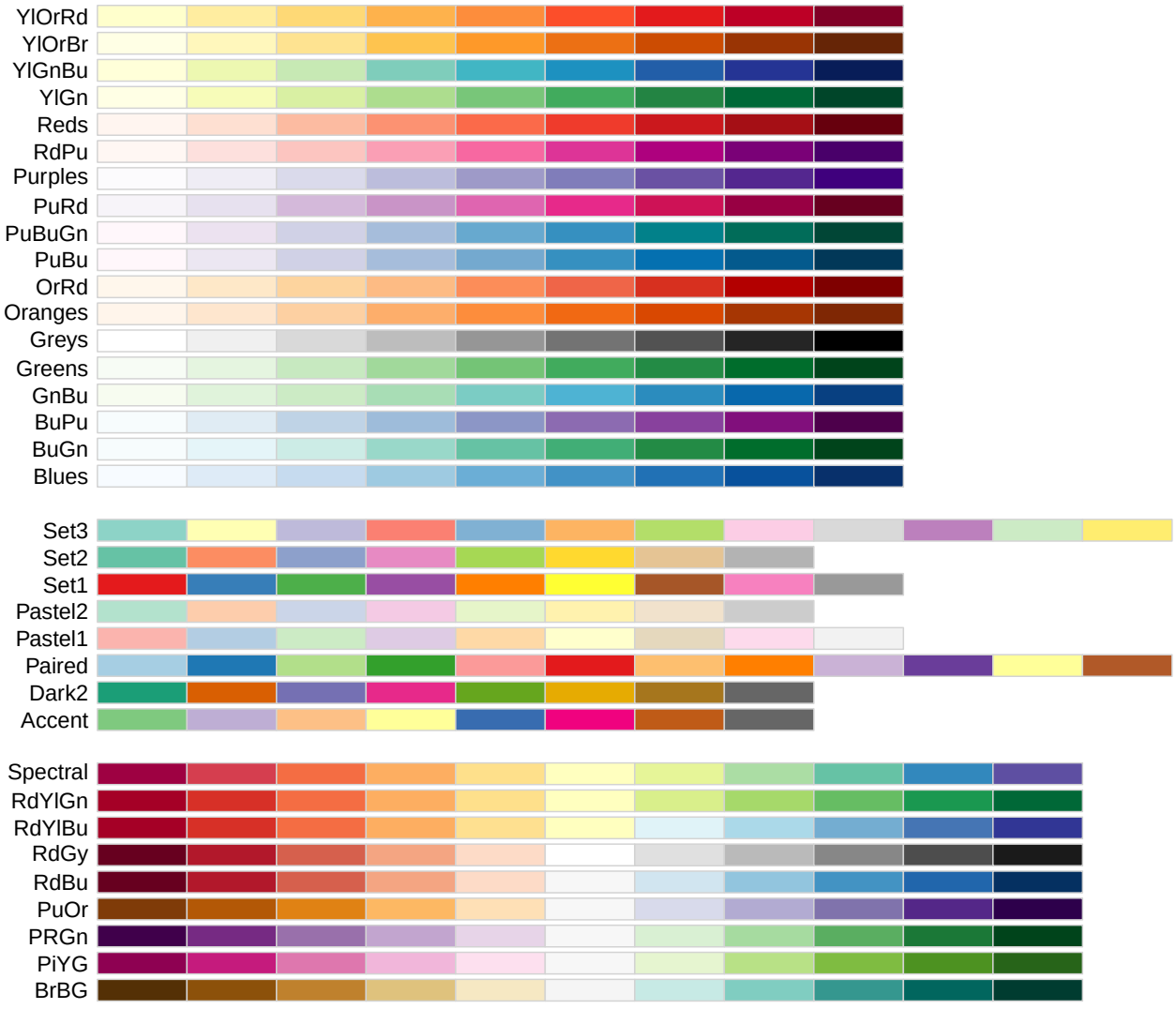
```
FALSE   [1] "white"           "aliceblue"       "antiquewhite"
FALSE   [4] "antiquewhite1"   "antiquewhite2"   "antiquewhite3"
FALSE   [7] "antiquewhite4"   "aquamarine"      "aquamarine1"
FALSE  [10] "aquamarine2"     "aquamarine3"     "aquamarine4"
FALSE  [13] "azure"           "azure1"          "azure2"
FALSE  [16] "azure3"          "azure4"          "beige"
FALSE  [19] "bisque"          "bisque1"         "bisque2"
FALSE  [22] "bisque3"         "bisque4"         "black"
FALSE  [25] "blanchedalmond"  "blue"            "blue1"
FALSE  [28] "blue2"           "blue3"           "blue4"
FALSE  [31] "blueviolet"      "brown"           "brown1"
FALSE  [34] "brown2"          "brown3"          "brown4"
FALSE  [37] "burlywood"       "burlywood1"      "burlywood2"
FALSE  [40] "burlywood3"      "burlywood4"      "cadetblue"
FALSE  [43] "cadetblue1"      "cadetblue2"      "cadetblue3"
FALSE  [46] "cadetblue4"      "chartreuse"      "chartreuse1"
FALSE  [49] "chartreuse2"     "chartreuse3"     "chartreuse4"
FALSE  [52] "chocolate"       "chocolate1"      "chocolate2"
FALSE  [55] "chocolate3"      "chocolate4"      "coral"
FALSE  [58] "coral1"          "coral2"          "coral3"
FALSE  [61] "coral4"          "cornflowerblue"  "cornsilk"
FALSE  [64] "cornsilk1"       "cornsilk2"       "cornsilk3"
FALSE  [67] "cornsilk4"       "cyan"            "cyan1"
FALSE  [70] "cyan2"           "cyan3"           "cyan4"
FALSE  [73] "darkblue"        "darkcyan"        "darkgoldenrod"
FALSE  [76] "darkgoldenrod1"  "darkgoldenrod2"  "darkgoldenrod3"
FALSE  [79] "darkgoldenrod4"  "darkgray"        "darkgreen"
FALSE  [82] "darkgrey"        "darkkhaki"       "darkmagenta"
FALSE  [85] "darkolivegreen"  "darkolivegreen1" "darkolivegreen2"
FALSE  [88] "darkolivegreen3" "darkolivegreen4" "darkorange"
FALSE  [91] "darkorange1"     "darkorange2"     "darkorange3"
FALSE  [94] "darkorange4"     "darkorchid"      "darkorchid1"
FALSE  [97] "darkorchid2"     "darkorchid3"     "darkorchid4"
FALSE [100] "darkred"
```

Palettes

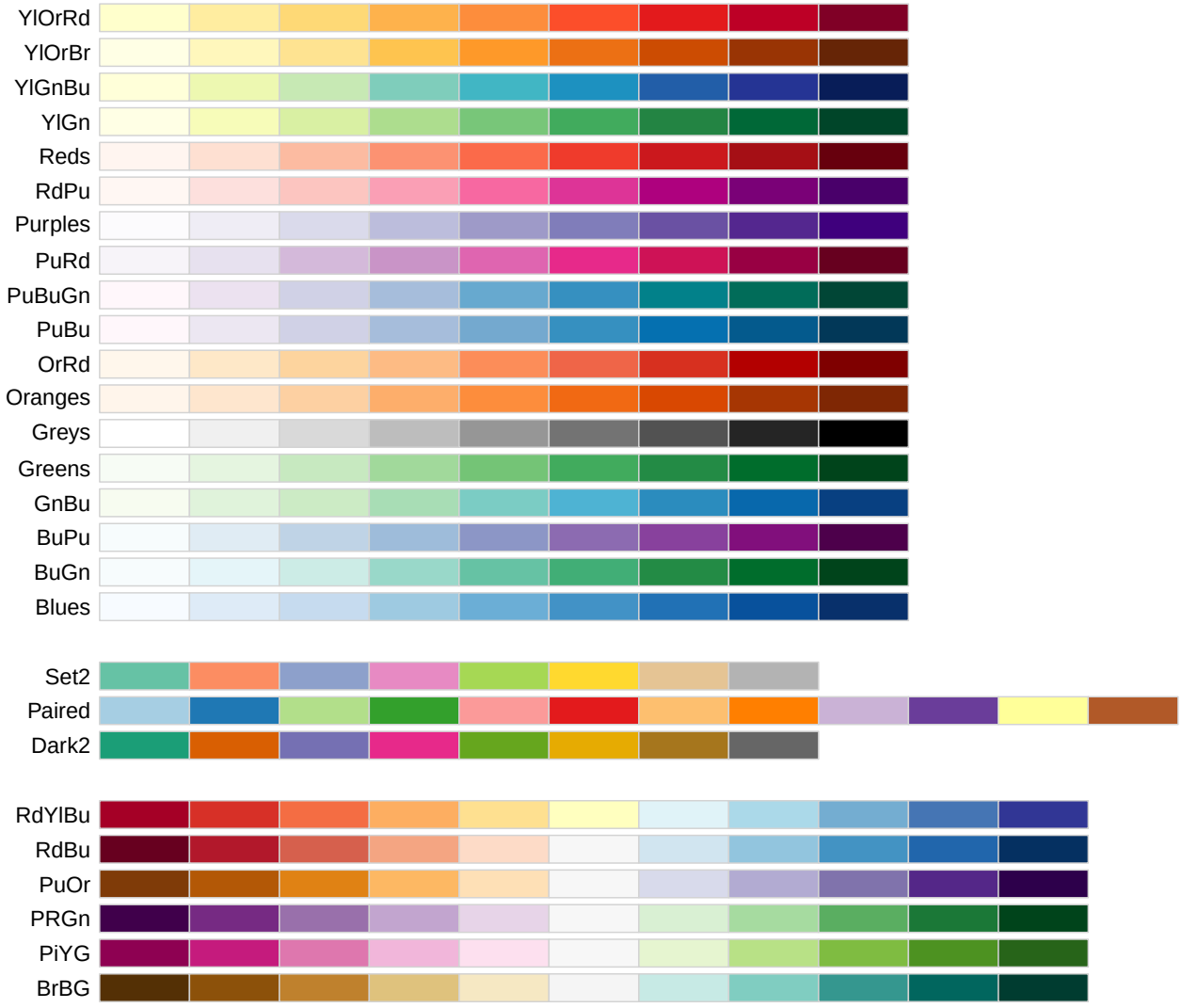
Several R packages propose color palettes. One particularly user-friendly R-package in this regard is RColorBrewer. The colors on the palettes are selected to fit well together for usage in different research contexts: heatmaps, topologies, color gradients, ...

```
library(RColorBrewer)
```

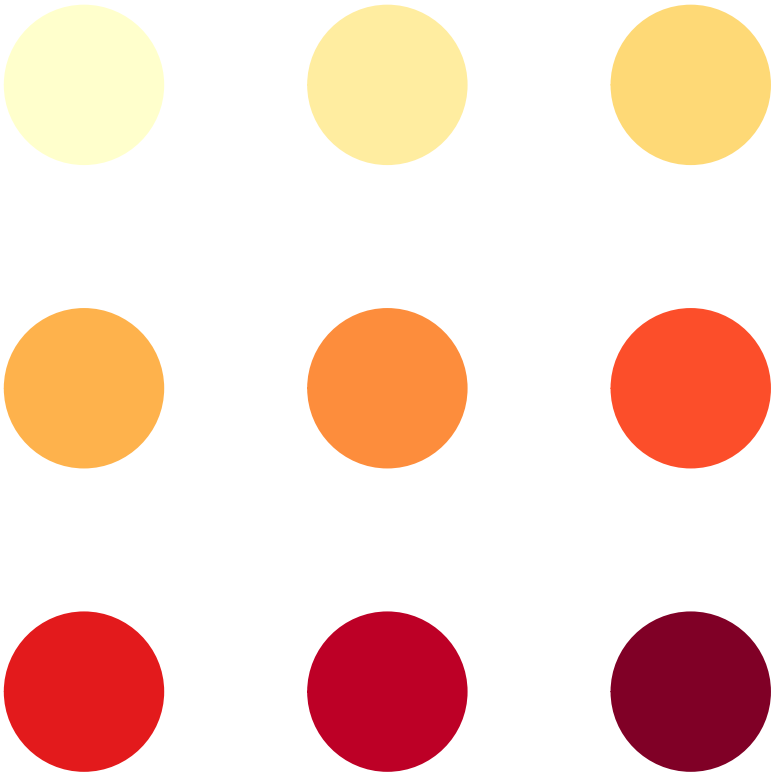
```
display.brewer.all()
```



```
display.brewer.all(colorblindFriendly = TRUE)
```



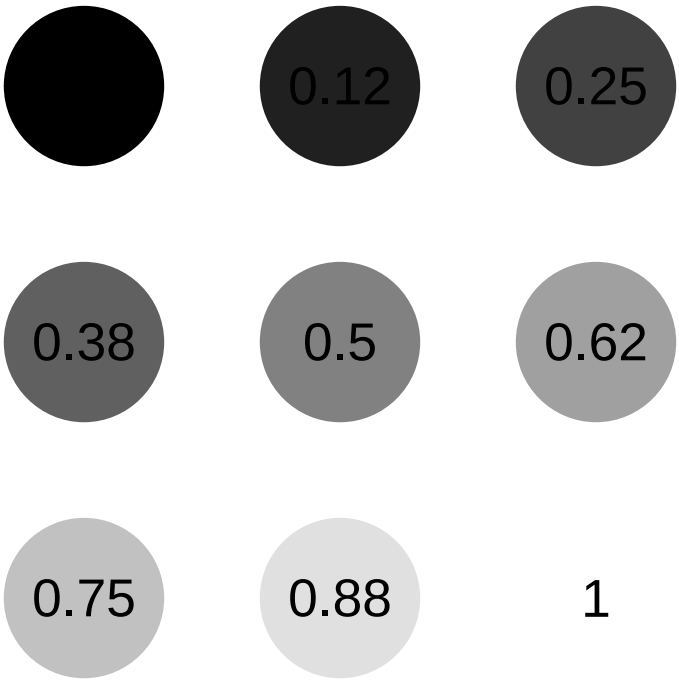
```r
# Example

par(mfrow=c(3,3))

for(i in 1:9){ # Loop (9 plots total)
# No figure margins (bottom, left, top, right), see ?par for details
par(mar=c(0,0,0,0))
plot(0.5, 0.5, # x and y coordinates
     pch=20, # Plotting character nbr. 20 (filled circle)
     cex=25, # Magnify point by a factor of 25
     xlim=c(0,1), ylim=c(0,1), # Range of x and y axes
     xlab="", ylab="", # No labels for axes
     xaxt="n", yaxt="n", # Axes type - "n" for no axes
     bty="n", # Type of box around plot - "n" for no box
     col=brewer.pal(n = 9, name = "YlOrRd")[i])  # Loop over RColorBrewer palette
  } # Close Loop
```



Transparence

Controlling for the transparency can be useful when visual information superposes. The library yarrr, a companion to the e-Book *YaRrr!: The Pirate's Guide to R*, contains the easy to use function *transparent()*. This function allows to create more transparent versions of a chosen color by entering its name or code and to indicate the degree of transparency from 0 (no transparence at all) to 1 (completely transparent).

```
library(yarrr)

# A sequence of 9 degrees of transparency

T=seq(0,1,1/8) # 9 step sequence from 0 to 1

par(mfrow=c(3,3))

for(i in 1:length(T)){
par(mar=c(0,0,0,0))
plot(0.5, 0.5,
     pch=20, cex=25,
     xlim=c(0,1), ylim=c(0,1),
     xlab="", ylab="",
     xaxt="n", yaxt="n",
     bty="n",
     # Transparency gradient for black
     col=transparent(orig.col="black", trans.val=T[i]))
  # Add text (the % of transparency) to the graphic
  text(0.5,0.5, labels=round(T[i],2), cex=2.5)
}
```



Mixing own colors

In principle, the color palette in R is large enough to meet a typical user's expectation. But in some cases, reports for institutions or companies, which have their own branding, making the layout recognizable by adressing their specific coloring scheme may be a plus.

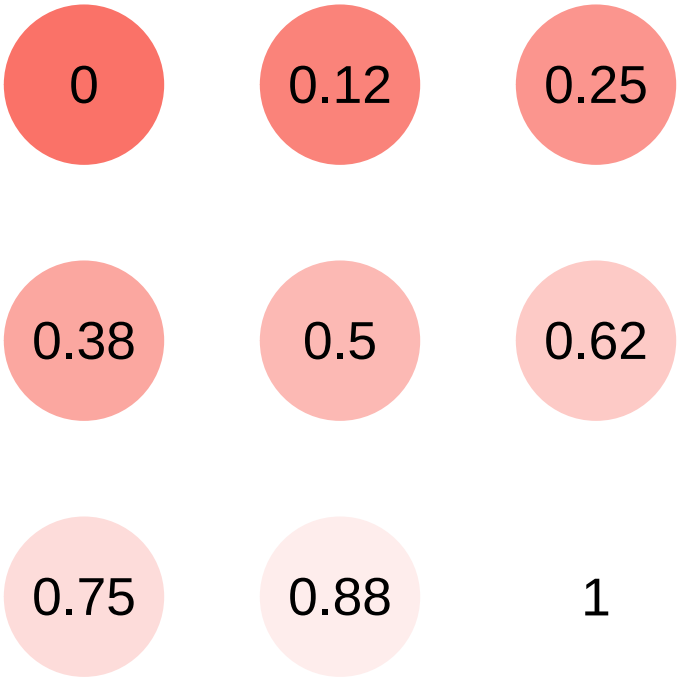
There are various ways to code colors. The hex(adecimal) and RGB systems are the most important ones in R. Colors in figures can be entered either with the color name or with the hex code. However, the hex code is not very intuitive. RGB is somewhat easier to grasp. RGB is an additive color model where the letters stand for Red, Green, and Blue and the code reflects their intensity, from 0 to full intensity. A broad array of colors can be implemented by combining these three primary colors.

R-users who want to programm colors and have a good intuition of mixing the three colors can try the function *rgb()* found in the R-package grDevices to translate the RGB code of a color into HEX to be used in plot functions.

Color codes for brands can be found either in RGB or HEX by searching the web.

The following example aims to reproduce the color of the year 2019 which was *Living Coral* by using the color RGB code found online.

```
# Programming one's own color or after having found its RGB composition

# Red green and blue composition of a color
Coral=rgb(250, 114, 104, maxColorValue = 255)

T=seq(0,1,1/8)

par(mfrow=c(3,3))

for(i in 1:length(T)){
par(mar=c(0,0,0,0))
plot(0.5, 0.5,
     pch=20, cex=25,
     xlim=c(0,1), ylim=c(0,1),
     xlab="", ylab="",
     xaxt="n", yaxt="n",
     bty="n",
     # Replace black by Coral
     col=transparent(orig.col=Coral, trans.val=T[i]))
  text(0.5,0.5, labels=round(T[i],2), cex=2.5)
}
```




```
# Exercise

# How does the color of the year 2020 look like in R?
# See RGB / HEX here: https://www.pantone.com/color-finder/19-4052-TCX
```

Plots

When doing plots of data in R a few functions are probably the most central: *plot()*, *barplot()*, *hist()*, *boxplot()*. Another important command, which is not a function, is *par*. When it comes to fine-tuning plots, starting the search by typing *?par* in the console is very likely to lead to a response.

Saving plots

Plots are saved with functions *pdf()*, *jpeg()*, *tiff()*, *postscript()*, *save_html()* from the htmltools package.

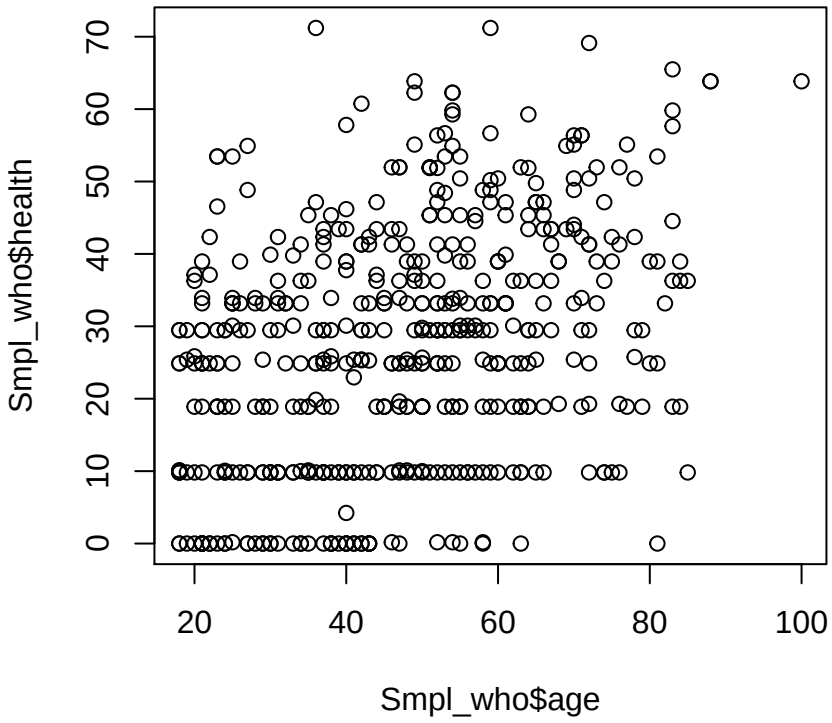
Type *?png*, *?pdf*, *?postscript* on how to use the functions. In general, it requires only to put the function before the plotting function and to close the function, i.e. stop saving, at the end off the plot with *dev.off()* or *graphics.off()*. Saving a plot requires mainly the specification of 1) the path to output 2) a file name and 3) the file type (.pdf, .jpeg, etc. . .). Further options for customization are available (e.g. the size of the figure, formatting options such as resolution, page dimensions e.g. A4 or US letter).

```
# 1) path_output has been set earlier,
# 2) name of the figure: Plot_to_save
# 3) and file type .pdf

pdf(paste(path_output, "Plot_to_save.pdf", sep=""), paper="a4")

# Copy paste an entire plot code here

dev.off() # Closes the pdf window
```

Basic plotting options

Let's explore different plotting options with a scatterplot of the variables age and health to illustrate, step by step, essential plotting options.

```
# Starting with the R-output of an x against y plot
#  where no further specifications are provided.

plot(x=Smpl_who$age, y=Smpl_who$health) # Zero adjustments plot
```



Axis range

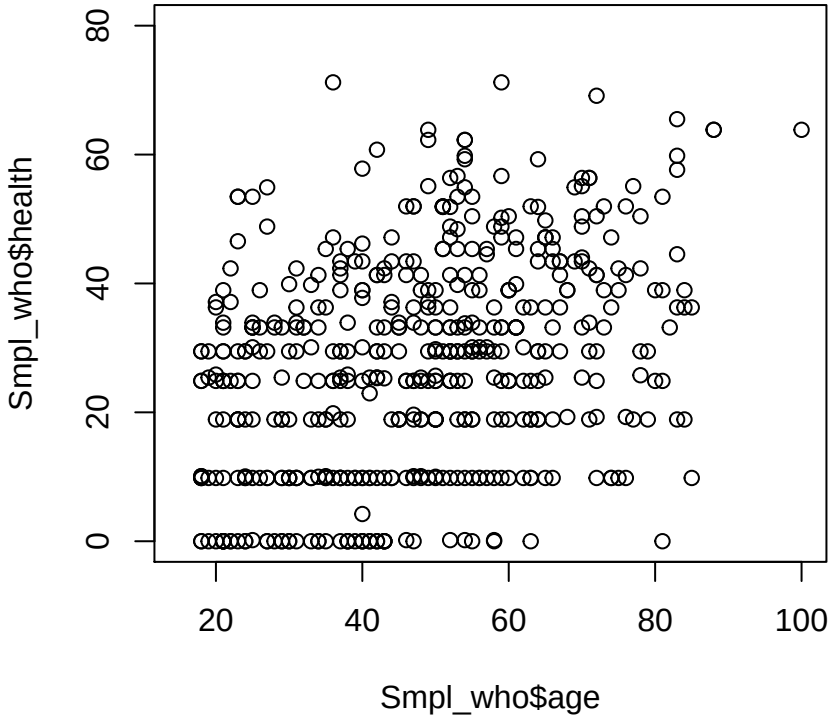
```
# Setting the ranges of the axes with xlim and ylim
range(Smpl_who$age)
```

```
[1]  18 100
```

```
range(Smpl_who$health)
```

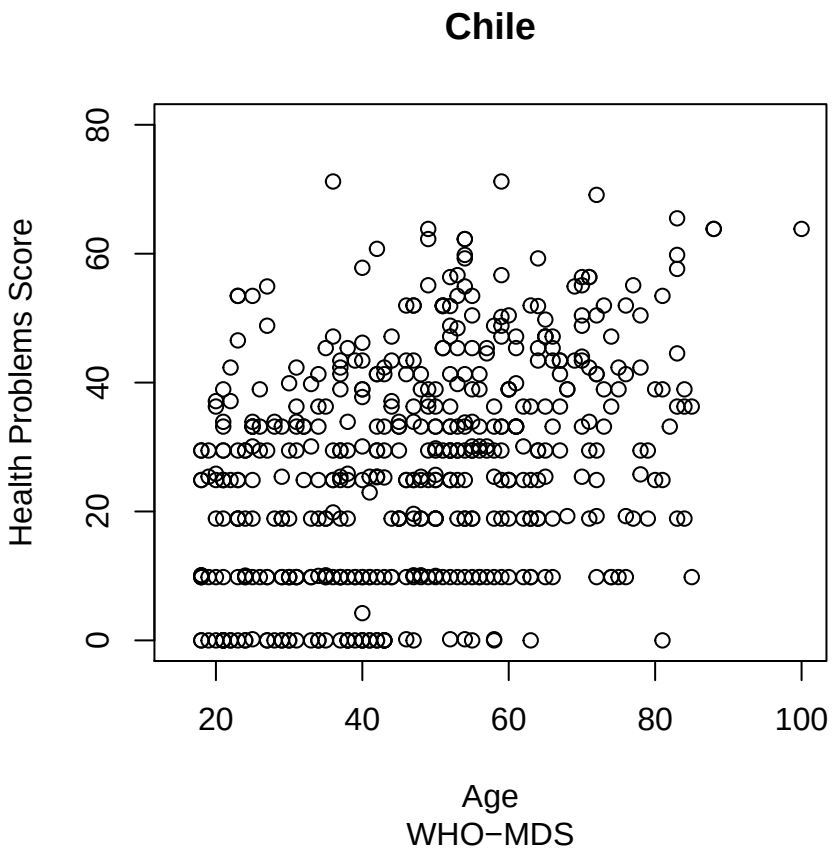
```
[1]  0.00000 71.20512
```

```
plot(x=Smpl_who$age, y=Smpl_who$health, # x and y coordinates
     xlim=c(15, 100),  # Age ranges from 18 to 100
     ylim=c(0, 80)) # True range of the measure is 0 to 100
```



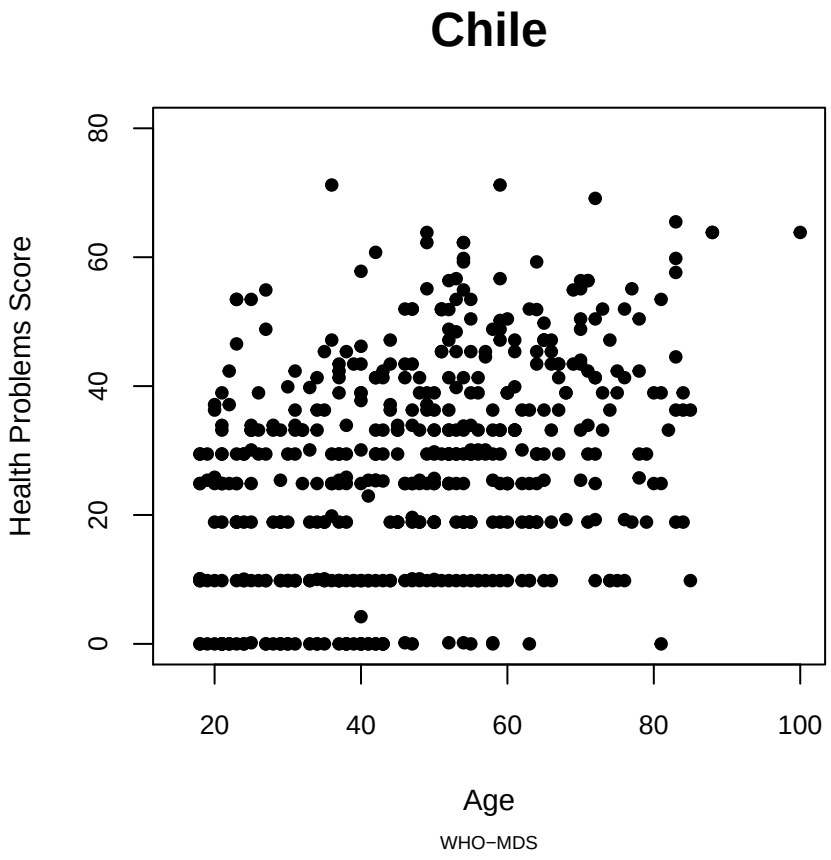
Title, subtitles, labels

```
plot(x=Smpl_who$age, y=Smpl_who$health,
     xlim=c(15,100), ylim=c(0,80), # Axes range
     xlab="Age", # Axis label
     ylab="Health Problems Score", # Axis label
     main="Chile", # Main title
     sub="WHO-MDS") # Subtitle
```



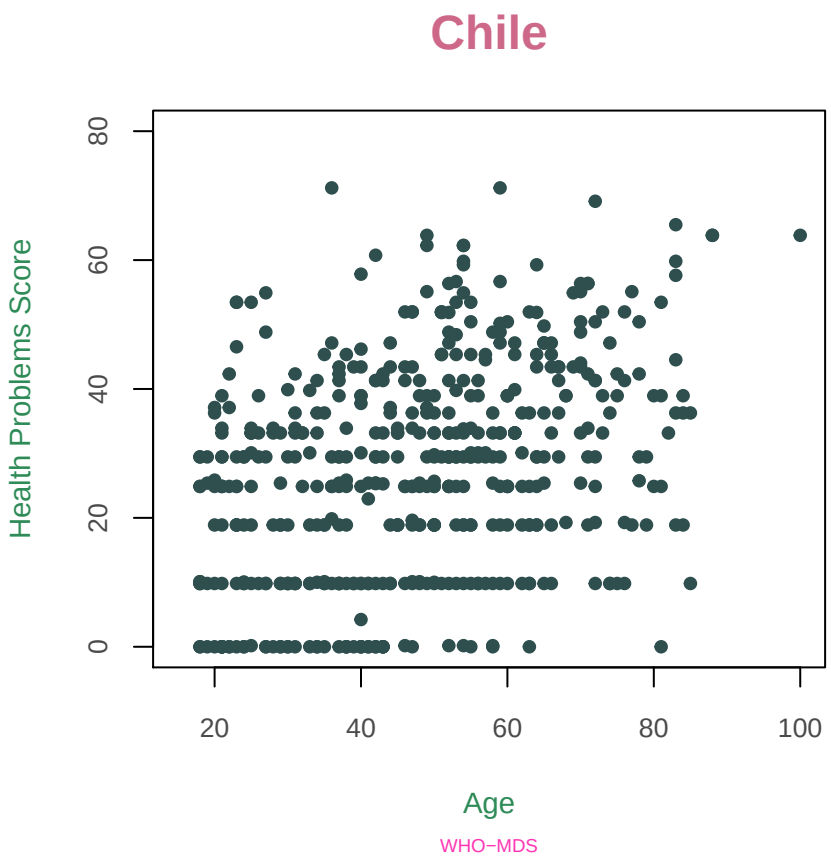
Character size (expansion)

```
# cex=1 does not change size, < 1 downsizes, > 1 enlarges
plot(x=Smpl_who$age, y=Smpl_who$health,
     xlim=c(15,100), ylim=c(0,80),
     xlab="Age",
     ylab="Health Problems Score",
     main="Chile",
     sub="WHO-MDS",
     cex.lab=0.9, # Downsizing axis labels
     cex.main=1.5, # Enlarging main title
     cex.sub=0.6, # Downsizing the subtitle
     cex.axis=0.8,  # Downsizing axis numbering
     cex=0.8, # Downsizing the points in the plot
     pch=19) # Change point type makes it look already a little bit better ?points
```



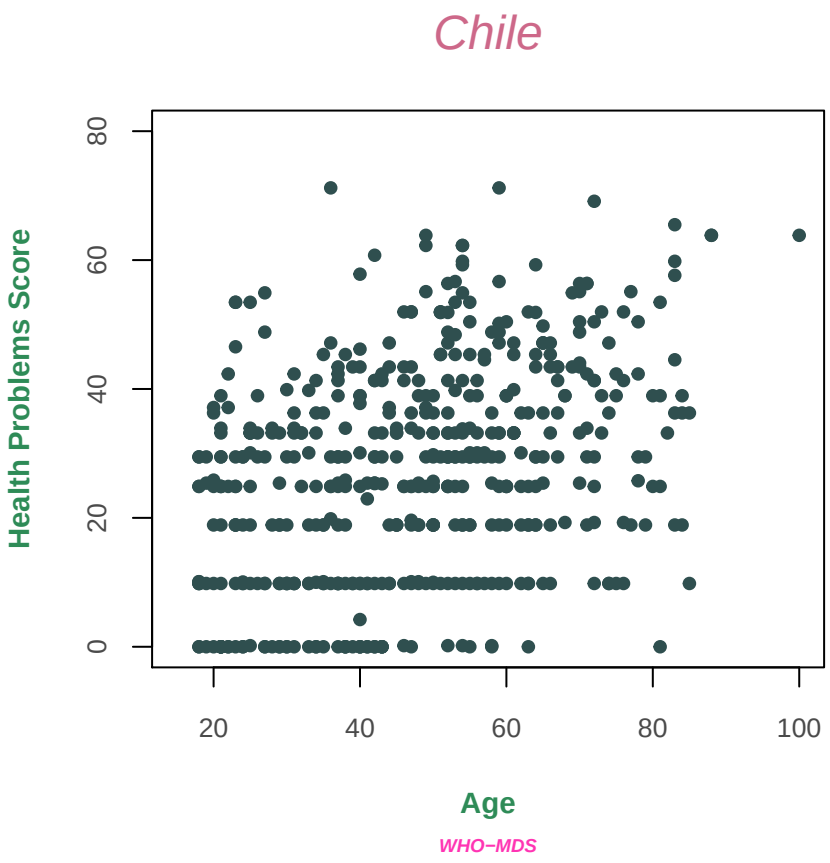
Coloring

```
# For the coloring, same command strategy col. + axis, lab, main, sub

plot(x=Smpl_who$age, y=Smpl_who$health,
     xlim=c(15,100), ylim=c(0,80),
     xlab="Age",
     ylab="Health Problems Score",
     main="Chile",
     sub="WHO-MDS",
     cex.lab=0.9, col.lab="seagreen4", # Downsizing & coloring axis labels
     cex.main=1.5, col.main="palevioletred3", # Enlarging & coloring main title
     cex.sub=0.6, col.sub="maroon1", # Downsizing & coloring the subtitle
     cex.axis=0.8, col.axis="gray29",  # Downsizing & coloring axis numbering
     cex=0.8, col="darkslategrey", # Downsizing & coloring the points
     pch=19) # Putting bullets
```



Font

```
# For the font, same command strategy font. + axis, lab, main, sub

plot(x=Smpl_who$age, y=Smpl_who$health,
     xlim=c(15,100), ylim=c(0,80),
     xlab="Age",
     ylab="Health Problems Score",
     main="Chile",
     sub="WHO-MDS",
     cex.lab=0.9, col.lab="seagreen4", font.lab=2, # Bold
     cex.main=1.5, col.main="palevioletred3", font.main=3, # Italic
     cex.sub=0.6, col.sub="maroon1", font.sub=4, # Bold and italic
     cex.axis=0.8, col.axis="gray29", font.axis=1,  # Standard font
     cex=0.8, col="darkslategrey",
     pch=19)
```

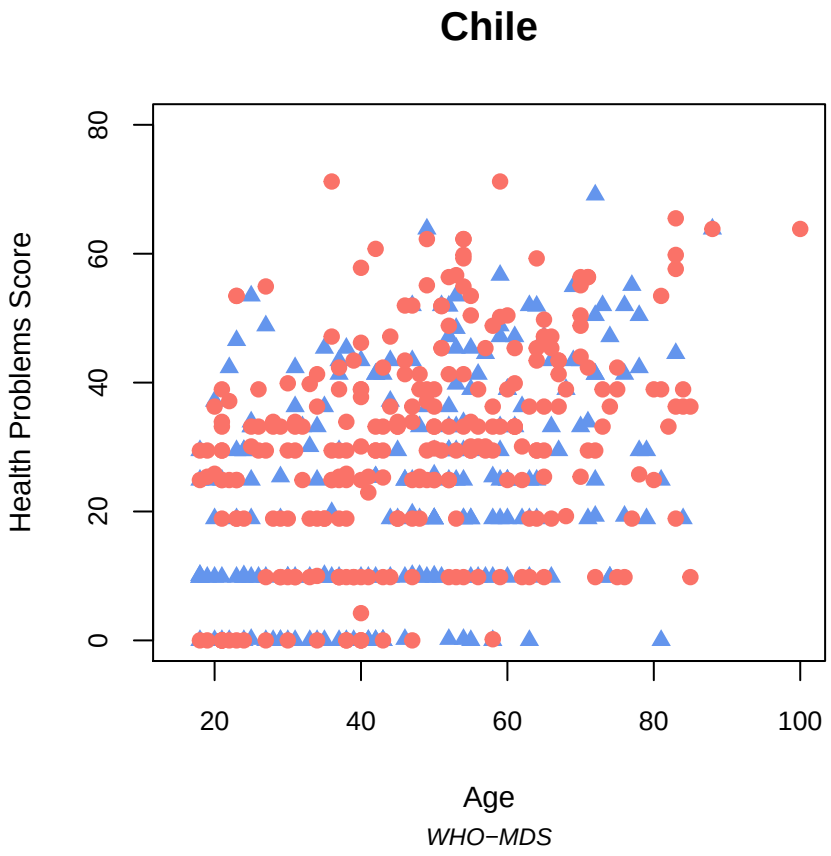


Visually speaking, the last plot here was not a hit, but helped to introduce some basic plotting options. Let's try to make the scatterplot more informative.

Points & symbols with pch

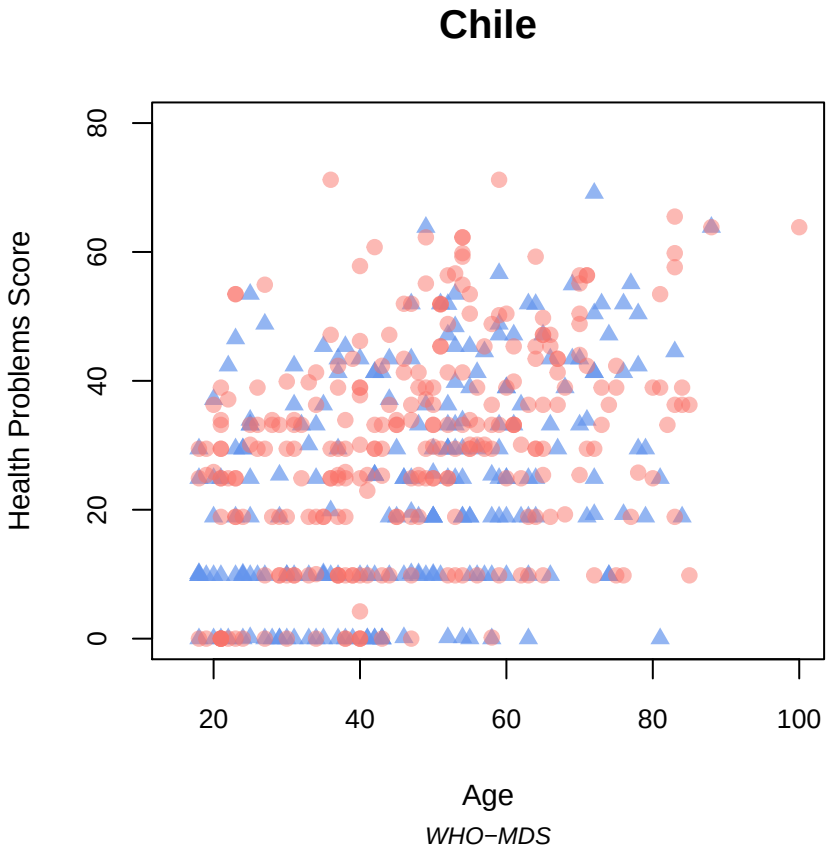
Next, use different symbols for male and females with the option *pch*. Now, the plot has to be constructed in a two-step approach, first by plotting the male subgroup, second by adding points for the female subgroup.

```
Smpl_who_male=subset(Smpl_who, sex=="male")
Smpl_who_female=subset(Smpl_who, sex=="female") # Could also be written sex!="male"


# First: make a plot for the male subsample
plot(x=Smpl_who_male$age, y=Smpl_who_male$health,
     xlim=c(15,100), ylim=c(0,80),
     cex.axis=0.85,
     xlab="Age", cex.lab=0.9,
     ylab="Health Problems Score",
     main="Chile", cex.main=1.25, font.main=2,
     sub="WHO-MDS", cex.sub=0.75, font.sub=3,
     pch=17, col="cornflowerblue", # Blue triangles
     cex=1)

# Second: add to the plot the points of the female subsample
points(x=Smpl_who_female$age, y=Smpl_who_female$health,
       pch=19, col=Coral,  # Living coral circles
       cex=1)
```



As the points overlap, the function *transparent()* presented earlier maybe helpful to improve the visualization.

```
# First: make a plot for the male subsample
plot(x=Smpl_who_male$age, y=Smpl_who_male$health,
     xlim=c(15,100), ylim=c(0,80),
     cex.axis=0.85,
     xlab="Age", cex.lab=0.9,
     ylab="Health Problems Score",
     main="Chile", cex.main=1.25, font.main=2,
     sub="WHO-MDS", cex.sub=0.75, font.sub=3,
     pch=17, cex=1,
     col=transparent(orig.col="cornflowerblue", trans.val=0.3)) # Transparency 30%

# Second: add to the plot the points of the female subsample
points(x=Smpl_who_female$age, y=Smpl_who_female$health,
       pch=19, cex=1,
       col=transparent(orig.col=Coral, trans.val=0.5)) # Transparency 50%
```



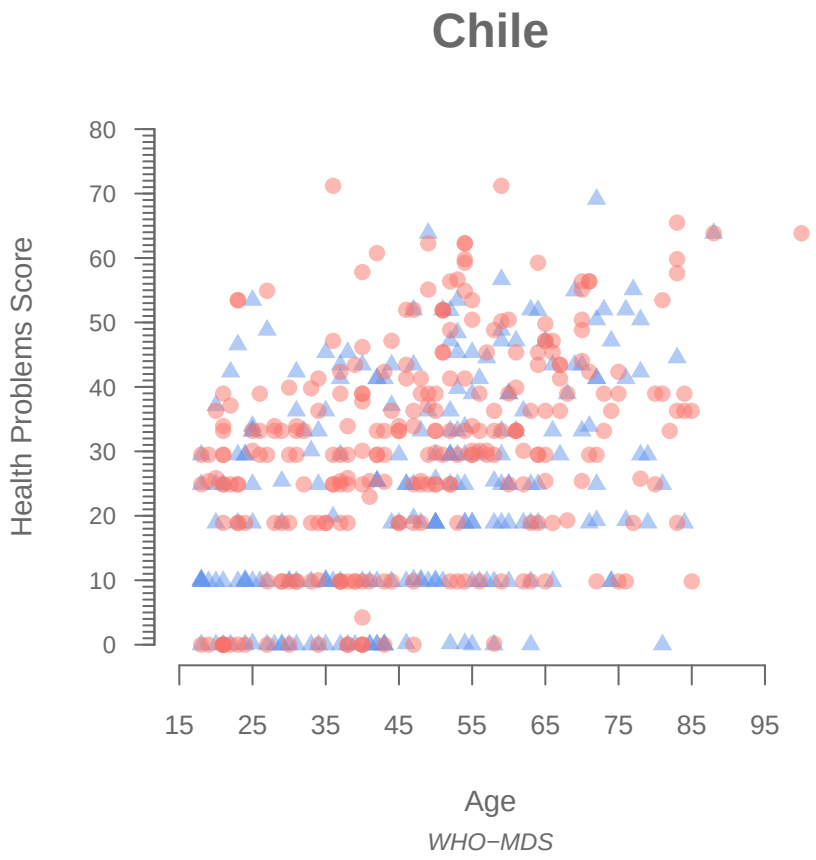

```
# Exercise

# Try out the different point types by using pch
# Check ?points to see the pch options displayed
# Vary the size, color and transparency of the plotting symbols
```

Axes

Axes are useful when magnitudes such as frequencies, proportions or effects need to be shown in relation to a metric. The *plot()* function automatically puts axes by default. However, the formatting of axes is sometimes better done outside of the plot. This can be achieved by first drawing a plot, which supresses the axes. The axes can then be re-entered separately in additional steps with the R-function *axis()*. A relatively wide serie of options are available for customization: axis position, tick position, tick size, tick width, direction of labels, line types, colors, font-related options and more (see *?axis* for all the options).

```
# Continuing with previous plot but supressing the axes and frame

plot(x=Smpl_who_male$age, y=Smpl_who_male$health,
     xlim=c(15,100), ylim=c(0,80),
     cex.axis=0.85,
     xlab="Age", cex.lab=0.9, col.lab="dimgrey",
     ylab="Health Problems Score",
     main="Chile", cex.main=1.5, font.main=2, col.main="dimgrey",
     sub="WHO-MDS", cex.sub=0.75, font.sub=3, col.sub="dimgrey",
     pch=17,
     col=transparent(orig.col="cornflowerblue", trans.val=0.5),
     bty="n", # Suppresses the box around the plot
     xaxt="n", yaxt="n")  # Suppresses the axes

# Add to the plot the points of the female subsample
points(x=Smpl_who_female$age, y=Smpl_who_female$health,
       pch=19,
       col=transparent(orig.col=Coral, trans.val=0.5))

# Now let's put some axes
# The number coming after the command indicates the side, it always starts
#  below and goes clockwise 1=bottom, 2=left, 3=top, 4=right

# The x axis is below the plot and we want a tick for every 10 years
# Start by checking the range of ages

axis(1, at=seq(15,100,10), labels=TRUE,
     cex.axis=0.85,
     col="dimgrey",
     col.ticks="dimgrey", # Color of tick marks
     col.axis="dimgrey") # Color of the axis

# The y axis will be on the left and we want a tick for every 10 years
# Start by checking the range of health

axis(2, at=seq(0, 80, 10),
     labels=TRUE, # Add labels to each tick (default: TRUE)
     cex.axis=0.75,
     las="1", # Orientation of the labels 1=horizontal
     col="dimgrey",
     col.ticks="dimgrey",
     col.axis="dimgrey")

# By varying the size of the ticks a metric type of ruler could be done, why not.
# Putting a more granular y-axis with smaller ticks - option tck
```

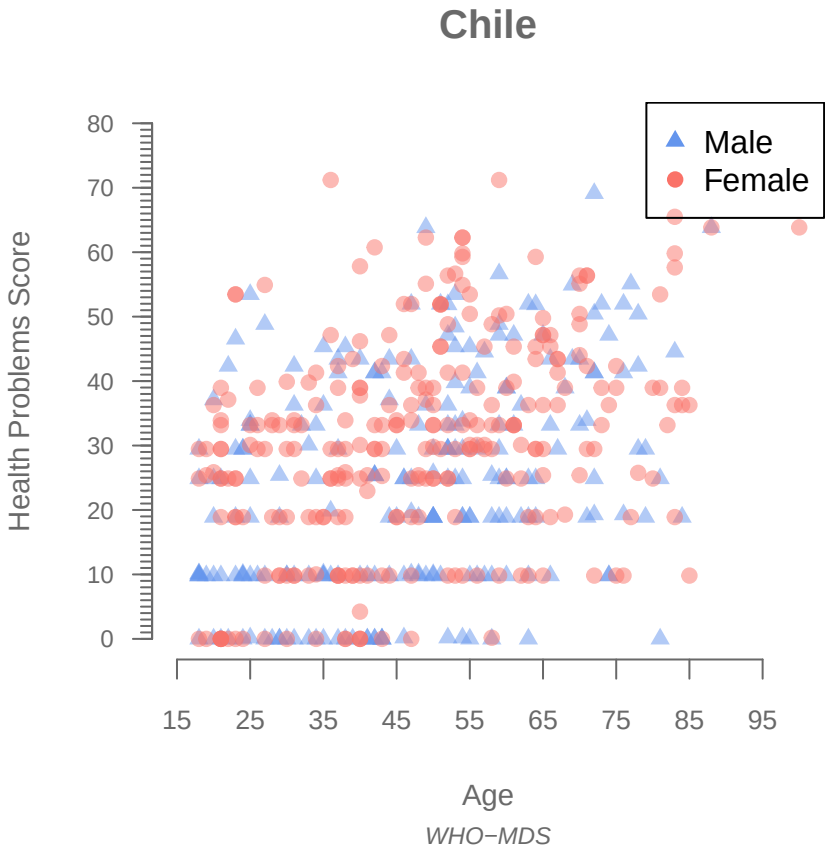
```
axis(2, at=seq(0, 80, 1), labels=FALSE, # Plot 80 ticks each 1 unit apart
     tck=-0.02, # shorter tick mark length
     cex.axis=0.85,
     col="dimgrey",
     col.ticks="dimgrey")
```



Legend

Similar to *axis()* and *points()*, the *legend()* can be added ad hoc to the plot as a separate function.

```
# Adding the legend
legend("topright", c("Male", "Female"),
       pch=c(17,19),  # Plotting symbols
       col=c("cornflowerblue", Coral)) # Colors of the plotting symbols
```

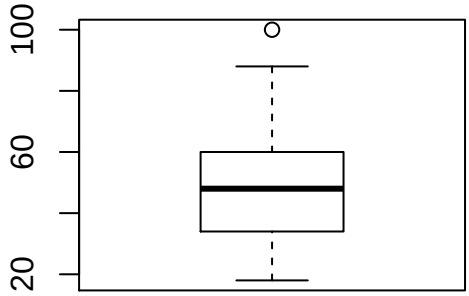



```
# Exercise

# The legend looks still terrible let's remove the box
# Reformat the legend in an appropriate size
# Reposition the legend using the coordinate system if required
# Add transparency to the legend points
```

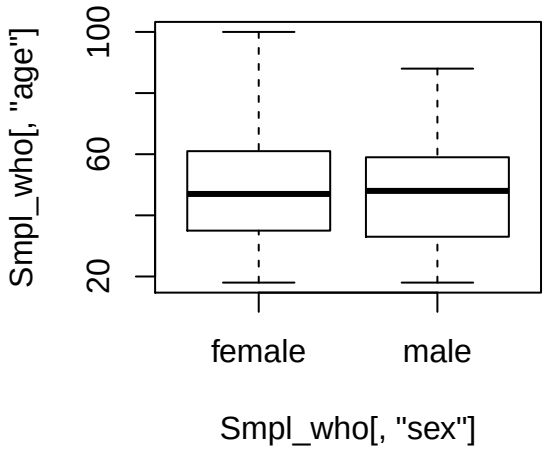
Boxplots

A boxplot allows displaying the essential descriptive statistics for continuous data with a five number summary (minimum, first quartile (Q1), median, third quartile (Q3), and maximum). The boxplot also shows outliers. However, it does not allow to check if data is normally distributed or not.

```
# Boxplot of age
boxplot(Smpl_who[,"age"])
```




```
# Boxplot of age and sex
boxplot(Smpl_who[,"age"]~Smpl_who[,"sex"])
```



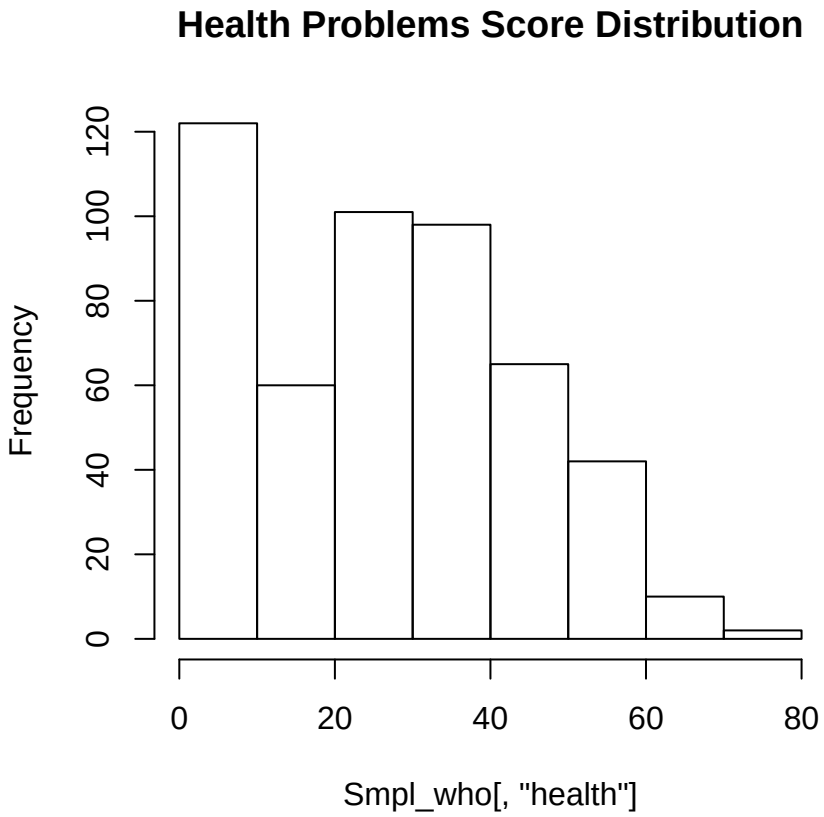

```
# Remark: the same function could also be written as:
boxplot(Smpl_who$age~Smpl_who$sex)
boxplot(age~sex, data=Smpl_who)

# Exercise

# Add a title for the second plot "Age Quantiles by Gender"
# Add colors to the plot, for example in lightpink1 for the females and
#  cornflowerblue for the males,

# Do a similar plot to display the health problem scores
```

Histograms and barplots

The function *hist()* plots a histogram for a vector of values. The function *lines()* allows to add a line to the plot. The function *density()* calculates the coordinates for the corresponding density line. Also, normal distribution curves can be added to make the histogram even more informative with *dnorm()*.

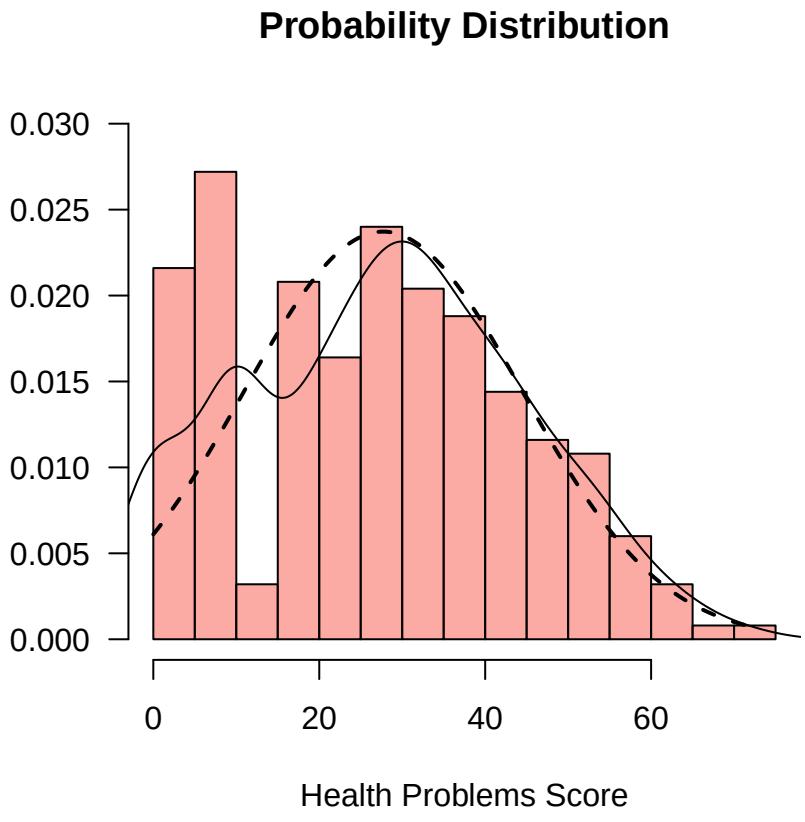
For factor and character variables the function *barplot()* is the better plotting function to represent the frequencies of variable levels with bars.

```
# Frequency distribution for numeric/continuous variables
hist(Smpl_who[,"health"], main="Health Problems Score Distribution")
```



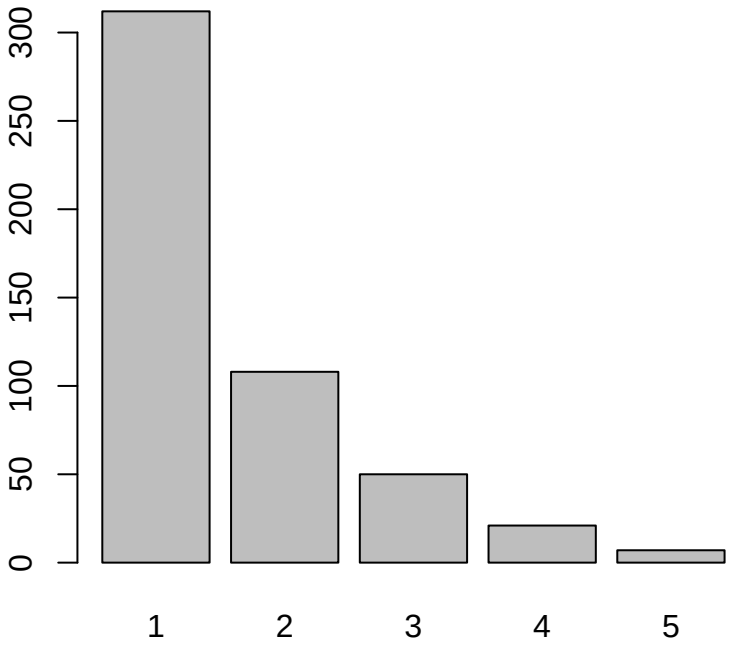
Format the histogram

```
# Probability distribution for numeric/continuous variables
hist(Smpl_who[,"health"], main="Probability Distribution",
       probability=TRUE, # Probability distribution
       ylim=c(0,0.03), # Adjusting the height of the y-axis
       xlab="Health Problems Score", # Putting an x-axis label
       ylab="",
       las=1, # Setting the y-axis labels to be horizontal
       breaks=15, # Specifiying the number of columns in the histogram
       col=transparent(orig.col=Coral, trans.val=0.4), # Color the columns
       border=NULL) # Remove the borders

# Density line (solid)
lines(density(Smpl_who[,"health"]), col = "black")  # Adding a density line


# Adding a normal distribution curve (dashed)
#   to visualize the departure from the normal distribution
xfit=seq(min(Smpl_who[,"health"]),max(Smpl_who[,"health"]),length=nrow(Smpl_who))
yfit=dnorm(xfit,mean=mean(Smpl_who[,"health"]),sd=sd(Smpl_who[,"health"]))

lines(xfit, yfit, col="black", lwd=2, lty="dashed")
```



Draw a barplot

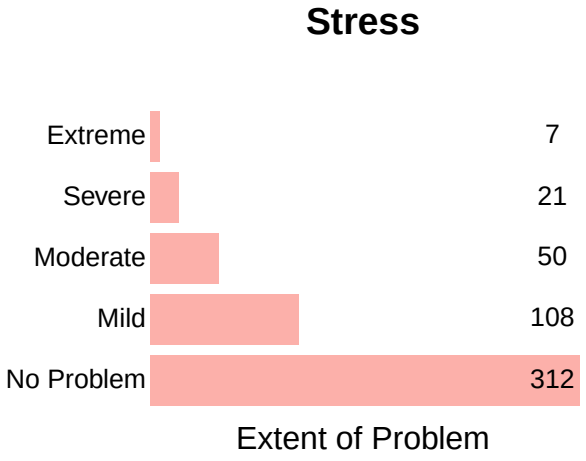
```
# Plot of heights, i.e. frequencies for factor variables with barplot
barplot(table(Smpl_who[,"stress"]))
```



Text

The *text()* function allows to put text on plots whereas *mtext()* can be used to add text in the margins of a plot.

```
# Starting with previous plot without characteristics to illustrate text functions
barplot(table(Smpl_who[,"stress"]),
        col=transparent(orig.col=Coral, trans.val=0.44),# Coral color
        border=FALSE, # No border around the columnns
        axes=FALSE, # No axes
        horiz=TRUE, # Displaying the bars horizontally
        names.arg="", # Removing the labels
        main="Stress") # Putting a title

# Adding text in and next to the columns
Labels=c("No Problem", "Mild", "Moderate", "Severe","Extreme")

# Adding text in the margins
mtext(text="Extent of Problem", side=1) # Adding a subtitle, side=1)

mtext(text=Labels, side=2, at=c(0.75, 1.95, 3.15, 4.35, 5.55), # Finding at= positions
                                                               # is trial & error
      cex=0.8, # Label size
      las=1) # Text shown horizontally

# Frequency of observations
N=table(Smpl_who[,"stress"])

# Add frequency x-coordinate set a little bit below the highest frequency
text(x=rep(max(N)-20, 5), y=c(0.75, 1.95, 3.15, 4.35, 5.55), labels=N, cex=0.8,
     col="black")
```



Plots from scratch

Layout

Another important graphical feature is the *layout()* function which allows to divide the graphical space into fields. It is very useful to create figures composed of several plots.

layout() requires the user to define a matrix where the number stands for the position of each figure in the graphical display, and its relative size. More specifically, the numbers determine the order in which the plots enter the graphical display. A size ratio can also be entered separately with the options widths and heights.

To divide the graphical space in 4 equally sized fields where the plots will be entered starting from top left (1) until bottom right (4) the user will have to define a matrix of the form:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

To start entering the plots from bottom left (1) to top left (4), the user will have to define a matrix of the form:

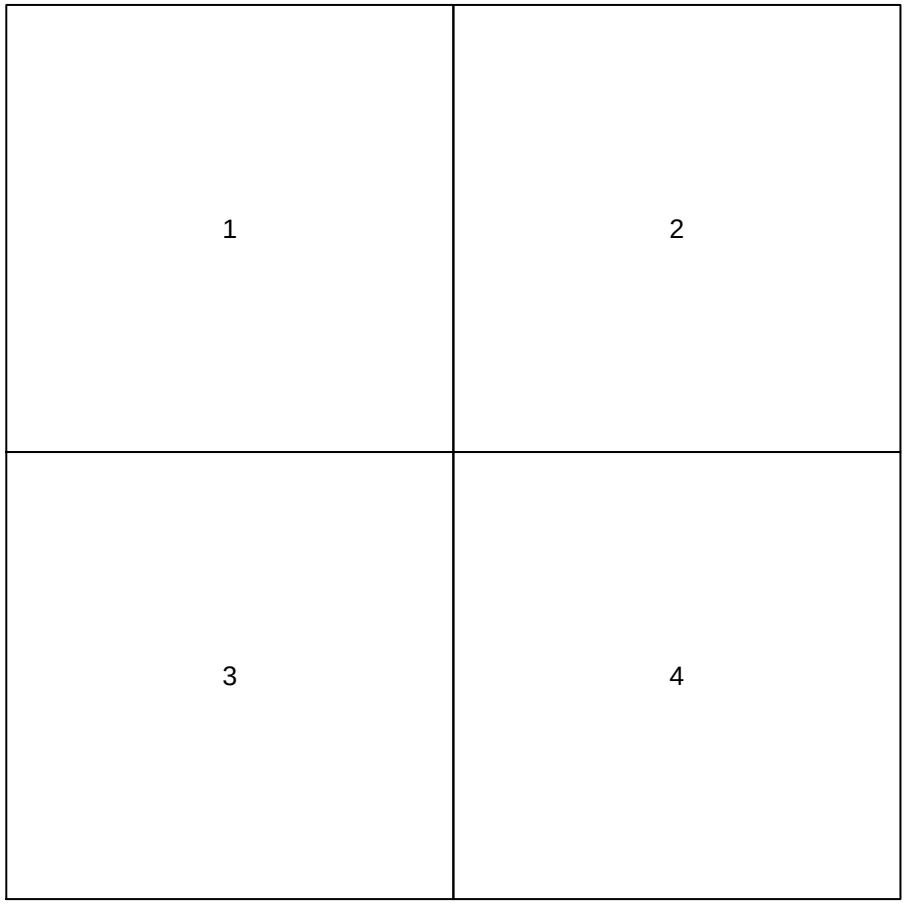
$$\begin{bmatrix} 4 & 3 \\ 1 & 2 \end{bmatrix}$$

An asymetric arrangement is also possible. For example, if one wants to draw a window with three plots on the left hand side (1,2,3) and two plots on the right hand side (4,6) with a small space between the two plots to add text (5), the user could proceed as follows:

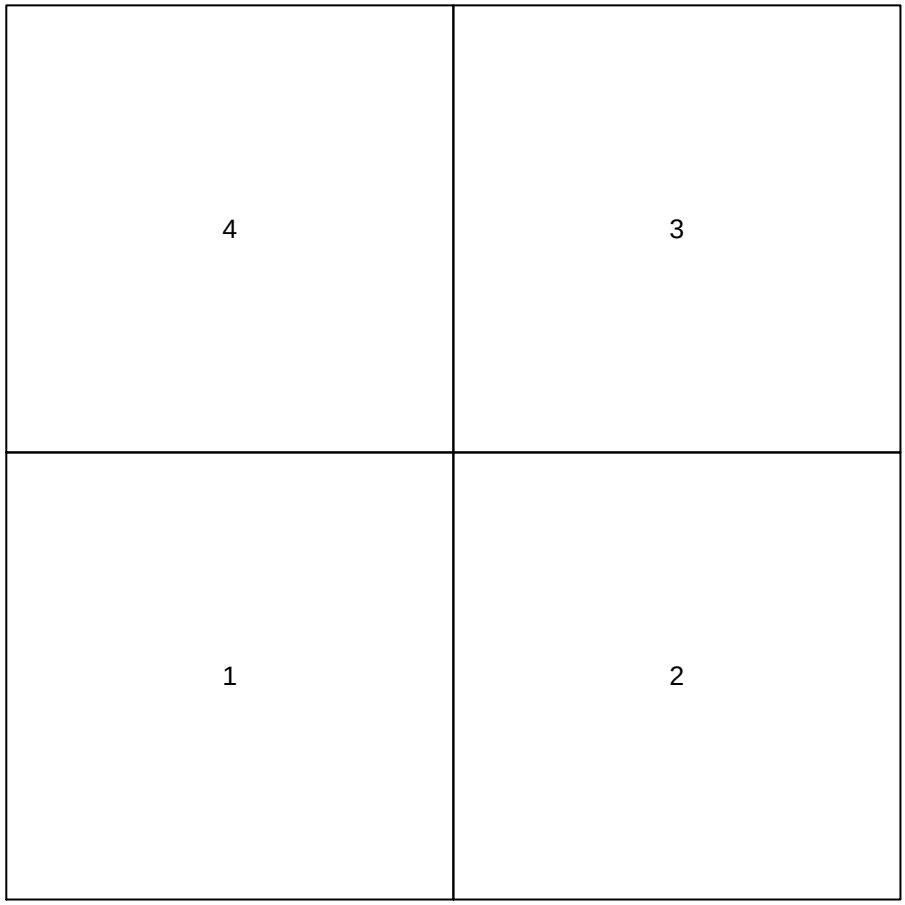
$$\begin{bmatrix} 1 & 4 \\ 1 & 4 \\ 1 & 4 \\ 2 & 4 \\ 2 & 5 \\ 2 & 6 \\ 3 & 6 \\ 3 & 6 \\ 3 & 6 \end{bmatrix}$$

The function *matrix()* allows to draw the design matrices to divide the graphical space. The function *layout.show()* allows to visualize the intended display.

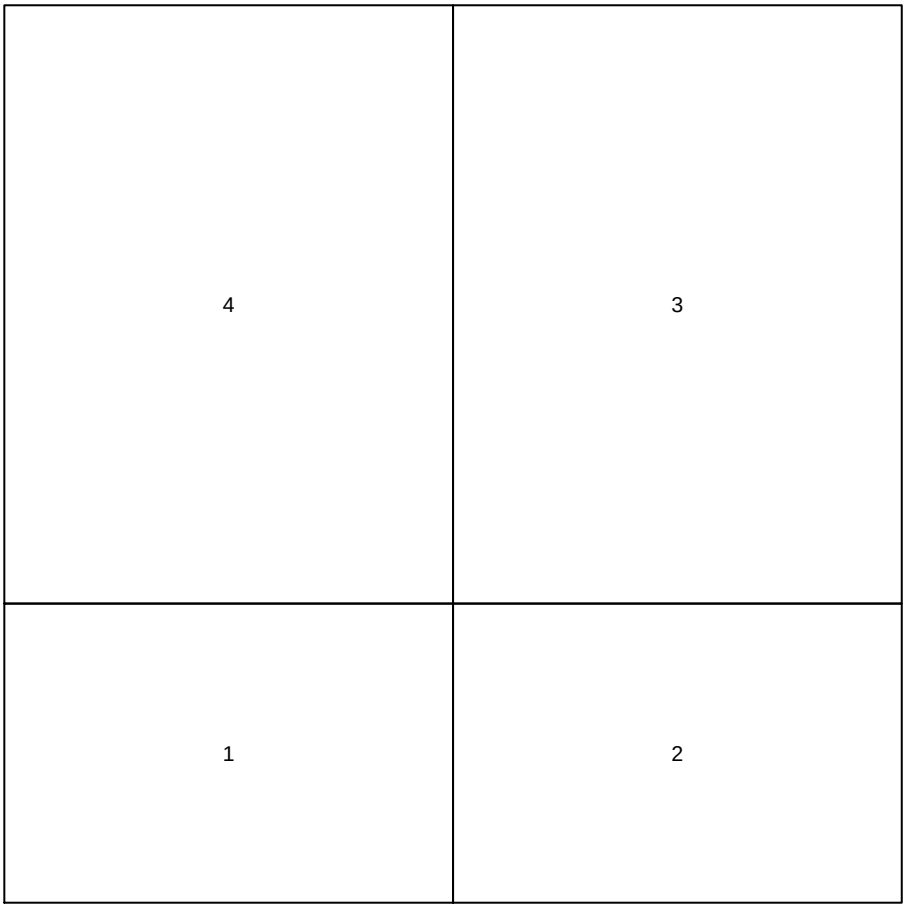
```
par(oma=c(.25,.25,.25,.25)) # Add margins to print entire plot

# 4 plotting fields from top left to bottom right
mat1=matrix(c(1,2,3,4), byrow=TRUE, ncol=2)
layout(mat1)
layout.show(n=4)
```



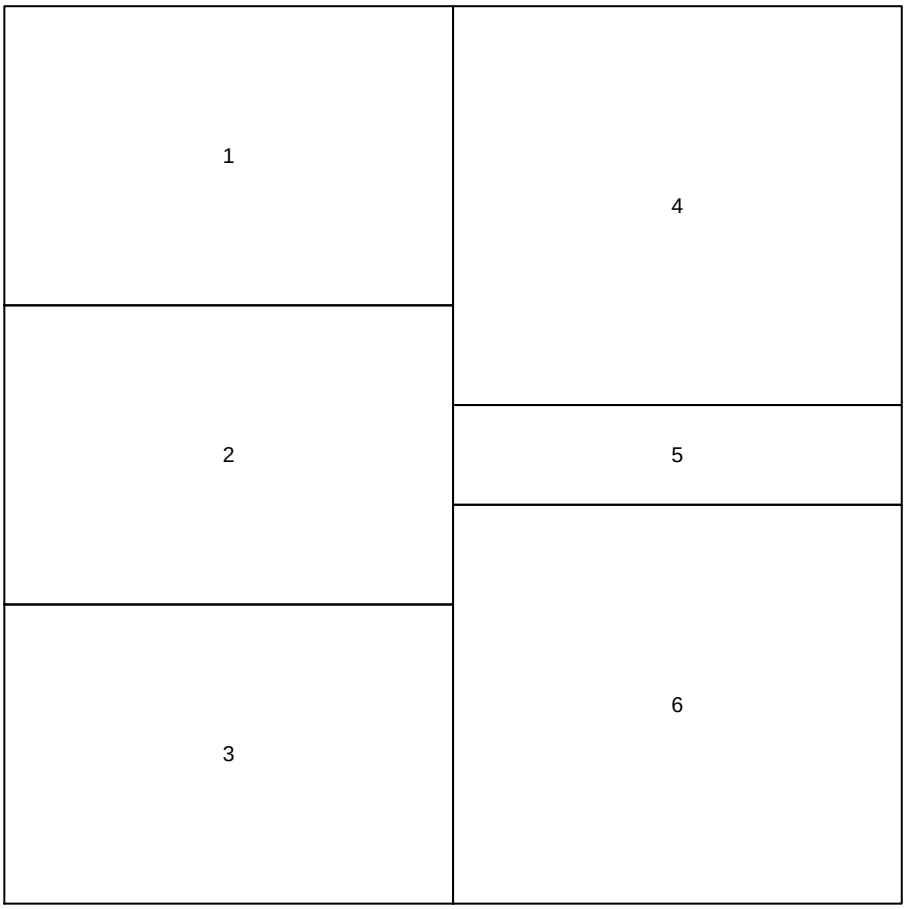
```r
par(oma=c(.25,.25,.25,.25)) # Add margins to print entire plot

# 4 plotting fields from bottom left to top left
mat2=matrix(c(4,3,1,2), byrow=TRUE, ncol=2)
layout(mat2)
layout.show(n=4)
```



```r
# 4 plotting fields from bottom left to top left, top plots being twice as large

par(oma=c(.25,.25,.25,.25)) # Add margins to print entire plot

mat3=matrix(c(4,3,4,3, 1,2), byrow=TRUE, ncol=2)
layout(mat3)
layout.show(n=4)
```



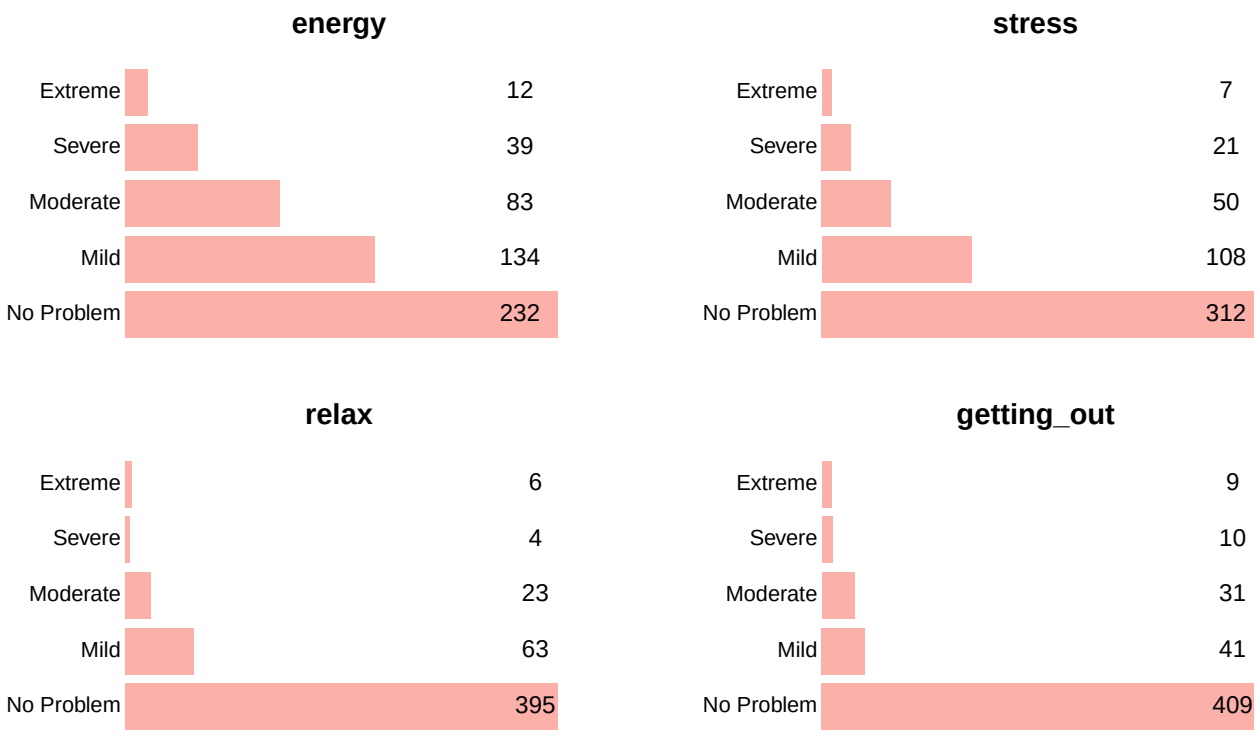
```
# 6 plotting fields 3 equal size on the left, two on the right
#   separated by a text field

par(oma=c(.25,.25,.25,.25)) # Add margins to print entire plot
mat4=matrix(c(1,4,
              1,4,
              1,4,
              2,4,
              2,5,
              2,6,
              3,6,
              3,6,
              3,6), byrow=TRUE, ncol=2)

layout(mat4)
layout.show(n=6)
```




```
graphics.off()  # This command allows to close all plotting windows in memory at once
                # Useful when running plots in loops, check also dev.off() to
                #  only close the last window.
```

The filling of the plotting fields is easy, it simply requires to draw the plots and they are entered automatically in the order set with *layout()*. Let's use a simple 2x2 display for a quick illustration.

```
# Plot 4 barplots into the layout:
mat=matrix(c(1,2,3,4), byrow=TRUE, ncol=2)
layout(mat)

# In fields 1, 2, 3 and 4: barplots for energy, stress, relax and getting_out
Labels=c("No Problem", "Mild", "Moderate", "Severe","Extreme")
Variable=c("energy", "stress", "relax", "getting_out")

for(i in 1:length(Variable)){
par(mar=c(1,7,2.5,2))
 barplot(table(Smpl_who[,Variable[i]]),
        col=transparent(orig.col=Coral, trans.val=0.44), # Coral color
        border=FALSE, # No border around the columnns
        axes=FALSE, # No axes
        horiz=TRUE, # Displaying the bars horizontally
        names.arg="", # Removing the labels
        main=Variable[i], # Putting a title
        cex.main=1.25) # Resizing title
 # Adding text in the margins
mtext(text=Labels, side=2, at=c(0.75, 1.95, 3.15, 4.35, 5.55),
      cex=.75, # Label size
      las=1) # Text shown horizontally
# Adding frequency of obeservation in and next to columns
N=table(Smpl_who[,Variable[i]])
# y-coordinate set to the highest frequency
text(x=rep(max(N)-20, 5), y=c(0.75, 1.95, 3.15, 4.35, 5.55),
         labels=N, cex=1, col="black") }
```



Empty plots

To draw a plot, the plotting window has to be opened somehow. There are several ways to do it, one being to call *plot.new()* the other to draw an empty plot.

The advantage of drawing an empty plot is, that some settings can already be specified, even if they are not visible.

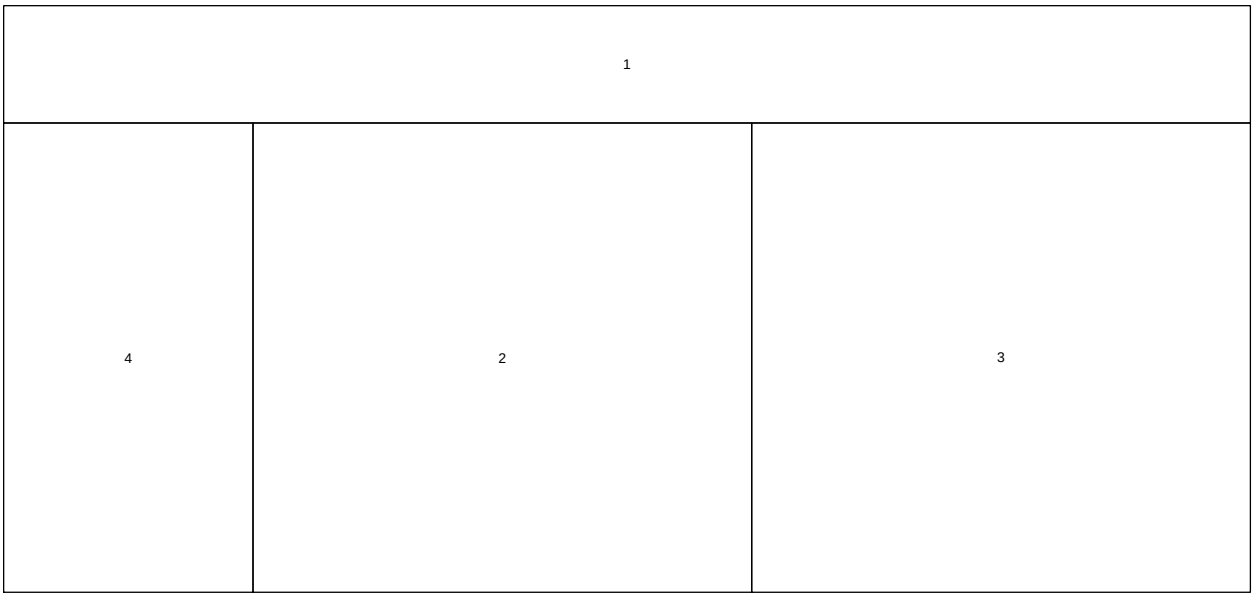
```
# The 'drawing an empty plot' approach

plot(1,1, col="white", # White point on white background
          xlim=c(15,100), ylim=c(0,80), # Specify required axes ranges
          bty="n", # No frame
          yaxt="n",xaxt="n",   # No axes
          xlab="", ylab="")   # No default labels


# Exercise

# Draw only a frame
# Draw only an x-axis
```

Drawing a plot

It can happen that one has a concrete idea what to show and the entire plot needs to be drawn from scratch. A few useful functions in this regard are *segments()*, *abline*, *points()*, *rect()*, *polygon()*.

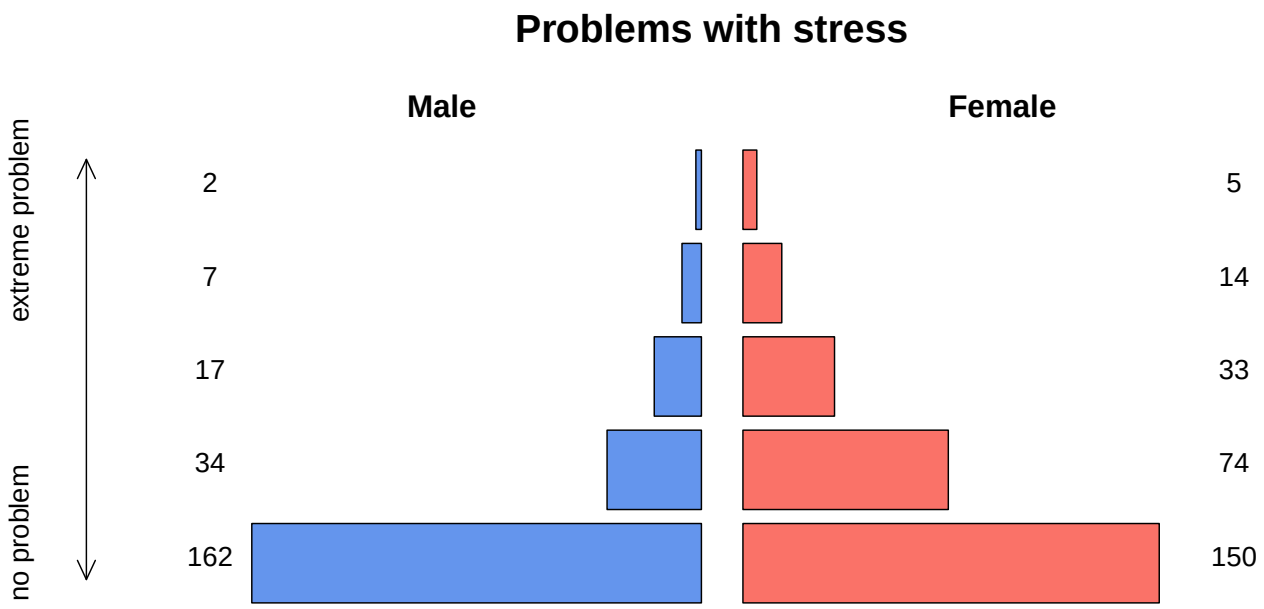
Example of a mirrored histogram for stress by gender using *rect()* and *text()*.

```
# Define layout matrix
par(oma=c(0.25,0.25,0.25,0.25))
mat = matrix(c(1,1,1,1,1,
               4,2,2,3,3),
               byrow=TRUE, nrow=2)
layout(mat, height = c(0.2, 0.8)) # Set the layout including a height option
layout.show(n=4) # Plot the layout
```




```
par(mar=c(0,0,0,0))
# The 'drawing an empty plot' approach
plot(1,1, col="white",
           xlim=c(0,10), ylim=c(0,1), # Random size (only ratio imports)
           bty="n", # No frame
           yaxt="n",xaxt="n", # No axes
           xlab="", ylab="") # No default labels
text(6,.5, "Problems with stress", # Some title for the example
        font=2,
        col="black",
        cex=2.5)


# Computing the frequencies per gender
Freq=table(Smpl_who[,c("stress", "sex")])
Freq_male=Freq[,2]
Freq_female=Freq[,1]

# Histogram for male subgroup
```

```
par(mar=c(0,0,1.25,0)) # Margin size adjustment

# Starting with an empty but sized plotting field
 plot(1,1, col="white",
           xlim=c(max(Freq)+25,0), ylim=c(0,nrow(Freq)), # Give it the size needed
           main="Male", # Title instead of legend,
           cex.main = 2,
           bty="n", # No frame
           yaxt="n",xaxt="n", # No axes
           xlab="", ylab="") # No default labels

 # Drawing rectangles of the size of the frequencies for the male subgroup
for(i in 1:length(Freq_male)){
rect(xleft=0, ybottom=i-1, xright=Freq_male[i], ytop=i-0.15,
     col="cornflowerblue", border=NULL)
text(x=max(Freq)+15, y=i-0.5, labels=Freq_male[i], cex=1.8)
}


# Histogram for female subgroup
par(mar=c(0,0,1.25,0)) #margin size adjustment

# A second empty but sized plotting field
  plot(1,1, col="white",
           xlim=c(0, max(Freq)+25), ylim=c(0,nrow(Freq)), # Give it the size needed
           main="Female", # Title instead of legend,
           cex.main = 2,
           bty="n", # No frame
           yaxt="n",xaxt="n", # No axes
           xlab="", ylab="") # No default labels

# Drawing rectangles of the size of the frequencies for the female subgroup
for(j in 1:length(Freq_female)){
rect(xleft=0, ybottom=j-1, xright=Freq_female[j], ytop=j-0.15,
     col=Coral, border=NULL)
text(x=max(Freq)+15, y=j-0.5, labels=Freq_female[j], cex=1.8)
}

  # Empty plot
   plot(1,1, col="white",
           xlim=c(0, 4), ylim=c(0,10), # Give it the size needed
           bty="n", # No frame
           yaxt="n",xaxt="n", # No axes
           xlab="", ylab="") # No default labels

   # Putting an arrow
   arrows(x0=3, y0=0.5, x1=3, y1=9.5,
                   length=0.15, # Size of arrow head
                   angle=25, # Angle 'from shaft to edge' of the arrow
                   code=3) # Type of arrow

   # Add some text left to the arrow
   text(2,1.5, "no problem", cex=1.8, srt=90) # srt=string rotation in degree
   text(2,8.2, "extreme problem", cex=1.8, srt=90)
```


Problems with stress
Male
Female
extreme problem
no problem
2
7
17
34
162
5
14
33
74
150

ggplot

The R-package ggplot2 offers functionalities to create figures with a professional layout. Elements from ggplot are "woven" into a set of layers and many options are set to reasonable defaults.

Every ggplot2 plot has three key components:

- a data set
- an aesthetic mapping between variables in the data set and visual properties
- a layer to describe how to render each observation (typically with a geom function)

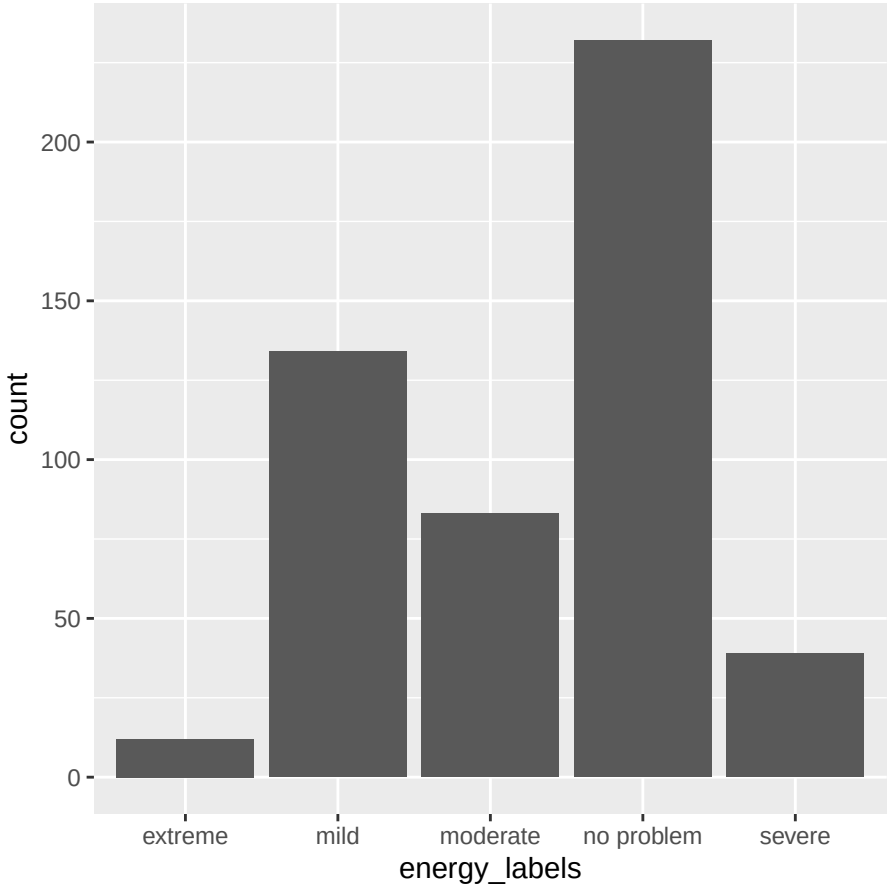
Barcharts with ggplot2

First, load the ggplot2 package and calculate an aggregated data set of the frequency per energy level as the data set to use. Note, that at this point the actual plot geometry (e.g. bar chart or histogram) is still missing.

```
library(ggplot2)
library(dplyr)
# Calculate average health problems score per level of getting_out
data <- Smpl_who %>%
                    group_by(energy) %>%
                    summarize(count = n())

# Recode numbers to text labels
data$energy_labels <- recode(data$energy, "1" = "no problem",
                                          "2" = "mild", "
                                           3" = "moderate",
                                          "4" = "severe",
                                          "5" = "extreme")

# Define an ordered factor for energy labels
data$category <- factor(data$energy_labels, ordered = TRUE,
                  levels = c("no problem", "mild", "moderate",
                             "severe", "extreme"))

# Define a color scale from green to red and a mapping to
#   the levels of the energy variable
green_red <- rev(brewer.pal(5, "RdYlGn"))
names(green_red) <- c("no problem", "mild", "moderate", "severe", "extreme")

# Store basic plot in variable p
p <- ggplot(data) + # Data set
                    aes(x=energy_labels, y = count)  # Base aesthetics
```

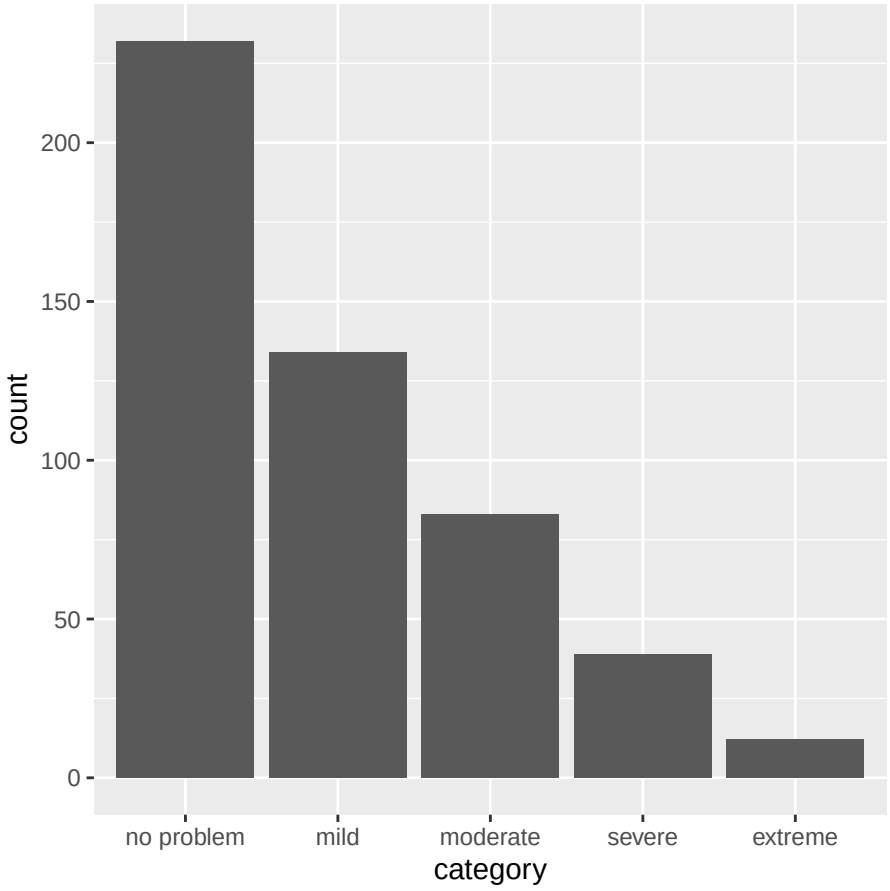
Add a bar chart geometry. This will plot a basic bar chart for the energy variable. Note, however, that the bars are ordered alphabetically.

```
p2 = p +
  # stat = "identity" tells ggplot to plot the formerly calculated
  #   frequency values "as is", i.e. no further aggregation needs
  #   to be performed
  geom_bar(stat = "identity") # Standard bar chart
p2
```



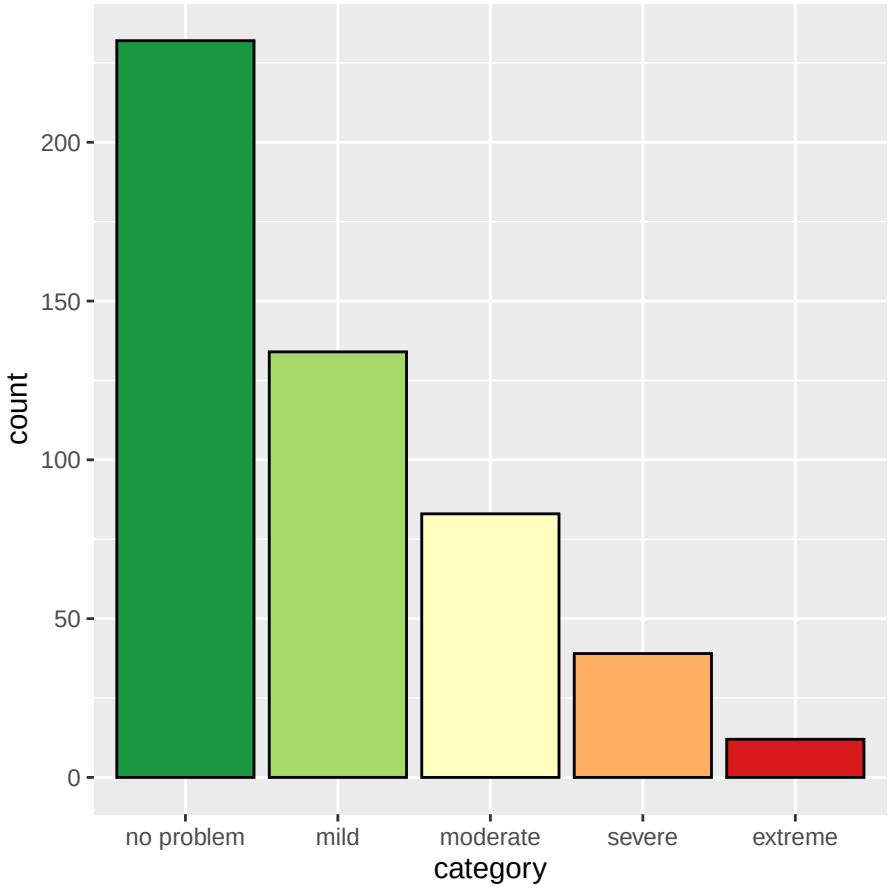
Reorder the bars from “no problem” to “extreme problem”.

```
p3 = ggplot(data) + # Data
       aes(x=reorder(energy_labels, energy), y = count) + # Reorder
      geom_bar(stat = "identity") # Standard barchart
# p3

# Alternatively, an ordered factor can be used
p3 = ggplot(data) + # Data
       aes(x=category, y = count) + # Reorder
      geom_bar(stat = "identity") # Standard barchart
p3
```



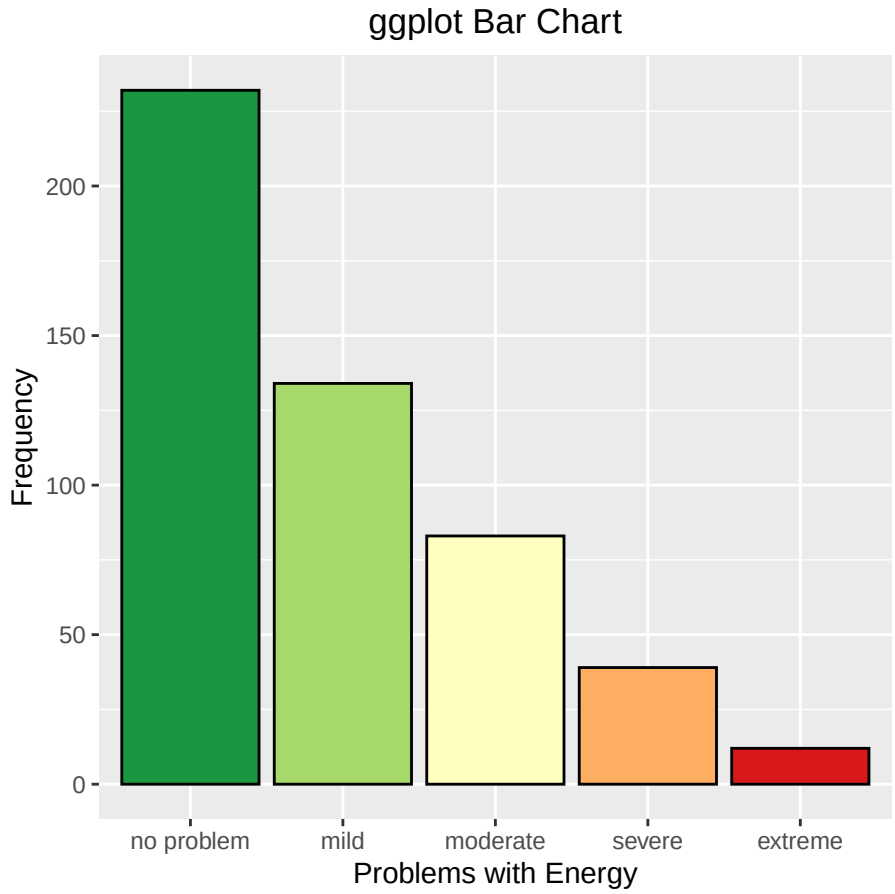
Modify the bar colors to represent the problem severity.

```
library(RColorBrewer)
p4 = p3 +
      # Modify fill color and add black box around each bar
      geom_bar(stat = "identity", fill=green_red, color="black")
p4
```



Modify axes labels, add a main title and center the main title.

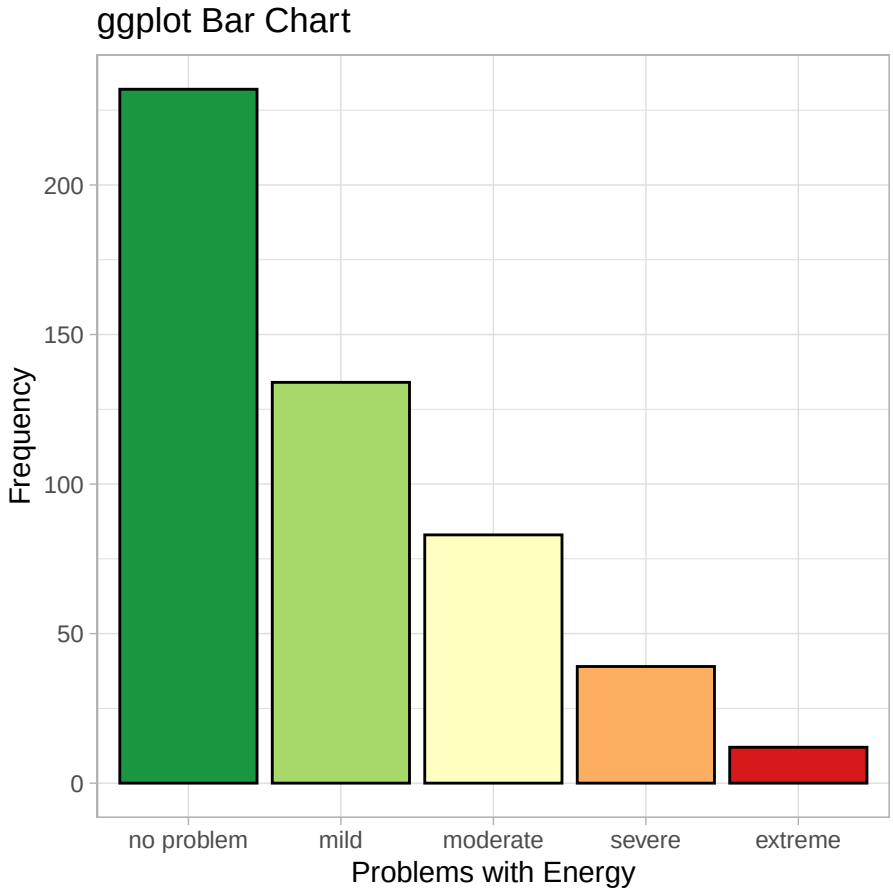
```
p5 = p4 +
        xlab("Problems with Energy") + # Modify x-axis label
        ylab("Frequency") + # Modify y-axis label
        ggtitle("ggplot Bar Chart") + # Add title
        theme(plot.title = element_text(hjust = 0.5)) # Center the title
p5
```



Many details of ggplot formatting (such as centering of the title above) are controlled through themes. A wide variety of different themes is available: https://ggplot2.tidyverse.org/reference/ggtheme.html

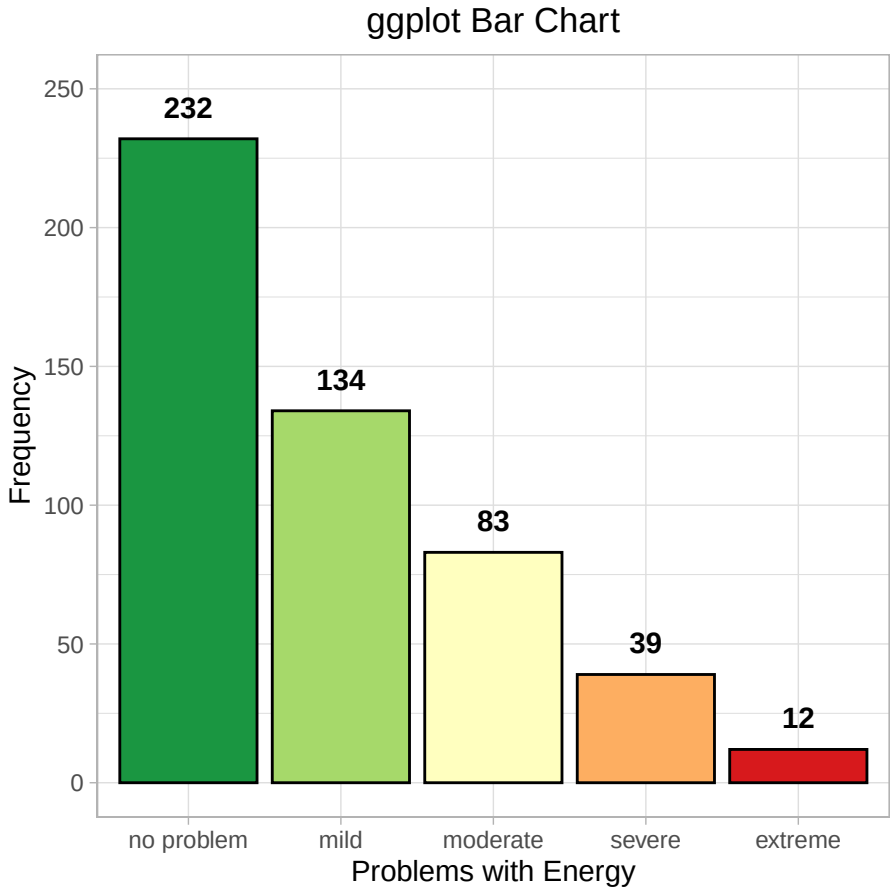
Note, that the main title is no longer centered, as this theme overwrites the earlier theme-based modification.

```
p6 = p5 +
        theme_light() # Theme with light background grid
p6
```



Next, add data values to the plot and recenter the main title. Also, extend the y-axis range, to ensure that the data values fit above each bar.

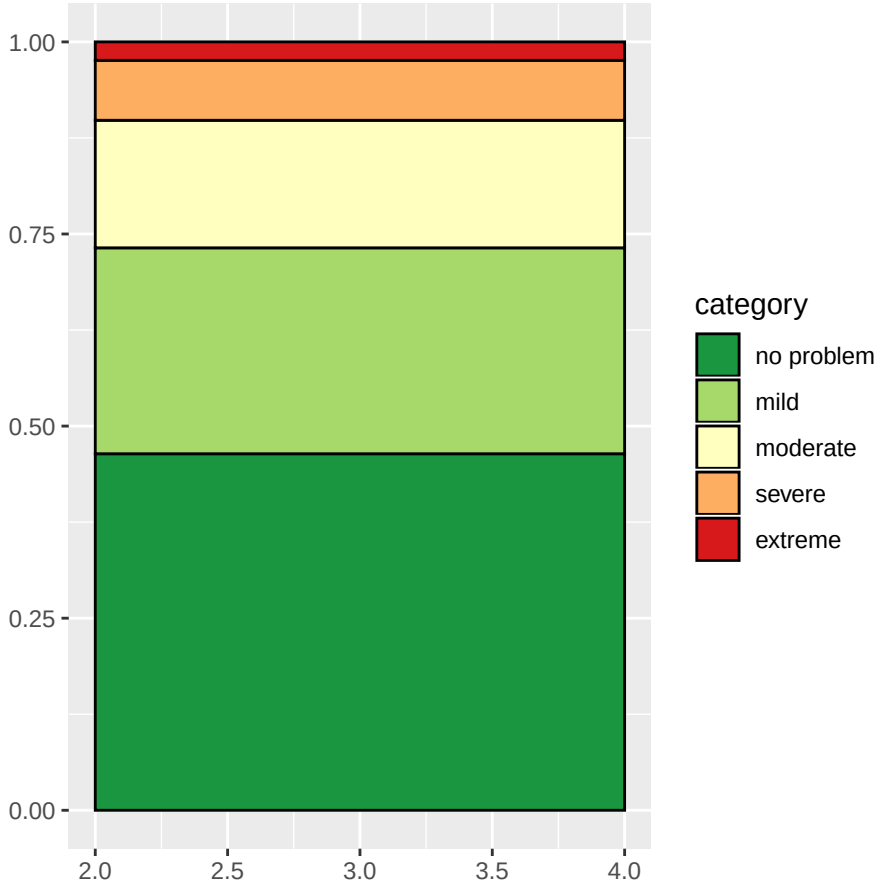
```
p7 = p6 +
      geom_text(aes(label=count), # Values to plot
                vjust=-1, # Position
                color="black", size=4, # Color and size
                fontface="bold") + # Bold font
      ylim(0,250) + # Extend y-axis to fit the data label
      theme(plot.title = element_text(hjust = 0.5)) # Center the title
p7
```



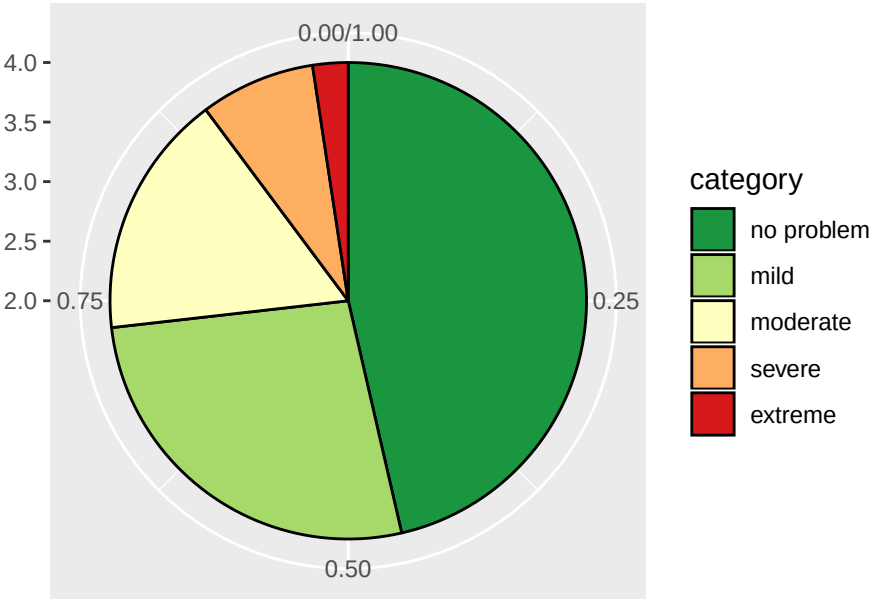
Pie charts with ggplot2

There is no off-the-shelf template for a donut chart / pie chart in ggplot2. However, they can be calculated by a transformation of the coordinate system (basically by bending a single stacked barchart into a pie / donut shape).

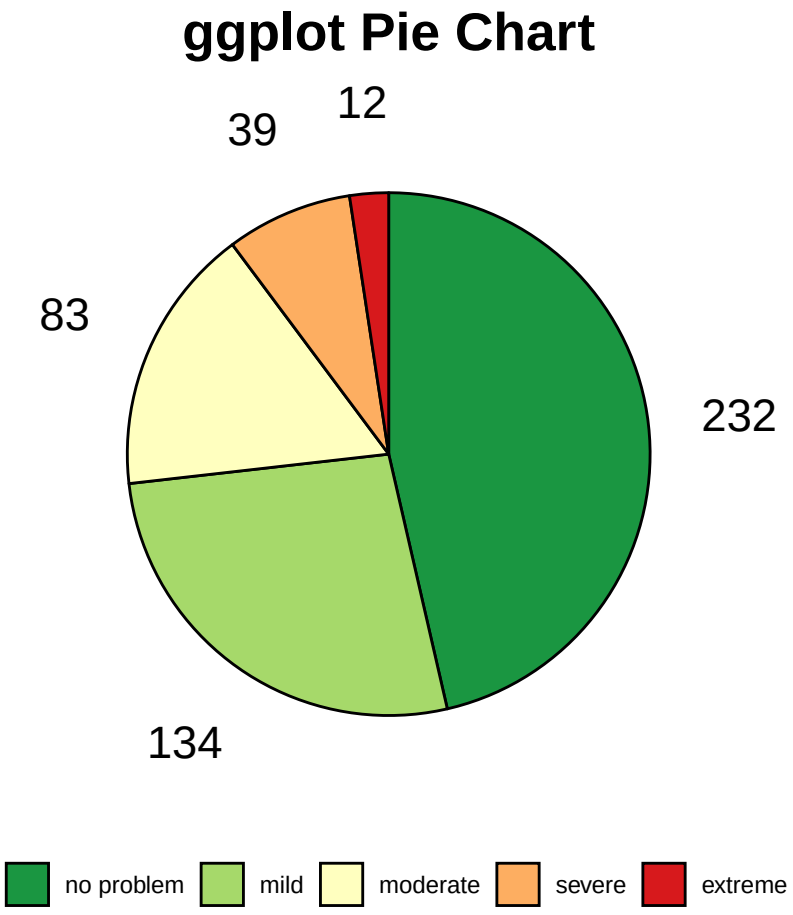
```
# Compute percentages
data$fraction <- data$count / sum(data$count)

# Compute the cumulative percentages (top of each rectangle)
data$ymax <- cumsum(data$fraction)

# Compute the bottom of each rectangle
data$ymin <- c(0, head(data$ymax, n=-1))

# Compute label position
data$labelPosition <- (data$ymax + data$ymin) / 2

# The above transformations allow to draw a stacked bar chart with
#   height from 0 to 1 and width from 2 to 4
stacked_bar <- ggplot(data, aes(ymax=ymax, ymin=ymin, xmax=4, xmin=2, fill=category)) +
  geom_rect(color="black") +
  scale_colour_manual(values = green_red, aesthetics = "fill") # Color scheme
stacked_bar
```



```
# The axis around the circle ranges from 0 to 1 (clockwise)
#   The axis from the center to the edge of the pie chart goes from 2 to 4
#   Note, that this corresponds to the stacked barchart dimensions defined above
pie <- stacked_bar + # Start from stacked barchart
  # Define position and size of the text
  coord_polar(theta="y")  # Transformation into pie
pie
```



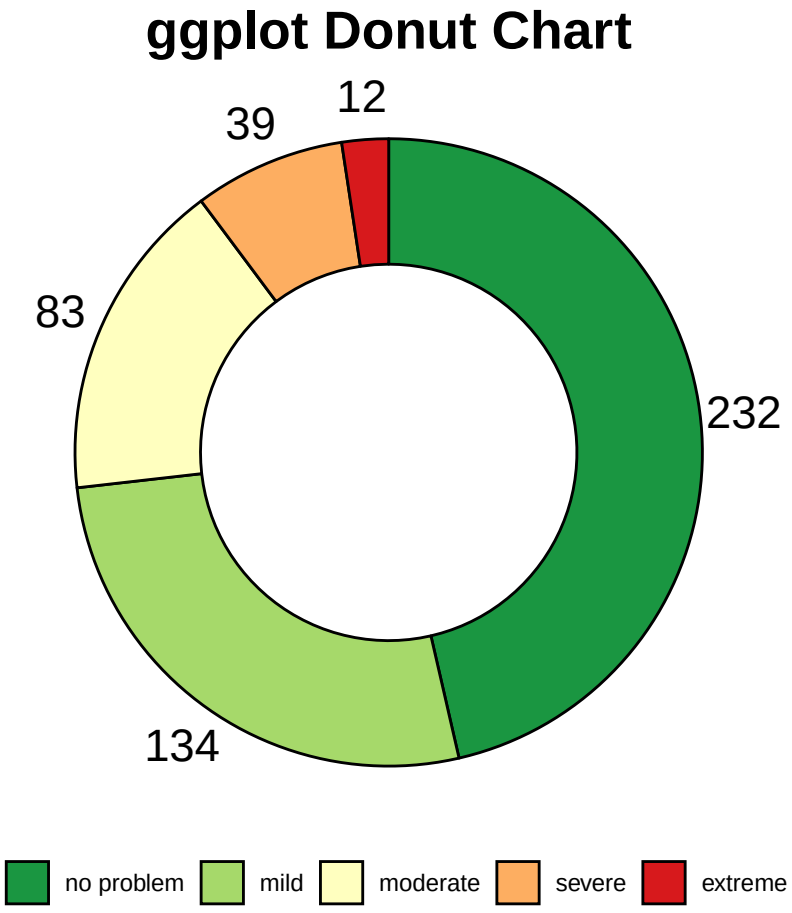
```r
# Add data values around the chart and format the legend
pie2 <- pie +
  xlim(c(2, 4.4)) + # Slightly extend x-axis to have space for data values
  # 4.7 is outside of the pie's position.
  geom_text( x=4.7, aes(y=labelPosition, label=count), size=6) +
  theme_void() + # No background
  theme(legend.position = "bottom") + # Legend at bottom
  labs(fill="") + # Legend without title
  ggtitle("ggplot Pie Chart") +  # Add title
  # Format title
  theme(plot.title = element_text(hjust = 0.5, size = 20, face="bold"))
pie2
```

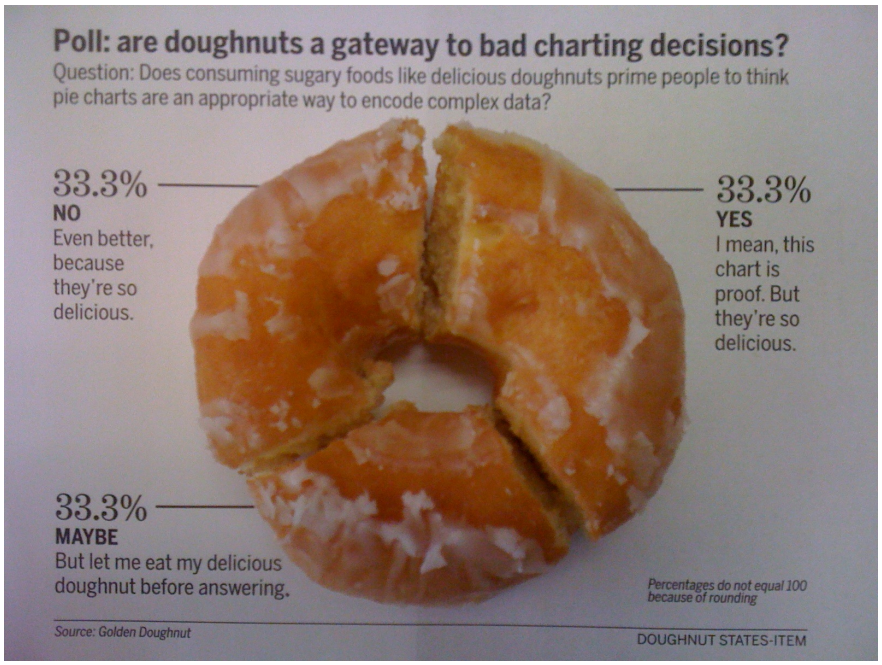


Donut charts with ggplot2

The donut chart can be derived from the pie chart by cutting a hole into the middle.

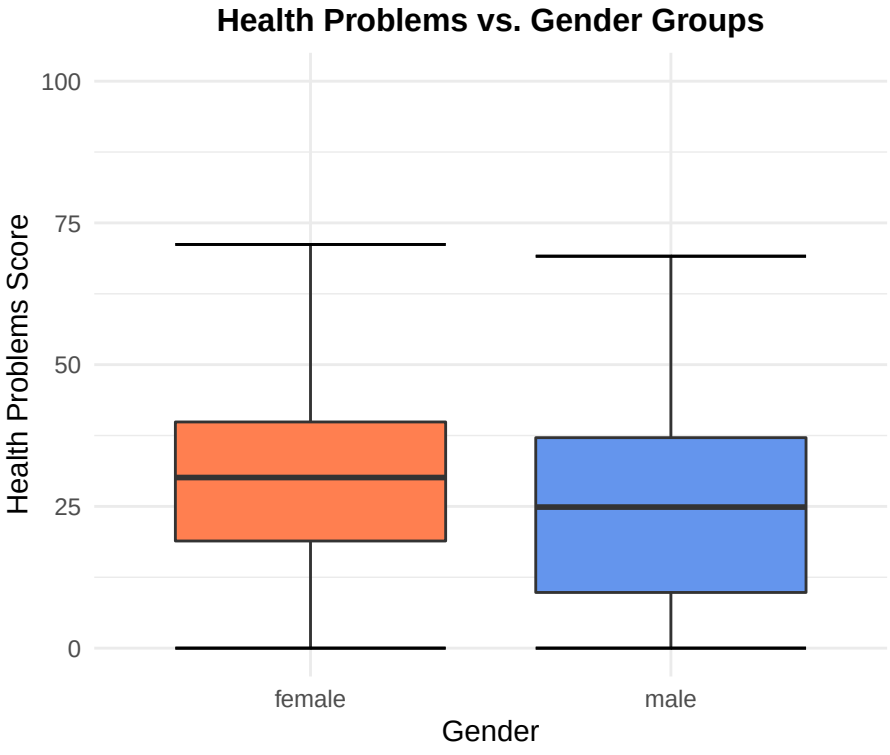
```
# Basic aesthetics
# Donut chart is basically the same plot, just with a hole in the middle,
# Cut the hole into the donut chart by extending the x-axis
                # from 2 (center of the pie) to -1, hence "moving" the
                # pie along the axis and adding some free space into the middle
donut <- pie2 +
  xlim(c(-1, 4)) + # -1 is the new center of the plot, 4 remains the outer edge
  ggtitle("ggplot Donut Chart")  # Add title
donut
```




Poll: are doughnuts a gateway to bad charting decisions?
Question: Does consuming sugary foods like delicious doughnuts prime people to think pie charts are an appropriate way to encode complex data?
33.3%
NO
Even better, because they're so delicious.
33.3%
YES
I mean, this chart is proof. But they're so delicious.
33.3%
MAYBE
But let me eat my delicious doughnut before answering.
Percentages do not equal 100 because of rounding
Source: Golden Doughnut
DOUGHNUT STATES-ITEM

```
# Exercise
# In this exercise you will plot a boxplot of the health problems score
#   across the two gender groups
# Refer to the reference https://ggplot2.tidyverse.org/reference and
#         https://ggplot2.tidyverse.org/reference/geom_boxplot.html for details

# Build the graph step by step using ggplot
# First, tell R which dataset to use (Smpl_who), establish the basic aesthetics (aes) of
#        what to draw on the x and y axis and which variable to use to color the
#        two boxplots, and define the geometry (geom_boxplot)

# Next, color (use fill-argument) females in "Coral" and males in "cornflower" blue
#     tip: define a help vector analogue to green_red called coral_blue and
#          assign names() for the vector to establish the mapping to the gender groups
#     tip: use scale_color_manual with the "fill" aesthetics property

# Next, assign nicer x and y axis labels

# Next, change the plot template to theme_minimal

# Next, add a main title and center it

# Remove the plot's legend
# Tip: this is a theme option

# Final task (...tricky): Add a horizontal line to the end of your whiskers
#   as is often the case in standard boxplots
#   Tip: Modify stat_boplot() and add to plot
```

The resulting plot should look similar to the one below



A few more plots

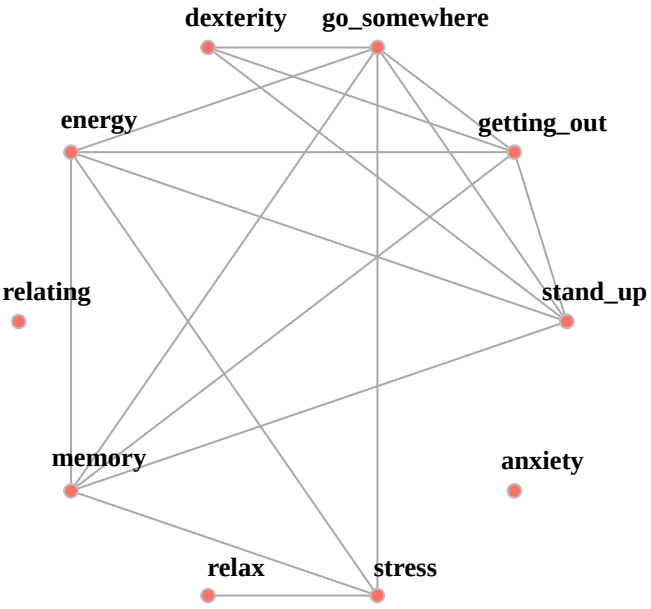
There is an ever increasing number of useful and performant plot functions that can be found in the R. Many nice examples are found on: https://www.r-graph-gallery.com/. What follows provides pieces of syntax or some information on how to do multidimensional plots, association plots as well as some 'fancy' functions: *mosaicplot()*, *wordcloud()*, *VennDiagram()*, etc... Some of these plots may be more difficult to include in scientific reports and publications, but can leave an 'impression' in presentations or meetings.

Association plots

Association plots are a graphical way to display associations or distances between variables. Using some of the variables from Smpl_who an association plot illustrating the strength of their correlation is drawn in what follows.

```
library(igraph) # A package for network plots

# Putting the numbers as numeric and calculating the rank correlation
Smpl_who[,c("stand_up", "getting_out", "go_somewhere",
            "dexterity", "energy", "relating",
            "memory", "relax", "stress", "anxiety")]=
  apply(Smpl_who[,c("stand_up", "getting_out",
                    "go_somewhere", "dexterity",
                    "energy", "relating",
                    "memory", "relax", "stress",
                    "anxiety")],2,as.numeric)

Correl=cor(Smpl_who[,c("stand_up", "getting_out", "go_somewhere",
                       "dexterity", "energy", "relating",
                       "memory", "relax", "stress", "anxiety")],
               method="spearman",
               use="pairwise.complete.obs")

# Keep positive correlations
Correl[Correl<0.3] = 0  # This can be varied to keep only higher positive correlations

# Make an Igraph object from this matrix:
network = graph_from_adjacency_matrix(Correl,
                                      weighted=TRUE,
                                      mode="undirected",
                                      diag=FALSE)

# Layout the spacial arrangement of vertices based on
#  the circle-algorithm
coords <- layout_in_circle(network)

# Basic chart, see ?igraph.plotting for all available options
plot.igraph(network, # Data to plot
            layout=coords, # Coordinates for vertex arrangement
            vertex.color=Coral, # Vertex coloring
            vertex.size=5, # Reduce point size
            vertex.frame.color="gray",  # Edge color
            vertex.label.color="black", # Vertex label color
            vertex.label.cex=0.8, # Vertex label size
            vertex.label.dist=2, # Distance of label from vertex
```

```
vertex.label.font=2) # Increase label font size
```



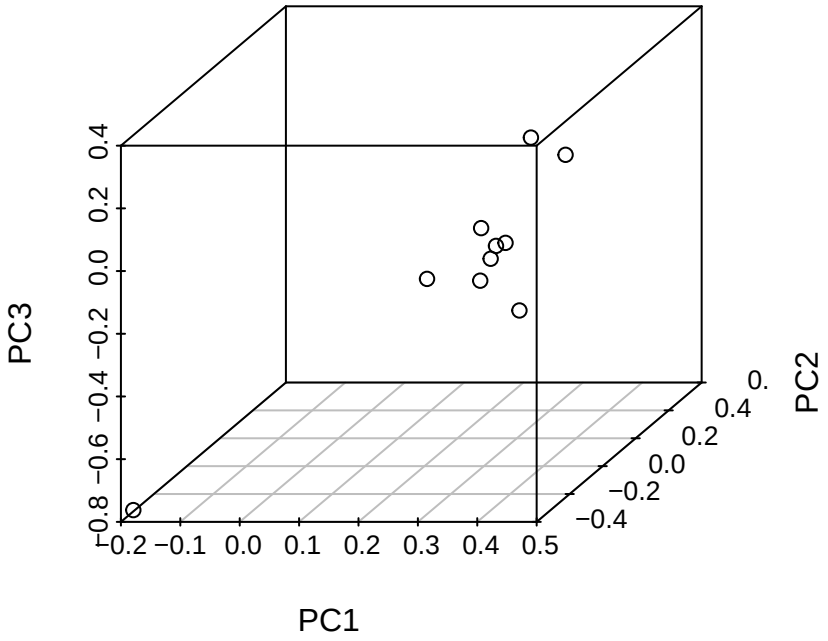

```
# Exercise
# Calculate coords using layout_nicely and redraw the plot
# Change the color of go_somewhere to "cornflowerblue"
# Change the graph to slightly curved edges (e.g. edge.curved=0.2)
```

Three-dimensional plots

Typically, 3d-plots are not very suitable to be displayed in a 2-dimensional space such as reports, articles or presentations. The perception and derived meaning of a 3-dimensional content may vary a lot depending on the perspective in which it is shown and thus may cause suspicion in a print-format.

However, R provides functions to show 3-dimensional plots on plotting windows where the user can rotate the view, zoom in and zoom out easily. This can be helpful when analysing distances or association matrices in a xyz coordinate system.

```
library(scatterplot3d)

Complete_Cases=complete.cases(Smpl_who[,c("stand_up", "getting_out",
            "go_somewhere", "dexterity", "energy", "relating", "memory",
            "relax", "stress", "anxiety")])

Smpl_who_CC=Smpl_who[Complete_Cases,c("stand_up", "getting_out",
                "go_somewhere", "dexterity", "energy", "relating",
                "memory", "relax", "stress", "anxiety")]


Principal_Components=prcomp(Smpl_who_CC, scale=TRUE)$rotation

scatterplot3d(x = Principal_Components[,1],
              y = Principal_Components[,2],
              z = Principal_Components[,3],
              xlab="PC1", ylab="PC2", zlab="PC3")
```


PC3
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−0.2 −0.1 0.0 0.1 0.2 0.3 0.4 0.5
PC1
0.
0.4
0.2
0.0
−0.2
−0.4
PC2

GUI for 3D-plots

```
# On Mac: requires installation of XQuartz from xquartz.org
#         need to restart computer after installation
library(rgl)

rgl::plot3d(Principal_Components[,1],
            Principal_Components[,2],
            Principal_Components[,3], type="n",
            xlab="PC1", ylab="PC2", zlab="PC3",
            axes=TRUE,
            box=FALSE)
rgl::text3d(Principal_Components[,1],
            Principal_Components[,2],
            Principal_Components[,3],
            text=rownames(Principal_Components),
            cex=0.7)
```

Histoire à considérer sous plusieurs angles (Engl.: *History to be considered from several angles*)[1]

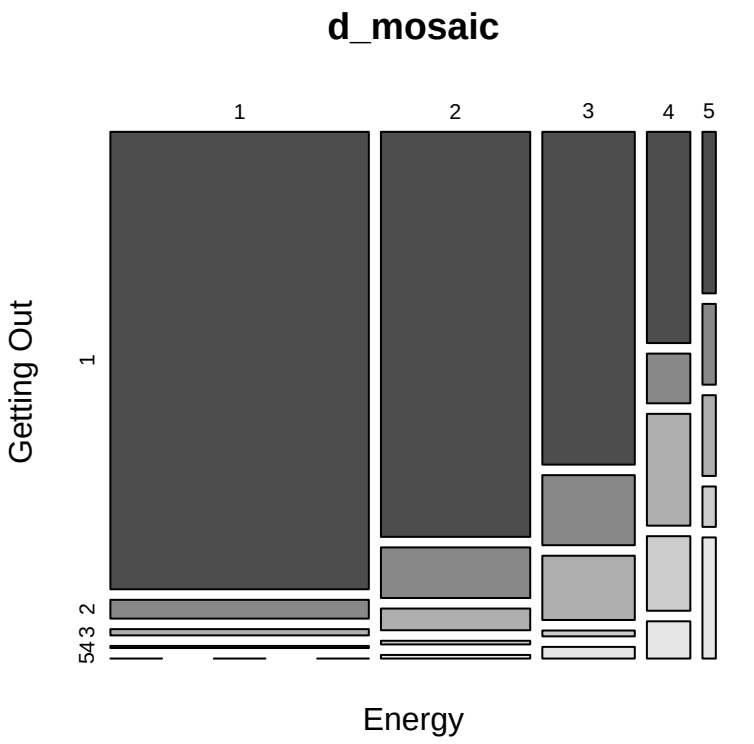
[1] Image from Gotlib - Rubrique-à-Brac – Tome 1 à Tome 5 : L'intégrale.

Mosaic plots

The mosaic plot, also known as Marimekko plot, is a graphical method to visualize data from qualitative or ordered variables. The mosaic plot allows to constrast two variables or more. However, with more than 2 variables the plot may become difficult to read. The mosaic plot shows the frequency of observations within combination of variable levels by mean of areas of mosaic tiles.

The below code produces a mosaic plot of energy vs. getting_out.

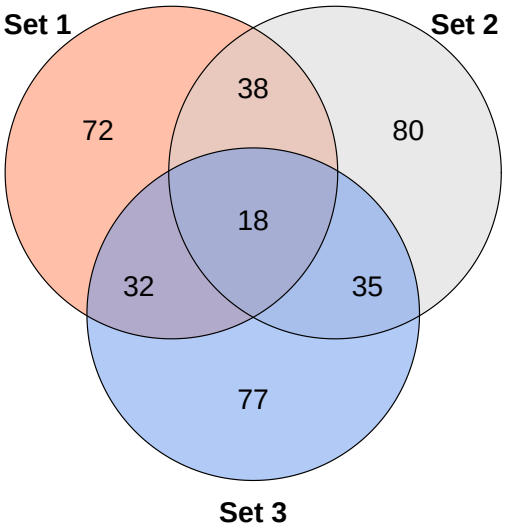
```
d_mosaic <- table(Smpl_who$energy, Smpl_who$getting_out)
mosaicplot(d_mosaic, xlab = "Energy", ylab = "Getting Out", color = TRUE)
```



Venn diagrams

A Venn diagram can be used to visualize overlap between different sets. The elements within each set are compared against the other sets. If an element is present in more than one set, this is visualized as the intersection (overlap) of these sets.

```
set.seed(1234)

# Generate 3 sets of 200 numbers that randomly overlap some elements
set1 <- sample(1:500, size = 200, replace = TRUE)
set2 <- sample(1:500, size = 200, replace = TRUE)
set3 <- sample(1:500, size = 200, replace = TRUE)

# install.packages("VennDiagram")
library(VennDiagram)

# Chart
venn.plot=venn.diagram(
        x = list(set1, set2, set3),
        category.names = c("Set 1" , "Set 2 " , "Set 3"),
        filename = NULL,
        # Circles
        lwd = 1, # Reduce line width
        fill =  c("Coral", "lightgray", "cornflowerblue"),

        # Numbers
        cex = 1.5, # Reduce size of numbers
        fontfamily = "sans",  # Font without serifs

        # Set names
        cat.cex = 1.5, # Reduce size of labels
        cat.fontface = "bold", # Put labels in bold
        cat.dist = c(0.05, 0.05, 0.05), # Distance of set labels
        cat.fontfamily = "sans") # Font without serifs

grid.draw(venn.plot)
```



Word clouds

Using the wordcloud package, the following code produces a word cloud using some randomly selected color names and attributing them random sizes - and putting them in the respective color. Random selection of a number of vector elements can be done with the function *sample()*.

```
library(wordcloud)

set.seed(123) # Make the results reproducible
# Selecting 200 colors from the color name list with the function sample()
n <- 200
# Use rbeta to create frequencies s.t. large weights become increasingly unlikely
# This will allow to plot a word cloud with some larger words and many smaller words
# This resambles proportions in a typical text analysis word cloud
colors_cloud = data.frame(word = sample(colors(), n), freq = rbeta(n, .2, 5))

# Order from largest to smallest frequency
colors_cloud = colors_cloud[order(colors_cloud$freq, decreasing=T),]
# Keep only frequencies larger than .09 to avoid too extreme size differences
colors_cloud = subset(colors_cloud, freq >= .09)
# Multiply by 1000 to turn frequencies (.05 - 1) into whole numbers (5-1000)
colors_cloud$freq <- round(colors_cloud$freq,3)*1000
# Assign row names
rownames(colors_cloud) = colors_cloud$word

# Draw the word Cloud
wordcloud(words = colors_cloud$word, # Words to draw
          freq = colors_cloud$freq, # Sizing
          colors=(rownames(colors_cloud)), ordered.colors = TRUE) # Coloring
```

ggBubble

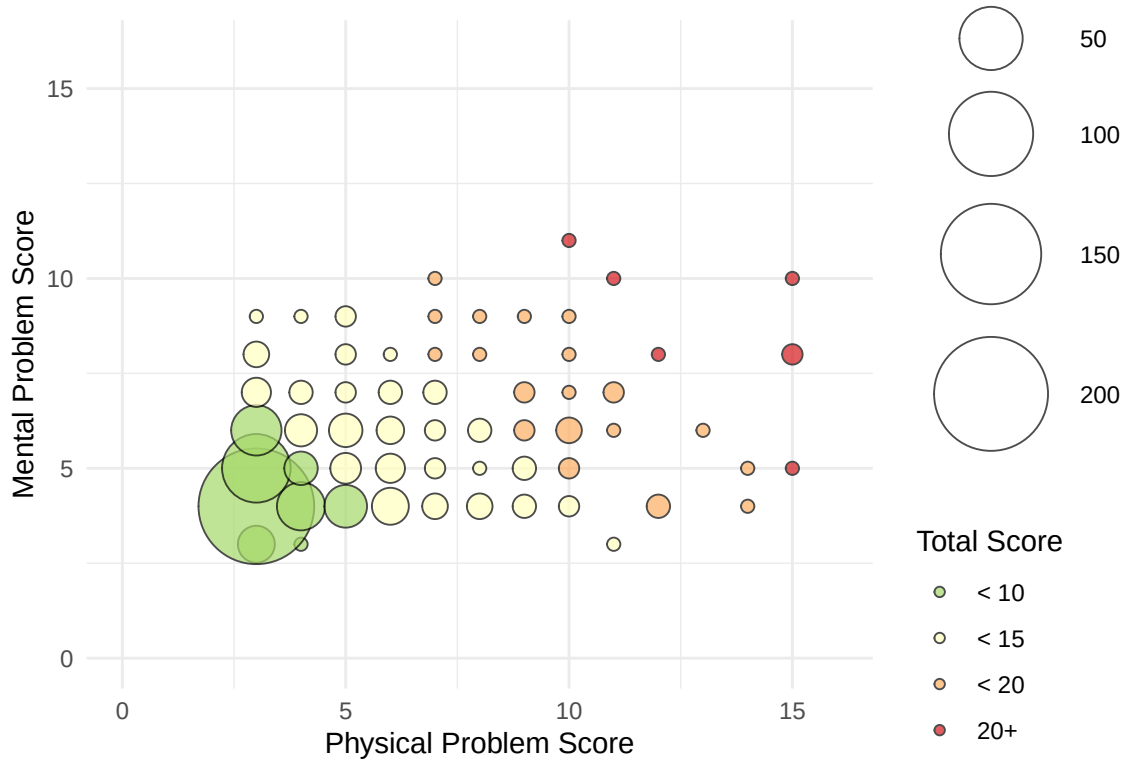
Common bubble chart

The following code calculates two subscales for physical and mental problems. For these two scales, the number of study participants for every combination is counted. Larger bubbles indicate more persons for a given combination of the two scales.

In addition, the bubbles are colored from low to high total problem scores, where the total problem score is the sum of the two sub scales.

```
library(ggplot2)
library(dplyr)

# Calculate the two sub scales
Who_Scales <- data.frame(mental = Smpl_who$stress + Smpl_who$memory + Smpl_who$anxiety,
                         physical = Smpl_who$stand_up + Smpl_who$getting_out +
                                    Smpl_who$go_somewhere
                         )

# Calculate total problems score
Who_Scales$total_problems = Who_Scales$mental + Who_Scales$physical

# Count the number of observations per combination of the two scales
Who_Scales <- Who_Scales %>%
      group_by(mental, physical, total_problems) %>%
      summarize(count_obs = n())

green_red <- rev(brewer.pal(5, "RdYlGn"))

# Logic for bubble coloring
Who_Scales$total_problems_col = ifelse(Who_Scales$total_problems<5, green_red[1],
                                ifelse(Who_Scales$total_problems<10, green_red[2],
                                ifelse(Who_Scales$total_problems<15, green_red[3],
                                ifelse(Who_Scales$total_problems<20, green_red[4],
                                                                     green_red[5]))))

# Assign the colors also to a vector, so that every element can be named
green_red2 <- green_red
names(green_red2) <- c("<  5", "< 10", "< 15", "< 20", "20+")

# Calculate the re-coded total problems variable for labelling the plot
Who_Scales$total_problems_lab = ifelse(Who_Scales$total_problems<5,  "<  5",
                                ifelse(Who_Scales$total_problems<10, "< 10",
                                ifelse(Who_Scales$total_problems<15, "< 15",
                                ifelse(Who_Scales$total_problems<20, "< 20",
                                                                     "20+"))))

# Remove 4 rows with missing Values
Who_Scales <- Who_Scales[rowSums(is.na(Who_Scales))==0,]

# Call ggplot and define the basic elements to plot
  ggplot(Who_Scales, aes(x=physical, y=mental, size = count_obs)) +
    # Draw the bubbles
    geom_point(alpha=0.7, # Transparency
               pch=21, # Plotting character: circle with border
```

```
          color="black", # Black color for the border
          # Color the bubbles based on the values of the total score
          aes(fill=total_problems_lab)) +
labs(fill="Total Score") + # Label for coloring legend
scale_size(range = c(2, 20), # Scaling of bubble sizes
          name="Nbr. Obs.") + # Title for bubble size legend
xlim(0,16) + ylim(0,16) + # Axes ranges
xlab("Physical Problem Score") + # Axis label
ylab("Mental Problem Score") + # Axis label
# Match the value label (e.g. <10) to its respective color
scale_colour_manual(values = green_red2,  # Color scale
                    aesthetics = "fill") + # Fill color
theme_minimal()   # Minimal background layout
```

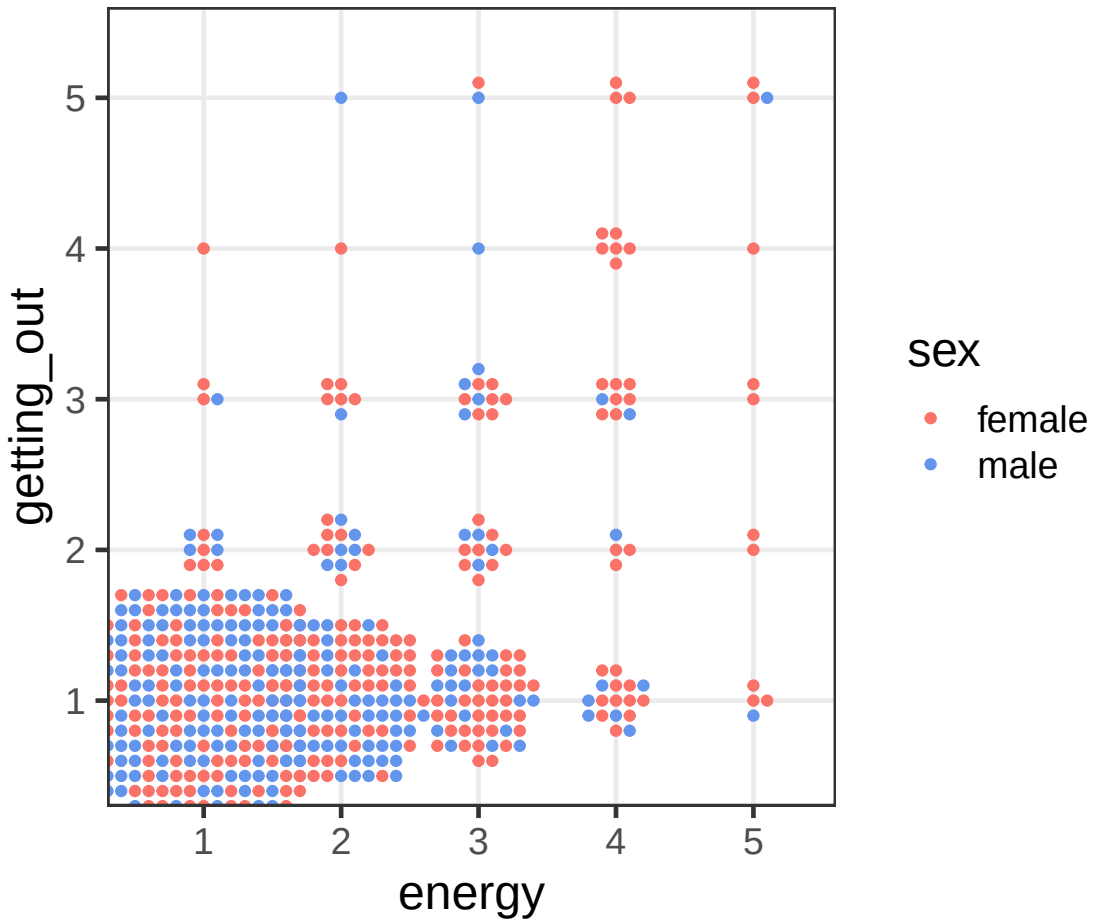


Jittered bubble chart

The following code plots energy scores vs. getting_out scores in a jittered bubble chart. Instead of using the frequency to size the bubble, a number of surrounding bubbles are plotted. The higher the frequency of the respective combination, the more bubbles are drawn.

Bubbles are colored based on sex (female=coral, male=cornflowerblue). If one gender group has more observations for a given combination, then more bubbles in that color will be drawn.

```
library(ggBubbles)
ggplot(# Data to plot
       data = Smpl_who,
       aes(x = as.factor(energy),
           y = as.factor(getting_out),
           col = sex)) +
       # Draw multiple points - calculations by position_surround function
       geom_point(position = position_surround())+
       # Colors to use
       scale_colour_manual(values = c(Coral, "cornflowerblue"))+
       # Add grid formatting
       theme_bw(base_size = 18) + # Similar to cex-parameter in base R
       # Nicer axes Labels
       xlab("energy") + ylab("getting_out")
```

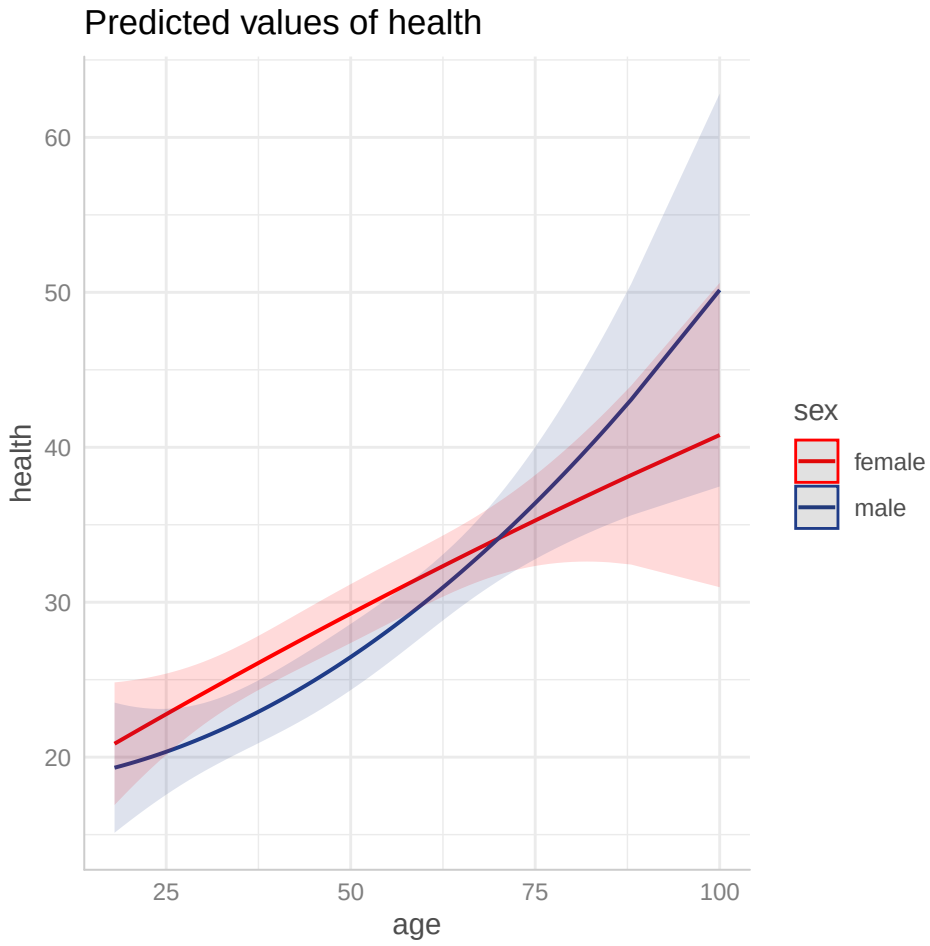


ggpredict

If significant changes in format are required, it can become quite cumbersome to figure out how to achieve this with ggplot alone. From personal experience, building a custom template from scratch using base R functions (and potentially on top of ggplot functions) allows more fine-granular control.

The following example shows how marginal means, calculated with *ggpredict()*, can be used to draw a marginal mean plot in ggplot2. Thereafter, an extended version is presented.

```
library(ggeffects)
library(ggplot2)

# An additive model with interaction effect and polynom
LM_Health=lm(health ~ sex*poly(age,2)+getting_out+energy+relating+stress+anxiety,
             data=Smpl_who, na.action=na.omit)

# Calculating adjusted mean health problems scores for age by gender
Age_sex=ggpredict(LM_Health, terms=c("age [all]",  "sex"))
plot(Age_sex, colors="quadro")
```



```
# Look at the Age_sex output
Age_sex


# Predicted values of health
# x = age

# sex = female

  x | Predicted |   SE |         95% CI
-----------------------------------------
 18 |     20.87 | 2.02 | [16.91, 24.83]
 29 |     23.84 | 1.08 | [21.72, 25.97]
 41 |     26.99 | 0.90 | [25.21, 28.76]
 53 |     30.01 | 0.98 | [28.10, 31.93]
 64 |     32.69 | 1.03 | [30.68, 34.70]
100 |     40.79 | 5.01 | [30.97, 50.61]

# sex = male

  x | Predicted |   SE |         95% CI
-----------------------------------------
 18 |     19.32 | 2.14 | [15.12, 23.52]
 29 |     21.07 | 1.17 | [18.78, 23.36]
 41 |     23.82 | 1.06 | [21.75, 25.89]
 53 |     27.45 | 1.08 | [25.33, 29.57]
 64 |     31.56 | 1.12 | [29.37, 33.74]
100 |     50.15 | 6.47 | [37.47, 62.83]

Adjusted for:
* getting_out = 1.33
*      energy = 1.93
*    relating = 1.20
*      stress = 1.60
*     anxiety = 1.86
```

```
# To save the output, get the names of the object to see how it is stored and how it
#  can be saved
names(Age_sex)
```

```
[1] "x"         "predicted" "std.error" "conf.low"  "conf.high" "group"
```

```
# The values of the ggplot
Graph_values=as.data.frame(cbind(age=Age_sex$x,
                                 sex=Age_sex$group,
                                 health=Age_sex$predicted,
                                 health_se=Age_sex$std.error,
                                 health_ci_low=Age_sex$conf.low,
                                 health_ci_up=Age_sex$conf.high))

# Recode sex here a factor with label as in the Smpl_who data
Graph_values[,"sex"]=factor(Graph_values[,"sex"],
                            levels=c(1,2),
                            labels=c("male", "female"))

# Let's add a column with the information on the number of persons for each value row
```

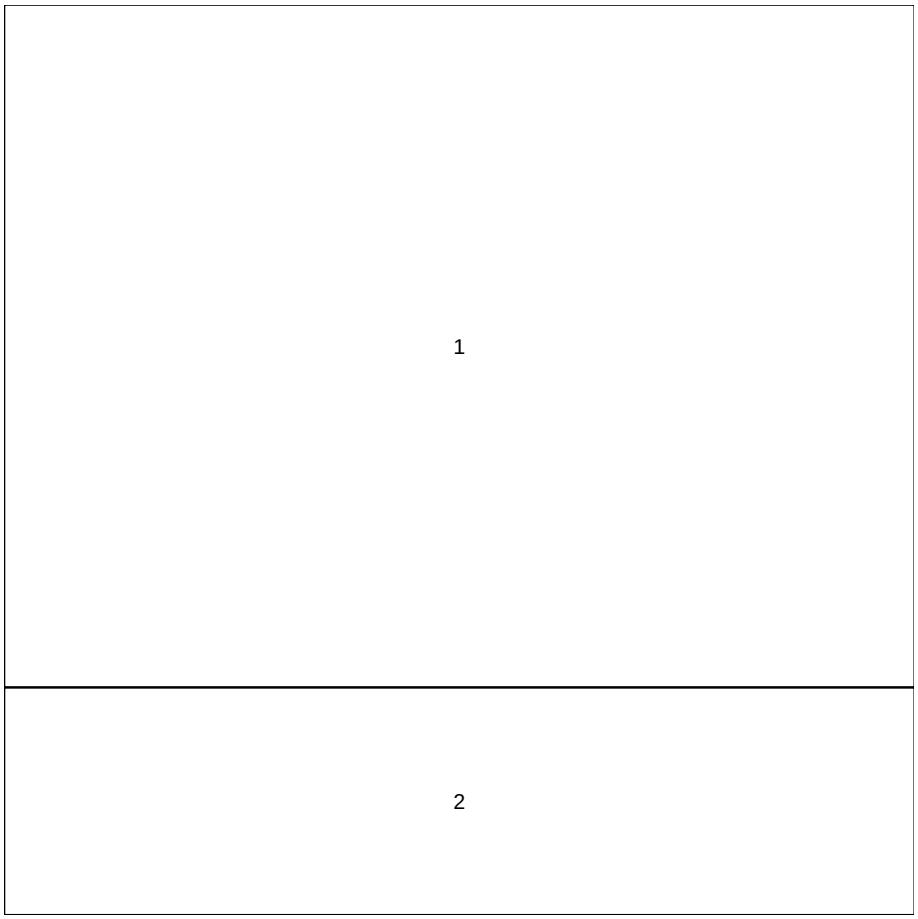
```
for(i in 1:nrow(Graph_values)){
  Graph_values[i,"N"]=nrow(Smpl_who[which(Smpl_who[,"age"]==Graph_values[i,"age"]
                                          &
                                          Smpl_who[,"sex"]==Graph_values[i,"sex"]),])
}

# Exercise

# Check if the sum of N values is equal to the sample size of Smpl_who (tip:
#   compare sum over Graph_values$N vs row number of Smpl_who)
# Check if total number of males and females in Graph_values is the same
#   as in the data Smpl_who
```

ggplot-Faking with base R

The gg-plot issued through *ggpredict()* looks fabulous. However, we would like to add a serie of features which are more readily set up using base R. ggplot2 is to some extent a universe on its own with a wide variety of plotting elements. Modifications, however, such as freely adding, removing or modifying individual elements, can be challenging, if not impossible at times.

Here, we would like to add, below the ggplot, two lines with segments indicating roughly the number of persons of the sample with a certain age in each gender group. To do so, we will use the function *layout()* presented previously, which will create fields in the plotting window to fill them with the desired visualization.

```
# Two panes 1 for the line plot of predicted health and 2 for the age distributions
layout(matrix(c(1, 1,  1,  2), ncol=1, byrow=FALSE))

# Quick look at the fields in the plotting window
layout.show(n=2)
```

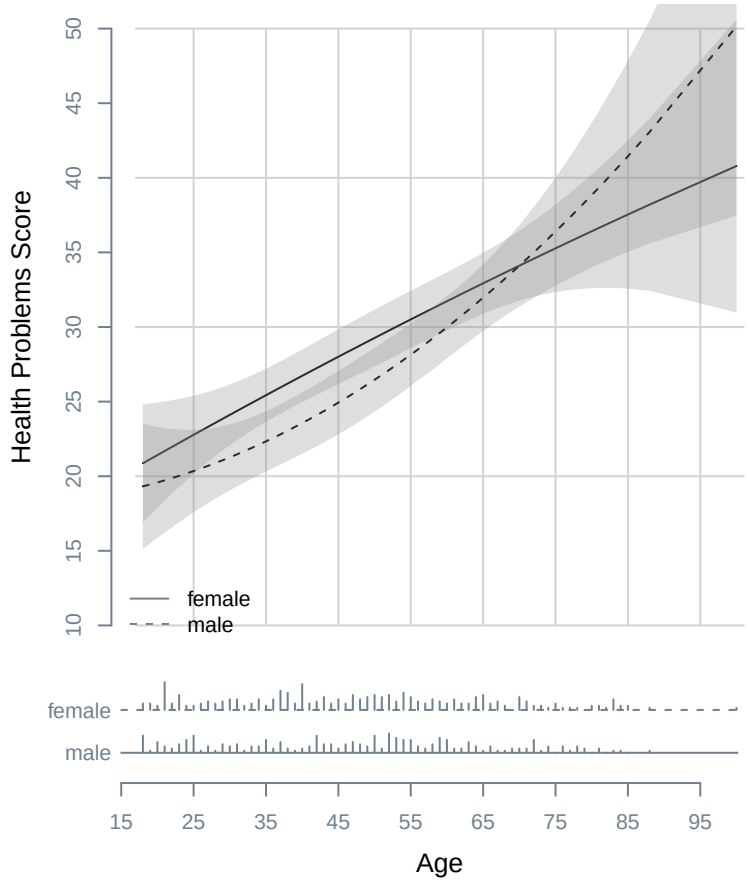



```
# Also, instead of the colors, for example because it is a black and white publication,
#  the line type is varied.

Lines=c("solid", "dashed") # Check ?par and lty for all different line types

# And we will create a palette of grey colors for the confidence intervals,
#  gray.colors() is found in the package grDevices
ColLine=rep("black", 2)
GraysCI=rep(gray.colors(4, start = 0.3, end = 0.9, gamma = 2, alpha = 0.3)[2],2)
```

```
# Start with an empty plot of some kind
# The x-axis width is the range of ages
# The y-axis height chosen here (10,50) is not the actual possible data range, but
#   rather serves as an example for illustration
# We do not draw the axes yet: xaxt="n", yaxt="n"


# Putting graphics:: before the plot function tells the engine that we want
#   the plot function from the package graphics (the generic function)
#    and not from package ggplot
# The nested notation 'packagename::function()' is good to remember when things
#   are not working as expected and many package libraries have been called

# Empty plot
par(mar=c(0.5,5,1,5))  # Control the amount of space around the figure
graphics::plot(1,1, col="white",
          xlim=c(min(Smpl_who$age)-1, max(Smpl_who$age)+1), xlab="", xaxt="n",
          ylim=c(10,50),  ylab="Health Problems Score", yaxt="n",
          cex.lab=1.2,
          bty="n",
          col.lab="black")

# Drawing the y-axis only
axis(2, at=seq(10,50,5), labels=seq(10,50,5),
     cex=0.5,
     col="slategrey",
     col.axis="slategrey",
     col.ticks="slategrey")

# Add grid lines to the plot just as in ggplot
# There is a function grid()... but it is finally handier to put segment lines on its own
# Draw the segments before the plot so that they are in the background
# The function segments() draws straight lines

HL=c(10,20,30,40,50) # The location of the horizontal lines
for(i in 1:length(HL)){
segments(min(Smpl_who$age)-1, HL[i], max(Smpl_who$age)+1, HL[i],
         col="lightgrey", lty="solid")
}

VL=c(25, 35, 45, 55, 65, 75, 85, 95) # The location of the vertical lines
for(i in 1:length(VL)){
segments(VL[i], 10, VL[i], 50,
         col="lightgrey", lty="solid")
}


S=c("male", "female")
# Adding a legend
legend("bottomleft",legend=c("female", "male"),
       col=rep("slategrey",2),
```

```
       bty="n",
       lty=Lines)


# Run a loop over the values that sex takes
 X=list()  # Naming the list objects in the loop where the information will be stored
 Y=list()
 XYouter=list()


 for(i in c(1:2)){

# [[i]] means that the values are stored in a list type object, this is working
#   the best in loops to avoid overwriting for example

 X[[i]]=Graph_values[which(Graph_values[,"sex"]==S[i]),"age"] # x-axis age in subroup
 Y[[i]]=Graph_values[which(Graph_values[,"sex"]==S[i]),"health"] # y-axis health in subgroup
 XYouter[[i]]=Graph_values[which(Graph_values[,"sex"]==S[i]),
                           c("health_ci_low","health_ci_up") ]


 lines(X[[i]], Y[[i]], lty=Lines[i], col=ColLine[i])
 polygon(c(X[[i]], rev(X[[i]])), c(XYouter[[i]][,2], rev(XYouter[[i]][,1])),
         col = GraysCI[[i]], border = NA)
 }

# The next empty plot to fill the lower field
par(mar=c(3.5,5,0,5)) # Determine the space around
                      # Remark: par(oma=c(,,,) does the same job on another metric)

 graphics::plot(1,1, col="white", xlim=c(min(Smpl_who$age)-1, max(Smpl_who$age)+1),
               ylim=c(0,60), bty="n", yaxt="n",xaxt="n", xlab="", ylab="")


# Make an x-axis which fits the two plots
axis(1, at=seq(15,100,10), labels=seq(15,100,10), cex=0.5, line=-0.7, col="slategrey",
     col.axis="slategrey", col.ticks="slategrey")


# mtext allows to add whatever text or labels in the margins

mtext("Age", side=1, line=2, col="black", cex=0.85)
mtext(S, 2, at=c(20,40), las=1, col="slategrey", cex=0.7)


for(i in c(1:2)){  # Open i loop for sex

  segments(15, i*20, 100, i*20, lty=Lines[i], col="slategrey")

  for(j in 1:length(which(Graph_values[,"sex"]==S[i]))){#j-loop for ages in i-subgroup

    segments(Graph_values[which(Graph_values[,"sex"]==S[i])[j],"age"], i*20,
             Graph_values[which(Graph_values[,"sex"]==S[i])[j],"age"],
             (i*20)+Graph_values[which(Graph_values[,"sex"]==S[i])[j],"N"],
```

```
            col="slategrey", pch=19)
  } # Close j
} # Close i
```




```
# Exercise

# The legend is not very visible and badly positioned
# The sizes of labels, legends etc... are not optimal, coloring could be rethought
# Add a title to the plot, eventually make more space on the very top
# Make it look more dramatic by increasing the space on the left
#  and the right of the plot (par(mar=c(,,,,)))
# Can male and female be written with a capital letter at the beginning?
```

Table one

Table construction is not too complex and basic knowledge of R allows to show most contents. To terminate, this workbook glimpses into a package function which easily summarizes data in a format ready to be published. The R-Package tableone provides functions to quickly and nicely set up a first table of descriptive statistics for reporting a study. It only requires to specify the scale level of the variables to present. The continuous variables (*as.numeric()*) are than presented as mean and standard deviation if not specified otherwise. Non-continuous variables (*as.factor()*, *as.character()*) are shown as frequencies and percentages. Results can be stratified by specifying stratification variables, such as gender or age categories for example.

```
library(tableone)

# Checking the scale level of the variables
Fac=c("sex", "getting_out","energy", "relating", "stress", "anxiety")
Num=c("age", "health")

for(i in 1:length(Fac)){Smpl_who[,Fac[i]]=as.factor(Smpl_who[,Fac[i]])}
for(i in 1:length(Num)){Smpl_who[,Num[i]]=as.numeric(Smpl_who[,Num[i]])}


# Variables to describe
vars=c("age", "health", "anxiety", "getting_out", "energy", "relating")

# Set the stratification variable here, for example a description per gender group
strata=c("sex")

CreateTableOne(vars=c("sex", vars), data=Smpl_who)
```

```
                     Overall
  n                    500
  sex = male (%)       223 (44.6)
  age (mean (SD))    47.53 (17.51)
  health (mean (SD)) 27.71 (16.82)
  anxiety = 2 (%)      427 (85.4)
  getting_out (%)
     1                 409 (81.8)
     2                  41 ( 8.2)
     3                  31 ( 6.2)
     4                  10 ( 2.0)
     5                   9 ( 1.8)
  energy (%)
     1                 232 (46.4)
     2                 134 (26.8)
     3                  83 (16.6)
     4                  39 ( 7.8)
     5                  12 ( 2.4)
  relating (%)
     1                 437 (87.6)
     2                  34 ( 6.8)
     3                  22 ( 4.4)
     4                   3 ( 0.6)
     5                   3 ( 0.6)
```

```
# Same table stratified by gender
```

```
CreateTableOne(vars=vars, strata=strata, data=Smpl_who)
```

```
                    Stratified by sex
                     female        male          p      test
  n                    277           223
  age (mean (SD))    47.83 (17.66) 47.15 (17.36)  0.668
  health (mean (SD)) 30.45 (16.36) 24.30 (16.80) <0.001
  anxiety = 2 (%)      226 (81.6)    201 (90.1)   0.010
  getting_out (%)                                 0.013
     1                 213 (76.9)    196 (87.9)
     2                  26 ( 9.4)     15 ( 6.7)
     3                  23 ( 8.3)      8 ( 3.6)
     4                   9 ( 3.2)      1 ( 0.4)
     5                   6 ( 2.2)      3 ( 1.3)
  energy (%)                                      0.001
     1                 112 (40.4)    120 (53.8)
     2                  73 (26.4)     61 (27.4)
     3                  52 (18.8)     31 (13.9)
     4                  30 (10.8)      9 ( 4.0)
     5                  10 ( 3.6)      2 ( 0.9)
  relating (%)                                    0.336
     1                 244 (88.4)    193 (86.5)
     2                  18 ( 6.5)     16 ( 7.2)
     3                   9 ( 3.3)     13 ( 5.8)
     4                   3 ( 1.1)      0 ( 0.0)
     5                   2 ( 0.7)      1 ( 0.4)
```

```
# The table can be saved through print(CreateTableOne(....))

# Exercise

# Make a descriptive table with 3 columns:
    #1. the overall descriptive
    #2. the female participants
    #3. the male participants
# Take out p-values and test columns.
```

Thank you and happy plot designing!