Making figures for RoM in R

Max Lindmark

28 april 2019

Contents

1 Introduction

First, this is a first draft and a work in progress so any feedback on this document or the figures are highly appreciated! This document has been put together by Max Lindmark (K-lab), with code and input from Massimiliano Cardinale massimiliano.cardinale@slu.se (H-lab) and Martin Ogonowski martin.ogonowski@slu.se (Sö-lab). We are also the R-contact-persons for RoM work, so please send any questions to your corresponding lab-contact. And if it's a general R-question, please do also email the R-mailing list Aqua r-users aqua-r-users@slu.se.

These code snippets are aimed to be as generic as possible. However, because almost all non-assessed species have different data, there is no guarantee that the code will work straight from pdf with your species. Do give it a try if you feel adventurous, but if you get stuck don't hesitate contacting us!

1.1 Why R?

The main argument for using code to create RoM figures is to standardize them across species and to do so while limiting repetitive work (you only need to write a script once!).

1.2 Basic prerequisitis

It is an advantage if you know some basic R. I strongly recommend using R-studio and working in a so called R-studio project (but it is not needed to reproduce this code). To create a project, open R-studio, click File/New Project/New Directory and specify where you want to save it. Open the *"your_project_name".Rproj* and click File/New Script and save that in your project folder. And that's it! The best thing is that now all your search paths are relative and not absolute. If you want to read in data, put the data inside the project folder and you don't have to specify the full search path (*"C:/R/RoM/data-file.csv"* or whatever), it's enough you give the name of the file only (more on that below!). Another benefit with a relative search path is that I don't have to worry about setting the working directory and changing the directory each time I try to rerun anyone elses script.

Lastly, this is an R Markdown document. This means R-code is text with grey background. You can copy these chunks of code to a new R-script in your R-studio project and run it from there.

1.3 What do these scripts do?

These scripts give an example of how you can use R to create standardized figures for RoM, and save them in a predefined, ready to be used without further editing.

I have chosen data for freshwater pike (*Esox lucious*) and Baltic turbot (*Scophthalmus maximus*) to illustrate how you can use this code, because it has all potential data for a RoM species (recreational, multiple areas, error bars, landings-by-country etc.). These data have been uploaded on github.

1.4 How do I use this for my species?

Most basic RoM figures that all species have in some form can be grouped into different categories. Each category will have its own script. Once you identify the best category for your species you should not have to modify the actual plotting code that much, as a minimum you only have to:

(**1**) Pre-define your variables (set y-axis names), (**2**) Prepare your data in the right format (see examples below and do the same for your species), (**3**) If a multi-series plot: put hash tags in front of stuff you don't need! E.g. if you don't have points to plot, make sure to put a hash tag before the geom_point()-function, which plots points. But do not worry! Instructions for how to do that will be in the described below!

The categories are:

- Fig. 1. Single series
- Fig. 2. Multiple series
- Fig. 3. Multiple series and y-axes
- Fig. 4. Landings-by-country

Once you know which plot-type you want, make sure you follow how the data should be structured in your csv-file, and load the theme (see "*Define ggplot theme*").

2 Preparation

2.1 Load libraries

Before starting, we need to install a few packages:

```r
rm(list = ls()) # clear the workspace from objects

# Provide package names
pkgs <- c("devtools",
          "ggplot2",
          "RCurl",
          "RCurl",
          "tidyr",
          "dplyr",
          "scales",
          "png",
          "knitr")

# Install packages if not installed
if (length(setdiff(pkgs, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(pkgs, rownames(installed.packages())))
}

# Load all packages
invisible(lapply(pkgs, function(x) require(x, character.only = T, quietly = T)))
```

2.2 Load example data and clean it up!

Now let's read in the example data (freshwater pike):

```
# Go to https://github.com/maxlindmark/ROM to view the data in the browser
dat <- read.csv(
  text = getURL("https://raw.githubusercontent.com/maxlindmark/ROM/master/pike.csv"),
  sep = ";")
```

For your own species, the code would typically look like this if you put the data inside your R Project folder:

```
#dat <- read.csv("pike.csv", sep = ";")
```

Inspect the data. It is important that you use the same column names as I do here. "Year" is Year, "error_plus" and "error_minu" are min and max of the confidence interval of the response variable - if you have any, otherwise just type NA. "Group" is our grouping variable, typically different areas and response_variable could be biomass, cpue or abundance. You will specify the unit before plotting, so that everyone can use the same base code, irrespective of which data the species have.

```
head(dat)
```

```
##   Year error_plus error_minu            Group Response_variable
## 1 1997         NA         NA Stora sj<f6>arna               115
## 2 1998         NA         NA Stora sj<f6>arna               114
## 3 1999         NA         NA Stora sj<f6>arna               149
## 4 2000         NA         NA Stora sj<f6>arna               145
## 5 2001         NA         NA Stora sj<f6>arna               121
## 6 2002         NA         NA Stora sj<f6>arna               145
```

Importing data in the right format is often non-trivial. Here I show you how you can rename the levels of the column Group (i.e. the lake-names in this case). We do this because the beauty of *ggplot* is that all important information in the plot will be inherited from the data.

```
levels(dat$Group) <- c("Fritidsfiske",
                       "Hjälmaren",
                       "Mälaren",
                       "Stora sjöarna",
                       "Vänern",
                       "Vättern")
head(dat)
```

```
##   Year error_plus error_minu         Group Response_variable
## 1 1997         NA         NA Stora sjöarna               115
## 2 1998         NA         NA Stora sjöarna               114
## 3 1999         NA         NA Stora sjöarna                149
## 4 2000         NA         NA Stora sjöarna                145
## 5 2001         NA         NA Stora sjöarna                121
## 6 2002         NA         NA Stora sjöarna                145
```

```
tail(dat)
```

```
##     Year error_plus error_minu        Group Response_variable
## 121 2012         NA         NA Fritidsfiske                NA
## 122 2013         NA         NA Fritidsfiske                NA
## 123 2014        223         77 Fritidsfiske               150
## 124 2015        184         82 Fritidsfiske               133
## 125 2016         NA         NA Fritidsfiske                NA
## 126 2017         NA         NA Fritidsfiske                NA
```

There is one last thing you might need to do with the data before proceeding with plotting, and that is to change the order of the levels in the data. When plotting, *ggplot* sets the levels of the data in alphabetical order, but here we want a specific order: the first "level" should correspond to the total and the last to any special level, such as the recreational data. This is how you would do that, just replace the levels with your own names.

```
dat$Group <- factor(dat$Group,
                     levels = c("Stora sjöarna",
                                "Vänern",
                                "Vättern",
                                "Mälaren",
                                "Hjälmaren",
                                "Fritidsfiske"))
```

Now the data look much better! You might not need to do these modification on your own data. **But you need** to make sure it is structured in the same way, that is: **1 row = 1 observation (not multiple columns for different areas)**

2.3 Define *ggplot* theme

Now that you have loaded and cleaned up your or the example data, we can move on with general plotting settings. First the color palette:

```
pal <- c("#56B4E9", "#009E73", "#F0E442", "#0072B2", "#E69F00", "#D55E00")
```

Second, we define the theme we will use for all plots. This is made to match as closely as possible to the RoM style. This applies to all figures styles.

```
theme_rom <- function(base_size = 12, base_family = "") {
  theme_bw(base_size = 12, base_family = "") +
    theme(
      axis.text = element_text(size = 8),
      axis.title = element_text(size = 8),
      axis.ticks.length = unit(0.05, "cm"),
      axis.line = element_line(colour = "black",
                               size = 0.3),
      text = element_text(family = "sans"),
      panel.grid.major = element_blank(),
      panel.grid.minor = element_blank(),
      panel.border = element_blank(),
      plot.title = element_text(hjust = 0.5,
                                margin = margin(b = -3),
                                size = 9.6,
                                face = "bold"),
      legend.position = "bottom",
      legend.text = element_text(size = 8),
      legend.background = element_rect(fill = "transparent"),
      legend.key = element_rect(fill = "transparent"),
      legend.box.margin = margin(-25,-25,-25,-25),
      aspect.ratio = 1,
      plot.margin = unit(c(5.5, 5.5, 20, 5.5),
                         "points")
      )
}
```

You can also source the theme-function directly from github. That way you'll always have the latest version!

```
u<-c("https://raw.githubusercontent.com/maxlindmark/scaling/master/R/raincloud_plot.R")
script <- getURL(u, ssl.verifypeer = FALSE)

eval(parse(text = script))
```

3 Let's start making figures!

3.1 Fig. 1. Single series

For illustration purposes, we will now pretend our example pike data set only is a single series, by filtering it to only contain the area “Stora Sjöarna”. If you have a “Fig.1 situation”“, your data might look something like this, with one column for year, one for area and one for tonnes.

```
dat1 <- dat %>%
  select(Year, Response_variable, Group) %>%
  filter(Group == "Stora sjöarna")

head(dat1)
```

```
##   Year Response_variable         Group
## 1 1997               115 Stora sjöarna
## 2 1998               114 Stora sjöarna
## 3 1999               149 Stora sjöarna
## 4 2000               145 Stora sjöarna
## 5 2001               121 Stora sjöarna
## 6 2002               145 Stora sjöarna
```

We now need to specify the **y-axis title**. For the pike data we can set them as:

```
y_axis <- c("Landningar (ton)")
```

Now go ahead and create the plot:

```
p1 <- ggplot(dat1, aes(Year, Response_variable)) +
  geom_bar(data = dat1,
           aes(x = Year, y = Response_variable),
           stat = "identity", color = pal[1], fill = pal[1],
           width = 0.6) +
  labs(x = "", y = y_axis) + # here's where the axis title is called
  guides(color  = FALSE) +
  scale_x_continuous(expand = c(0, 0), breaks = scales::pretty_breaks(n = 6)) +
  scale_y_continuous(expand = c(0, 0), breaks = scales::pretty_breaks(n = 5)) +
  theme_rom()                # here we call our predefined theme
```

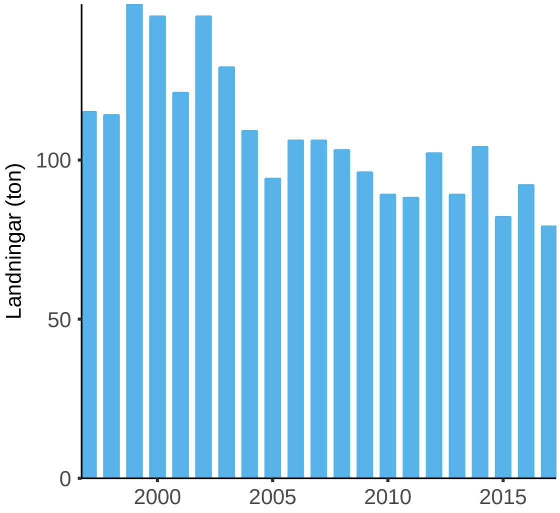

Figure 1: Gädda in the "Great Lakes" - example of a single-series plot using R

Save the file (to your working directory)

```
ggsave("Fig_1.tiff", plot = p1, dpi = 300, width = 8, height = 8, units = "cm")
```

3.2 Fig. 2. Multiple series

We now need to specify y-axis title, and in addition we need to define how many levels we have! For the pike data we can set them as:

```
y_axis <- c("Landningar (ton)")
main_series <- c("Stora sjöarna")   # This is the sum of all other series
n_lev <- length(unique(dat$Group))  # Number of unique lakes
special_series <- c("Fritidsfiske") # This series must be drawn with points, so we need to
                                    # separate it from the rest.
```

Note that we in this example use multiple data series, and that one of them is is recreational fisheries that in turn has error bars. **These need to be in columns, so that we have a high (error__plus) and a low (error__minu) column. Note also that they are NA when the Group is not equal to recreational fisheries**

```
head(dat)
```

```
##   Year error_plus error_minu         Group Response_variable
## 1 1997         NA         NA Stora sjöarna               115
## 2 1998         NA         NA Stora sjöarna               114
## 3 1999         NA         NA Stora sjöarna               149
## 4 2000         NA         NA Stora sjöarna               145
## 5 2001         NA         NA Stora sjöarna               121
## 6 2002         NA         NA Stora sjöarna               145
```

```
tail(dat)
```

```
##     Year error_plus error_minu        Group Response_variable
## 121 2012         NA         NA Fritidsfiske                NA
## 122 2013         NA         NA Fritidsfiske                NA
## 123 2014        223         77 Fritidsfiske               150
## 124 2015        184         82 Fritidsfiske               133
## 125 2016         NA         NA Fritidsfiske                NA
## 126 2017         NA         NA Fritidsfiske                NA
```

And with that set, we can make the second figure, with multiple levels:

```
p2 <- ggplot(dat, aes(Year, Response_variable, color = Group)) +
  geom_bar(data = subset(dat, Group == main_series),
           aes(x = Year, y = Response_variable),
           stat = "identity", color = pal[1], fill = pal[1],
           width = 0.6) +
  geom_line(data = dat, aes(Year, Response_variable, color = Group, alpha = Group),
            size = 1) +
  geom_point(data = subset(dat, Group == special_series), # here we set our special series
             aes(Year, Response_variable, fill = Group),
             size = 2, color = pal[max(n_lev)]) +
  # above the number of level enters (max(n_lev))
  geom_errorbar(data = dat, aes(x = Year, ymin = error_minu, ymax = error_plus,
                                color = Group),
                show.legend = FALSE, width  = 1) +
  scale_alpha_manual(values = c(rep(1, (n_lev-1)), 0)) +
  scale_color_manual(values = pal[seq(1, n_lev)]) +
  # above we set the line between rec fisheries transparent
  labs(x = "", y = y_axis) +
  guides(fill  = FALSE,
         alpha = FALSE,
         color = guide_legend(nrow = 3,
                              title = "",
                              override.aes = list(size = 1.3,
                                                  color = pal[seq(1, n_lev)]),
                              keywidth = 0.3,
                              keyheight = 0.1,
                              default.unit = "inch")) +
  scale_x_continuous(expand = c(0, 0), breaks = scales::pretty_breaks(n = 6)) +
  scale_y_continuous(expand = c(0, 0), breaks = scales::pretty_breaks(n = 5)) +
  theme_rom()
```

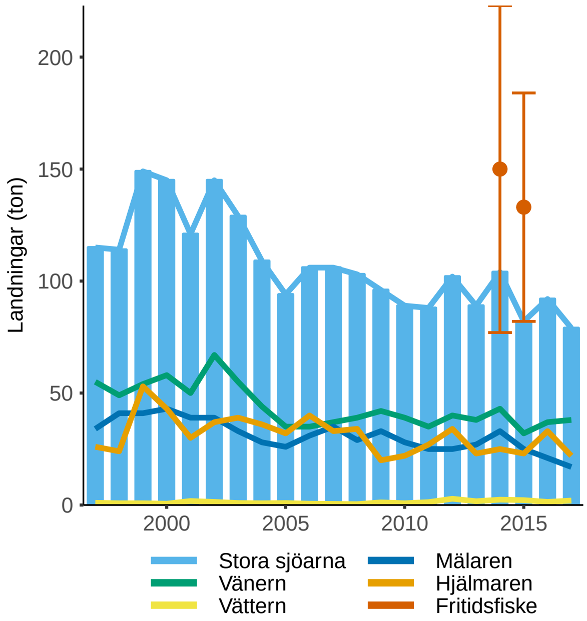

Figure 2: Gädda in the Great Lakes - example of a multiple-series plot using R

Now save the file (to your working directory)

```
ggsave("Fig_2.tiff", plot = p2, dpi = 300, width = 8, height = 8, units = "cm")
```

Let's say that you have 3 different areas and no special series and no error bars. Then you simply need to hash tag the code the plots those features. We subset the full data to select only the total and lake Mälaren as an example:

```
dat2 <- dat %>%
  select(Year, Response_variable, Group) %>%
  filter(Group %in% c("Stora sjöarna", "Mälaren"))
```

We now need to update the number of levels in the data:

```
n_lev <- length(unique(dat2$Group))
```

Repeat the general Fig. 2 plot, but remove the points and error bars using hashtags:

```
p2a <- ggplot(dat2, aes(Year, Response_variable, color = Group)) +
  geom_bar(data = subset(dat2, Group == main_series),
           aes(x = Year, y = Response_variable),
           stat = "identity", color = pal[1], fill = pal[1],
           width = 0.6) +
  geom_line(data = dat2, aes(Year, Response_variable, color = Group, alpha = Group),
            size = 1) +
#geom_point(data = subset(dat, Group == special_series), # here we define our special series
#           aes(Year, Response_variable, fill = Group), size = 2, color = pal[max(n_lev)]) +
#geom_errorbar(data = dat, aes(x = Year, ymin = error_minu, ymax = error_plus,
#                              color = Group),
#              show.legend = FALSE, width  = 1) +
#scale_alpha_manual(values = c(rep(1, (n_lev-1)), 0)) +
  scale_color_manual(values = pal[seq(1, n_lev)]) +
  labs(x = "", y = y_axis) +
  guides(fill  = FALSE,
         alpha = FALSE,
         color = guide_legend(nrow = 3,
                              title = "",
                              override.aes = list(size = 1.3,
                                                  color = pal[seq(1, n_lev)]),
                              keywidth = 0.3,
                              keyheight = 0.1,
                              default.unit = "inch")) +
  scale_x_continuous(expand = c(0, 0), breaks = scales::pretty_breaks(n = 6)) +
  scale_y_continuous(expand = c(0, 0), breaks = scales::pretty_breaks(n = 5)) +
  theme_rom()
```

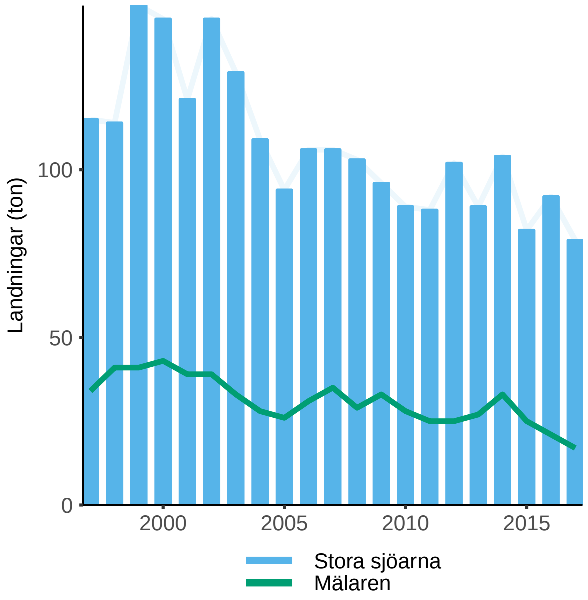

Figure 3: Gädda in the Great Lakes - example of a multiple-series plot using R

3.3 Fig. 3. Multiple series, two y-axes

In some cases, you may be forced to plot different series on two different axes, e.g. when you have things in different units (landings and and an index perhaps). Using the pike-data set, we will invent such a scenario treating the “Fritidsfiske” as if on a different unit.

```
## Increase the value of Fritidsfiske:
dat3 <- dat
dat3$Response_variable <- ifelse(dat3$Group == "Fritidsfiske",
                                 dat3$Response_variable * 100,
                                 dat3$Response_variable)

# Define plot settings again
y_axis <- c("Landningar (ton)")
y_axis2 <- c("Fångst per ansträngning (antal/timme)") #2nd axis title
main_series <- c("Stora sjöarna")   # This is the sum of all other series
n_lev <- length(unique(dat3$Group)) # Number of unique lakes
special_series <- c("Fritidsfiske") # This series must be drawn with points, so we need to
                                    # separate it from the rest.

# Define temporary maximum of primary y-axis
max_yax <- max(subset(dat3, Group == main_series, na.rm = TRUE)$Response_variable)*1.2
```

And with that set, we can make the second figure, with multiple levels. Hashtag geom_point and geom_errorbar as these are no longer on the same scale.

```
p3 <- ggplot(dat3, aes(Year, Response_variable, color = Group)) +
  geom_bar(data = subset(dat3, Group == main_series),
           aes(x = Year, y = Response_variable), stat = "identity",
           color = pal[1], fill = pal[1],
           width = 0.6) +
  geom_line(data = dat3, aes(Year, Response_variable, color = Group, alpha = Group),
            size = 1) +
#geom_point(data = subset(dat, Group == special_series), # here we set our special series
#           aes(Year, Response_variable, fill = Group), size = 2, color = pal[max(n_lev)]) +
#geom_errorbar(data = dat, aes(x = Year, ymin = error_minu, ymax = error_plus,
#                              color = Group),
#              show.legend = FALSE, width  = 1) +
  scale_alpha_manual(values = c(rep(1, (n_lev-1)), 0)) +
  scale_color_manual(values = pal[seq(1, n_lev)]) +
  labs(x = "", y = y_axis) +
  guides(fill  = FALSE,
         alpha = FALSE,
         color = guide_legend(nrow = 3,
                              title = "",
                              override.aes = list(size = 1.3,
                                                  color = pal[seq(1, n_lev)]),
                              keywidth = 0.3,
                              keyheight = 0.1,
                              default.unit = "inch")) +
  scale_x_continuous(expand = c(0, 0), breaks = scales::pretty_breaks(n = 6)) +
  scale_y_continuous(expand = c(0, 0),
                     limits = c(0, max_yax), breaks = scales::pretty_breaks(n = 5)) +
  theme_rom()
```

Now the trick is to first transform the variable that is to be plotted on the second y-axis to a similar scale, and then reset the scale to the original values::

```
## Scale for 'y' is already present. Adding another scale for 'y', which
## will replace the existing scale.
```

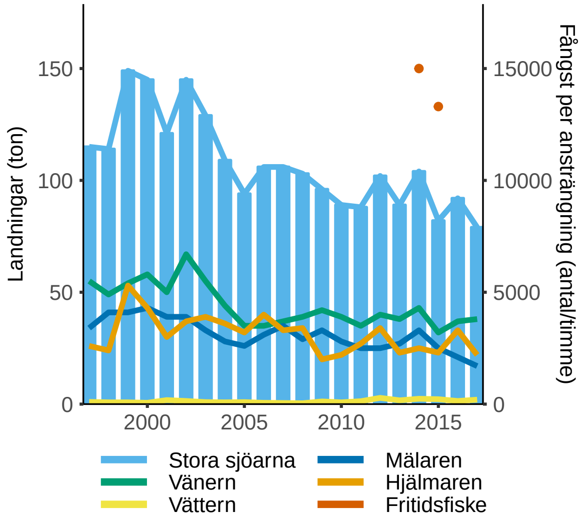

Figure 4: Gädda in the Great Lakes with secondary y-axes- example of a multiple-series plot using R

Now save the file (to your working directory)

```
ggsave("Fig_3.tiff", plot = p3f, dpi = 300, width = 8, height = 8, units = "cm")
```

This is not a very straightforward plot to make. First you need to manually find a value that scales the differnt units so they become close (see above when making p3f). When scaling back you need to verify all the data-points are within the plot range (which now is manual! By default it's based on your data but we need to bypass that). Also, in order to inheret the legend, we need at least one geom with all data we want in the legend. This is geom_line in the above case. In order to not get a gigantic primary y-axis, we set it manually and simple plot the series that is on a different scale outside the plot window. So, many things to keep in mind when/if adapting this code to your data!

3.4 Fig. 4. Landings by country

For this example, we will use Baltic turbot data:

```
u<-c("https://raw.githubusercontent.com/maxlindmark/ROM/master/Landingsbycountry_Baltic_Turbot.csv")
tur <- read.csv(
  text = getURL(u),
  sep = ";")

head(tur)
```

```
##   Year Danmark Estland Finland Lettland Litauen Polen Sverige Tyskland
## 1 1965   0.001   0.001   0.001    0.001   0.001 0.001   0.001       42
## 2 1966  37.000   0.001   0.001    0.001   0.001 0.001   0.001       58
## 3 1967  34.000   0.001   0.001    0.001   0.001 0.001   0.001       17
## 4 1968  32.000   0.001   0.001    0.001   0.001 0.001   0.001       70
## 5 1969  26.000   0.001   0.001    0.001   0.001 0.001   0.001       61
## 6 1970  24.000   0.001   0.001    0.001   0.001 0.001   2.000       45
##   Ryssland Andra.l.e4.nder    FishStock               Swedishname
## 1    0.001           0.001 tur.27.22-32 Piggvar i <d6>stersj<f6>n
## 2    0.001           0.001 tur.27.22-32 Piggvar i <d6>stersj<f6>n
## 3    0.001           0.001 tur.27.22-32 Piggvar i <d6>stersj<f6>n
## 4    0.001           0.001 tur.27.22-32 Piggvar i <d6>stersj<f6>n
## 5    0.001           0.001 tur.27.22-32 Piggvar i <d6>stersj<f6>n
## 6    0.001           0.001 tur.27.22-32 Piggvar i <d6>stersj<f6>n
```

```
# Rename column
tur <- tur %>% rename("Andra länder" = "Andra.l.e4.nder")

# Make data long (1 row, 1 observation)
tur_l <- tur %>% gather(land, landningar, 2:11, na.rm = TRUE)

# Show total tonnes by country
tur_l %>%
  group_by(land) %>%
  summarize(tot_land = sum(landningar)) %>%
  arrange(desc(tot_land))
```

```
## # A tibble: 10 x 2
##    land         tot_land
##    <chr>           <dbl>
##  1 Danmark      7136.   
##  2 Polen        5287.   
##  3 Tyskland     3295    
##  4 Sverige      2224.   
##  5 Ryssland      649.   
##  6 Litauen       466.   
##  7 Lettland      365.   
##  8 Andra länder    0.053
##  9 Estland         0.053
## 10 Finland         0.053
```

```
# We will only plot the top 5 countries and the total. Other countries will
# be regrouped to "Andra länder"
unique(tur_l$land)
```

```
##  [1] "Danmark"      "Estland"      "Finland"      "Lettland"    
##  [5] "Litauen"      "Polen"        "Sverige"      "Tyskland"    
##  [9] "Ryssland"     "Andra länder"
```

```
tur_l$land <- ifelse(tur_l$land %in% c("Litauen",
                                       "Lettland",
                                       "Estland",
                                       "Finland"),
                     "Andra länder",
                     tur_l$land)

unique(tur_l$land)
```

```
## [1] "Danmark"      "Andra länder" "Polen"        "Sverige"
## [5] "Tyskland"     "Ryssland"
```

We now need to specify the **y-axis title**. For the turbot landings data we can set them as:

```
y_axis <- c("Landningar (1000 ton)")
```

Now go ahead and create the plot:

```
p4 <- ggplot(tur_l, aes(x = Year, y = landningar/1000, fill = land)) +
  geom_bar(stat = "identity") +
  labs(x = "", y = y_axis) +
  scale_fill_manual(values = pal) +
  labs(x = NULL) +
  scale_x_continuous(expand = c(0,0)) +
  scale_y_continuous(expand = c(0,0)) +
  guides(fill = guide_legend(nrow = 2,
                             title = "",
                             keywidth = 0.1,
                             keyheight = 0.03,
                             default.unit = "inch")) +
  theme_rom() +
  theme(legend.box.margin = margin(-15,-15,-15,-15)) # Note we here overwrite
                                                     # the default in the RoM-theme
```

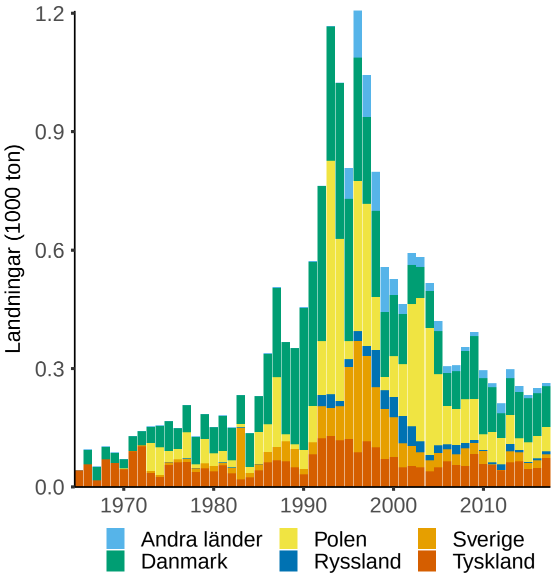

Figure 5: Landings of turbot in the Baltic Sea - example of a landings-by-country plot using R

Save the file (to your working directory)

```r
ggsave("Fig_4.tiff", plot = p4, dpi = 300, width = 8, height = 8, units = "cm")
```