[Stats602] Tutorial 2: Graphs

Revati Shivnekar

2023-08-11

Quick introduction

So far we know how to assign variables and write functions. We are going to spend more time on functions today because we will be using a whole bunch of them! But the main objective of this tutorial is learning to plot. We will start with a clean scatter plot today and as we go on, we will refine our technique. You are also encouraged to try your hand at plotting on your own.

This worksheet covers the following. There are some exercises at the end of each section. Make sure you do all of the exercises. These exercises will not be graded but they will help you solve the assignments much faster.

Contents

1. FUNCTIONS

Functions make our lives easy. You have to write a function only once (and save it!) or you can use the functions that are already written by someone else. When you talk to functions in the syntax they understand, they will save you tons of time and long lines of complicated code. Best thing about them is that most often you do not need to know how they are doing any of this. Just knowing the syntax is enough.

For instance, `sample()` is a base function which means that when you open RStudio, R already knows what it does. If *you* want to know what it does, type the following and look at the bottom right panel on your RStudio:

```
?sample()
```

Most functions have arguments which are the terms we pass in its bracket. You will find the description of each of these when you type `?function_name()`. Some of these arguments are keyword arguments which you *have* to input when you call a function. A telltale sign that an argument is a keyword argument is the lack of `=`. In the example above, `x` and `size` are keyword arguments. When you call this function, you must pass appropriate elements for these arguments like this:

```
sample(x = 1:1000, size = 5)
```

```
## [1]  52  32 189 573 858
```

The rest are default arguments which have some values assigned already. If you want to change them you can. Make sure to read what they mean though!

```
sample(x = 1:4, size = 5, replace = TRUE)
```

```
## [1] 2 3 4 4 4
```

```
#sample(x = 1:4, size = 5, replace = FALSE) #why doesn't this work? Figure out!
```

Exercise: Try out the following functions. See if you can figure out what each one does and which arguments are keywords. You have to also understand which appropriate elements to pass for these functions to do what they do!

NOTE: You can use `?write_function_name_here()` to know how a function works by reading the documentation. If you ABSOLUTELY can't figure out after reading it, then look it up online.

1. abs()
2. sqrt()
3. round()

4. cat() and print()

5. floor() and ceiling()
6. sign()
7. exp()

8. log()
9. as.Date()

---

2. INSTALLING AND LOADING PACKAGES

A package is a bundle of functions (and some other things) that some good Samaritans have written and made available to all of us. You can use the functions within packages ONLY after making it available to the current memory.

For starters, you have to get the package you want from the internet. This is called installing. Let's install some packages we will be using in this course:

```
#install.packages(c("ggplot2", "dplyr", "tidyverse", "tidyr", "shiny", "plotly"))
#install.packages('ggplot2')
```

Installing makes packages available to your PC. However, to use them in a particular session of R, you need to load them into your R environment. You need not load all of the packages you have; just the ones you need.

Remember: you need to install a package only once but you will have to load it every time you want to run the script. And this is how you do it:

```
#library(ggplot2)
#library(tidyverse)
```

Quick tip: Bottom right pane has a tab for packages. You can install and then load (by checking off) packages from there.

Exercise: Find out what the packages we just installed do without going online.

NOTE: If you have trouble installing a package, check whether you have spelled the name and written the syntax it properly. If you have, your best bet is to find help online by copy-pasting the error you are getting.

---

3. IMPORTING AND MANIPULATING DATA

You need data to do any statistics. You can make data or you can use what is available to you. You can get data from online sources or you can get it from your computer. We will stick to getting data from online sources for this tutorial. The rest we will learn as we go on.

NOTE: From this point, I am going to anchor this tutorial on some chapters from Andy Field's book 'Discovering Statistics with R'. I will alter the material whenever needed but you can read more if you wish to in the book. In fact, the main purpose of using this material is that you go through the book on your own and learn the stuff not covered here. The book is very easy to get through for programming and statistics novices.

```
df <- read.delim('https://www.discoveringstatistics.com/docs/FacebookNarcissism.dat')
View(df) #opens the dataframe in a new tab script panel
```

In the first line above, I imported the dataframe by simply copying the link between `'...'` and using the function `read.delim()`. I have also saved it as `df`. Now whenever I want to refer to this dataframe, I will use `df`. You can give it any name you like. When you run only this line, you will see it add to your environment but you still haven't *seen* it. For that, we write the second like. See what happens.

This dataframe has 4 variables. Field gives the following description for them:

1. *id:* a number indicating from which participant the profile photo came.
2. *NPQC_R_Total:* the total score on the narcissism questionnaire.
3. *Rating_Type:* whether the rating was for coolness, glamour, fashion or attractiveness (stored as strings of text).
4. *Rating:* the rating given (on a scale from 1 to 5).

We can do a lot of things with this dataframe using functions. There are specialized packages (some of them we installed in the earlier section) to manipulate these dataframes.

Exercise: Do the following exercise to get to know some of these functions. I have written where these functions are used after the `#` for the first few. Type the rest out in your script, figure out what inputs they need and see how they can work for you. Remember to use `?`.

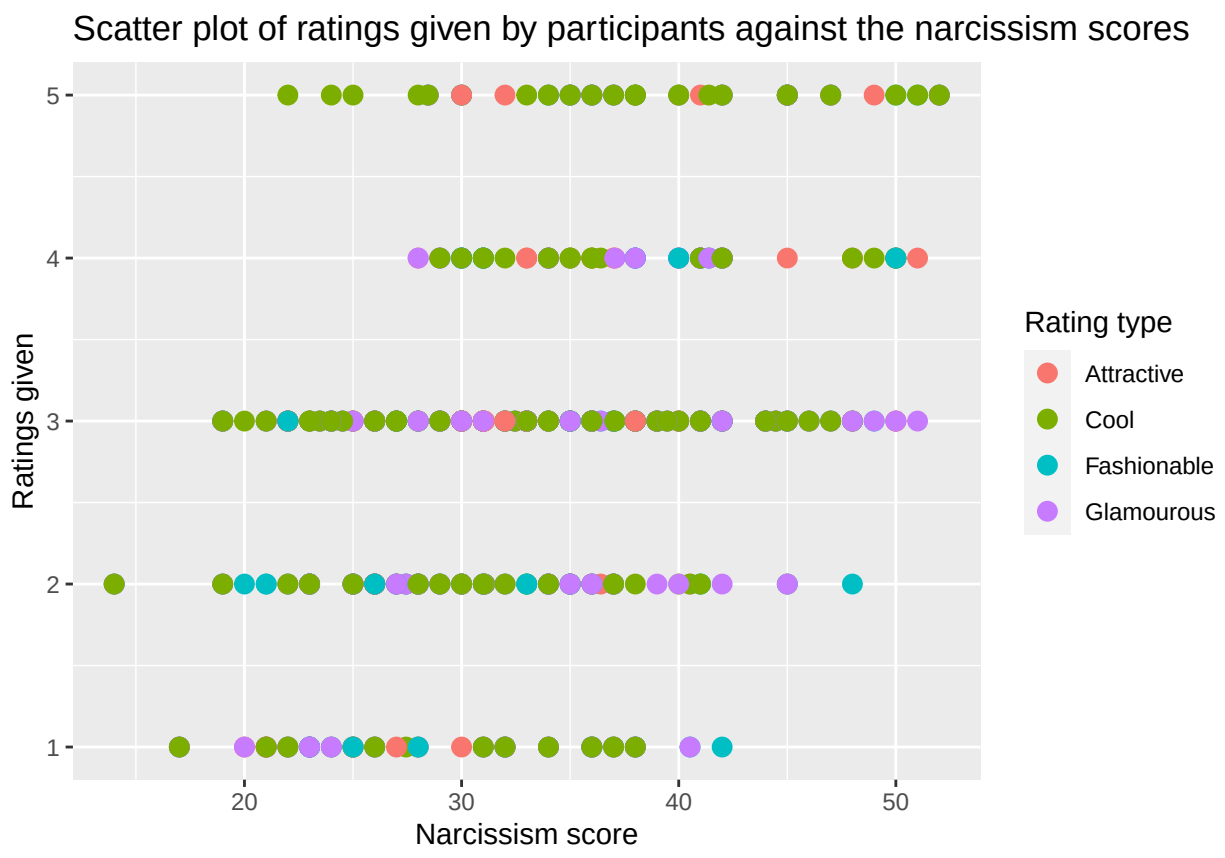
1. head(df, 10) `# show top 10 rows`
2. class(df) `# class can be used to know what data type the object is`
3. df[1] `# first col of mpg`
4. df$id `# show the manufacturer column of id`
5. df[1:10, 2:4] `# returns df rows 1 to 10 and columns 2 to 4`
6. summary(df)
7. nrow()
8. ncol()
9. dim()

10. rownames()
11. colnames()
12. subset()

---

4. GRAPHS!

All right. So far we have imported a dataframe in our environment, we know how it looks like and we (hopefully!) have a vague idea of how to manipulate it. When I was doing MSc, I was told that every good statistician first *eyeballs* the data (by which they meant need to see plots, plots and some more plots) before running any analysis. We may not be any good at statistics yet, but we can pretend and start graphing!



This is the graph that we want to create at the end of this tutorial with the data that we have. Let's do this step-by-step.

Step 0: using `ggplot` Graphing can be complex unless you have access to functions that are already written. This is where the packages come in handy. You will find lots of them out there for this purpose but the most flexible and popular (hence, you are very likely to get help online) is the library `ggplot2` which we are also going to use. You have already installed it, we just have to load it.

```
library(ggplot2) #loaded the package
```

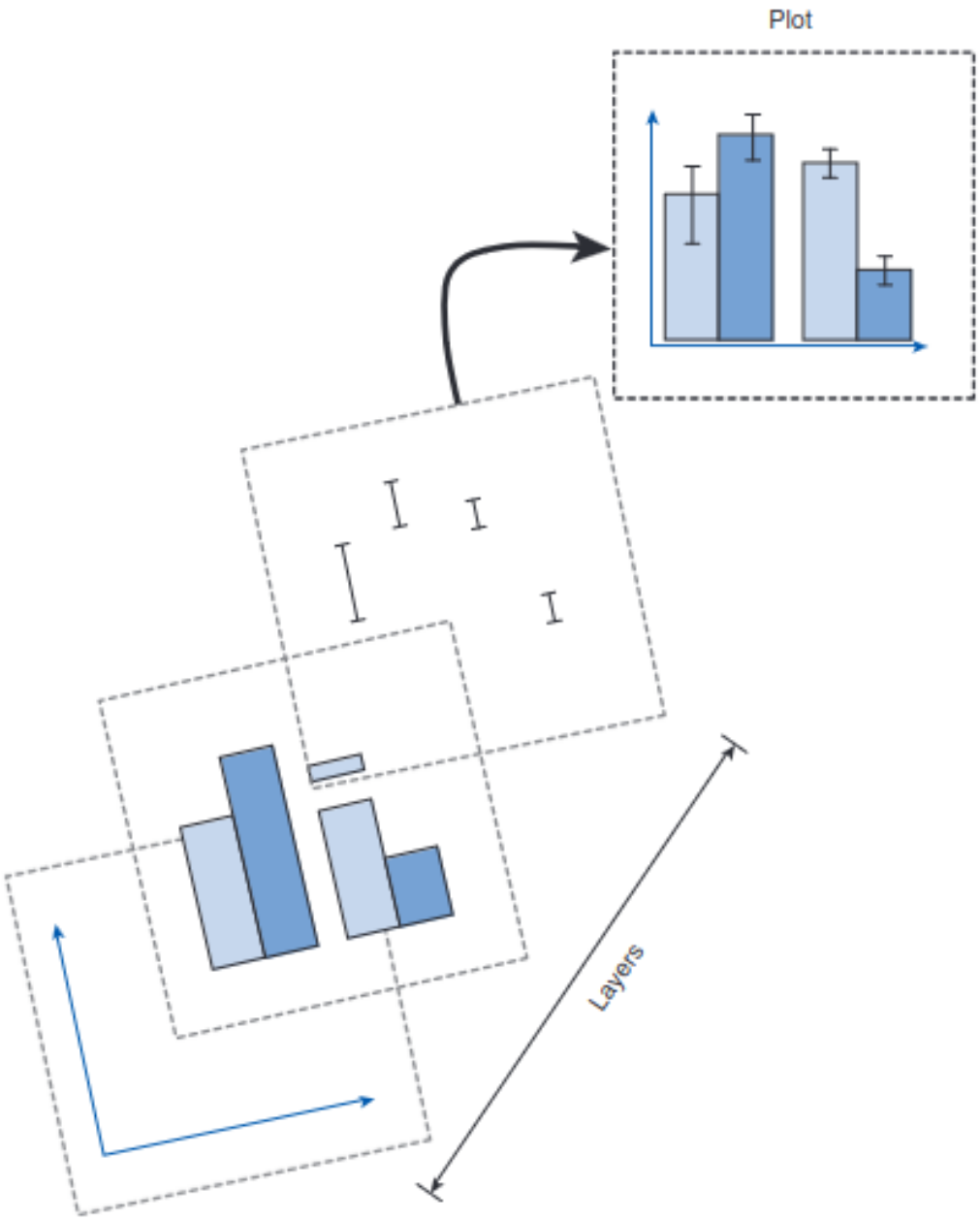


Figure 1: From Andy Field's Discovering Statistics

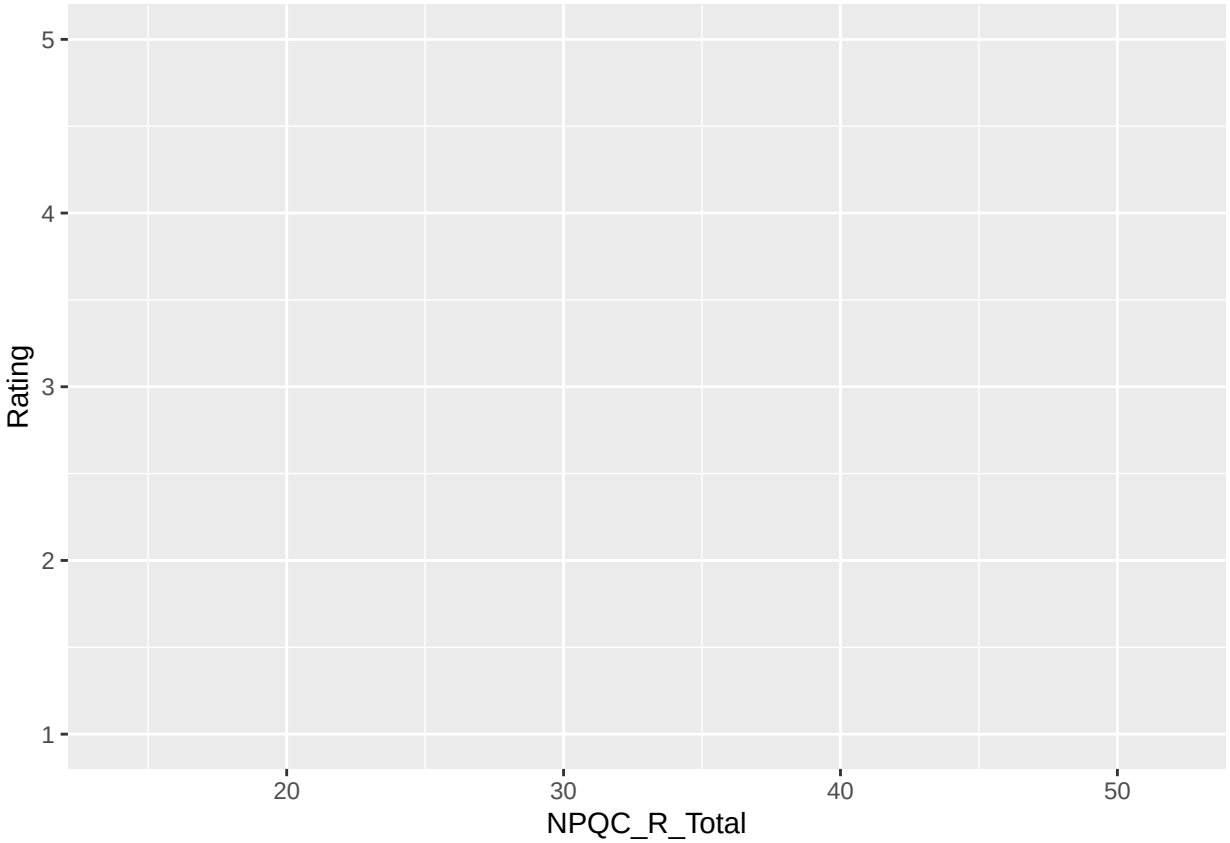
Step 1: Create a ggplot object Plotting works in layers. Field's illustration will make sense the best:

We will start with laying down the first layer. This code is only going to initialize a ggplot object, meaning, it is simply going to get ggplot ready to do some plotting! `ggplot()` needs to now what you want to plot, which you are going to let it know by passing `df` as one of the arguments.

```
ggplot(df)
```

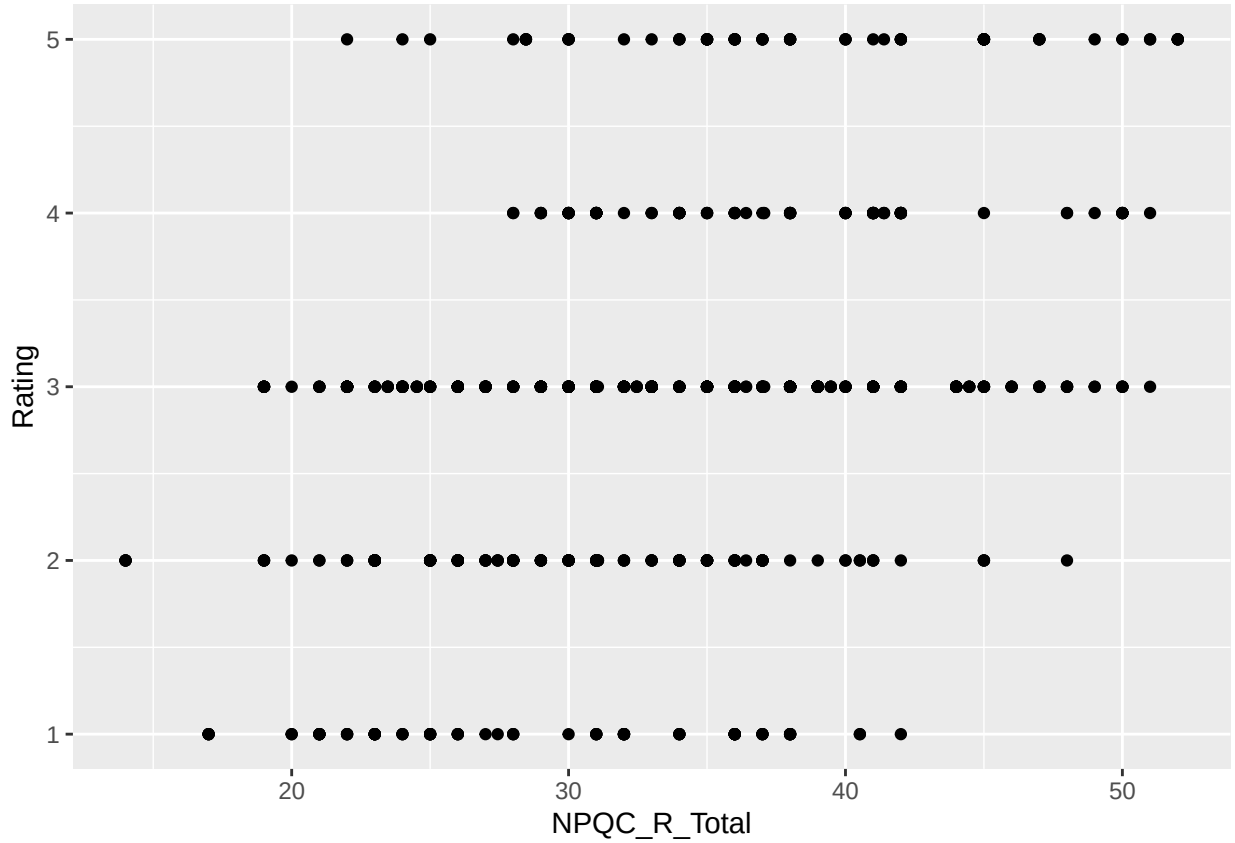

Step 2: Define axes Add axes by using `aes()` and our plot shows a sign of some life! By passing these arguments, I tell R that the variable `my.1st.plot` is a ggplot object. I want it to use `data = df`, particularly, the columns `mapping = aes(NPQC_R_Total, Rating)` from it.

```
ggplot(data = df, mapping = aes(NPQC_R_Total, Rating))
```



Step 3: Plot the data But where is the data in the plot? Here is the data in the plot:

```
ggplot(data = df, mapping = aes(NPQC_R_Total, Rating)) +
    geom_point()
```



`ggplot` can plot different plots including scatter, boxplots, histograms etc. Each of these have different functions. We add a data later onto the axes (from **Step 2**) by using the appropriate function for the kind of graph we need.

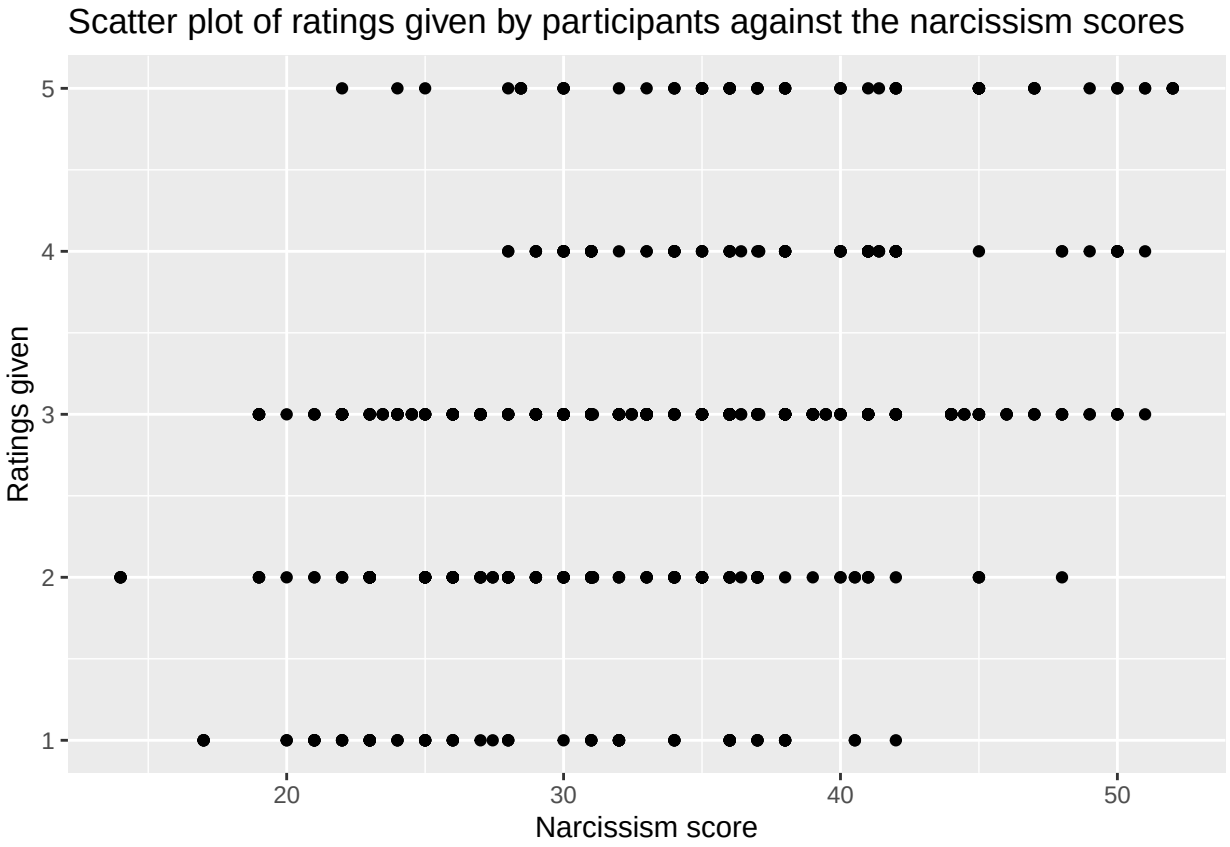
For scatter plots, the function is `geom_point()`. You will see if you use `?geom_point()` that it can also take `mapping` as one of the arguments. But since we have passed this argument before already, we need not do it again here.

PS: we add layers by using `+`.

Step 4: Title and axes labels Finally, let's change some text to make the plot easier to understand.

```
my.1st.rplot <- ggplot(data = df, mapping = aes(NPQC_R_Total, Rating)) +
    geom_point() +
    ggtitle('Scatter plot of ratings given by participants against the narcissism scores') +
    ylab('Ratings given') +
    xlab('Narcissism score')

my.1st.rplot #don't just save it; call it!
```



Exercise: So far you have been using `?` to get help within R environment. It's a quick but often times an impoverish source. If you have specific doubts, then you can look them up online. You may not find the answers you are looking for right away, but with practice you will know what exactly to ask to get what you want. For the following exercise, go online along with using `?`.

1. Can you change the shape, size and color of all the data points in `my.1st.rplot`?
2. Like there is `geom_point()` for scatter plots, what are the functions for histograms, box plots, and violin plots? Use different columns in `df` and plot these three.
3. You may have noticed that our graph doesn't quite look like the one in the beginning. I have color coded the data based on what type of rating it is. Find out how you can do that.