Introduction to R

Seija Sirkiä

2016-03-30

Contents

1 Getting used to R

1.1 What is R?

- A programming language and an environment
  - For data handling, data analysis and graphical display
  - "A vehicle for newly developing methods"
  - Extended with a large collection of packages
- Can be run on many different platforms (Windows, Linux, Mac. . .)
- Can be used with several different user interfaces (UI's)
  - Therefore, can look different in different situations

1.2 What is so great about R?

- Not everything about R is great!
  - Particularly "real" programmers tend to dislike it
- Much of the greatness comes from the community, other users
- Packages (libraries): all the methods you can imagine
  - and if you can think of a method that isn't there yet, you can create the package!
- In comparison to other statistical software such as SPSS or SAS: flexibility and freedom, efficiency of use, price (that is, lack of it)
- In comparison to other programming languages / environments such as Python: easier to do data analysis in particular, more straightforward to create plots and graphs

1.3 How to learn R?

- By using it! No way around it
- Good sources for help:
  - An introduction to R; by the R core team available at R homepage and from R help
  - www.datacamp.com; interactive tutorials using Shiny (*"Why am I even giving this course?"*)

  - Everything at https://www.rstudio.com/online-learning/
- There is a rather steep initial learning curve but probably any online course or some other such source will do the trick if you just put the effort in

1.4 What is there to learn

- The concept of typing in commands (which will evolve in to writing scripts)
- The concept of objects in the workspace; assignment
- Basic data structures: vectors, factors, data frames, lists
- Basic data types: numeric, character, logical
- Reading in data
- Handling data
- Using (statistical and such) methods via functions, producing graphs etc
- Finding help in the manual pages
- More advanced: writing your own functions, "smart" and "R-like" ways of doing things
- Miscellania: good habits, RStudio workflow etc.
- Alternative ways to do some things: specifically the packages `dplyr`, `tidyr`, `stringr`, `ggplot2`, ...

1.5 RStudio interface, first look

Four panes:

- Up left: scripts and data frame views
  - Note that this is not shown at the first startup or when ever you start a new project. You usually want to bring it up at once e.g. by starting a new R script from the New button or the File menu
- Bottom left: R console
- Top right: contents of the workspace, command history
- Bottom right: graphs, help, files, packages

1.6 Getting started with R console

Simple calculations happen using the common characters for the operations, with parentheses where needed:

```
2+3*6
```

```
## [1] 20
```

```
2.5/5-2
```

```
## [1] -1.5
```

```
(2^4+2)/-9
```

```
## [1] -2
```

```
1/0
```

```
## [1] Inf
```

Note that division by zero results in `Inf` and causes no errors - which might not be what you want! `Inf` can be used in calculations without errors as long as the result is defined (often another `Inf`).

More mathematical things:

```
sin(2*pi)
```

```
## [1] -2.449213e-16
```

```
sqrt(-1)
```

```
## Warning in sqrt(-1): NaNs produced
```

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

```
sqrt(-1+0i)
```

```
## [1] 0+1i
```

```
log(exp(1))
```

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

1.7 Expressions and assignments

- *Expressions* are formed by calculations and function calls, such as above, something that can be *evaluated* so that a result is found. If the evaluation can be done to the end, the result is printed (and lost!). If not, usually an error message appears.
- The results from an expression can be saved for later use using *assignment*: assignment happens with `<-` ("less than" and minus sign, forming a little arrow)
  - On the left hand side you put a name for a *variable* (can also be called an *object*, unfortunately both of these words have another specific meaning in statistics or programming)
  - On the right hand side you put an expression. The result is then not printed but
  - Assigned objects can then be used in further expressions
  - Overwriting an existing assignment will happen without any warning
  - Assigned objects are held in the *workspace*. In RStudio, the contents of the workspace are shown in the top right panel on the Environment tab

This is the universal logic of how the R console works!

Rules for variable names:

- The name must begin with a letter

- The name can contain numbers, dots and underscores, and is case sensitive.
- Letters with accents and umlauts may or may not work or be portable; try to avoid.

Note! The little arrow `>` at the beginning of the line in the console is called a *prompt*. It tells you that R is ready for the next command. If you see it unexpectedly turn in to a plus sign, `+`, it means that the previous command wasn't complete, R is still waiting for that to end before evaluating it. To get rid of that either complete the expression (usually a missing closing parenthesis) or press Esc to cancel. On the other hand, you can deliberately divide a long expression on several lines, in which case you do want the prompt to turn to `+`.

1.8 Activity 1

- How many minutes are there in a week?
- A glass vase of 39.90€ is put on 30% sale. What is the new price?
- Body mass index is calculated as weight (in kg) per height (in m) squared. Find yours! (or think up an imaginary person)
- If the current outside temperature in Fahrenheit degrees is 24, what should you wear on a trip to downtown?

1.9 RStudio interface, second look

Projects: a good way to stay organized, button at the top right corner. A project is created as a folder on the local computer and will keep the scripts, datasets and workspaces related to your work projects separate. You can easily switch between projects, leave your work with one and return to it later, and find it just like you left it.

Scripts: it is better to use scripts than the console alone.

- You can run a line or a selection from the script window using the run button (top right of the script pane) or with Ctrl-Enter.
  - **Note!** it is easy to make your code disappear by accidentally pressing Enter instead of Ctrl-Enter. Luckily, Undo exists (in the Edit menu).
- You can also write comments: lines that start with `#` are not run

1.10 Activity 2

Try out the previous simple calculations using projects and scripts: Create a project, type one part in a script, run it, close the project, create another one, type another part, switch back and forth. See what happens!

2 R programming basics

2.1 Numeric vectors

- Numeric vectors are ordered sequences of numbers
- They are created with the function `c()`
  - Or as a result of some other function
- You can do calculations with them just like with single numbers, they happen element by element
  - Actually, single numbers **are** vectors, just of length 1!

- In R, vectors are *recycled* in calculations when needed, meaning that different length vectors can be mixed in calculations

Here are the numbers of cyclists passing the counter on Baana in Helsinki during two consequtive weeks 06-04-2015 to 12-04-2015 and 13-04-2015 to 19-04-2015, and some calculations with them:

```
week1 <- c(1133, 2097, 2211, 2592, 2370, 1926, 941)
week2 <- c(2811, 2670, 2879, 2543, 1845, 1227, 1128)
both <- week1 + week2
increase <- week2 - week1
```

- Most mathematical functions also work elementwise
  - `exp()`, `log()`, `abs()` and so on
- Some functions treat the vector as a whole
  - `length()`, `sum()`, `mean()`, `median()`, `sd()`, `var()`, `min()`, `max()`
- `summary()` tells several things about a vector at once
- A special symbol for a missing value: `NA`

2.1.1 Accessing vector elements

- It is possible to refer to elements of a vector by their index using square brackets
- The result is a new vector
- Many indices must be given as a vector!
- The colon is used to create index vectors easily

```
friday <- week1[5]
sum(week1[c(1, 4, 5)])
```

```
## [1] 6095
```

```
avgwk <- mean(week1[1:5])
avgwkend <- mean(week1[6:7])
```

2.2 Logical vectors

- Logical values are "truth" values: `TRUE` and `FALSE`
- Usually come up as a result of a comparison or a query
  - comparison operators: equal, less than etc
  - `== < > <= >= !=`
  - queries for type etc: `is.numeric() is.na()`
- You can calculate with these using logical operators 'and', 'or' and 'not': `& | !`

2.2.1 Accessing elements using logical vectors

- Using a logical vector instead of an index vector in the brackets will return those elements where the logical value is `TRUE`
- This is convenient when you want to do something with elements that satisfy a condition

```
(over2k <- week1 > 2000)
```

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

```
week1[over2k]
```

```
## [1] 2097 2211 2592 2370
```

```
week2[over2k]
```

```
## [1] 2670 2879 2543 1845
```

2.3 Character vectors

- Character vectors are like numeric vectors, except that the elements are character strings
- Must be enclosed in quotation marks, otherwise it looks like an object name
- `daynames <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday")`
- Usually show up as names for other objects, sometimes as actual data (but then they may actually be factors!)
- Comparisons are available (with respect to alphabetical order) but other calculations are not defined
  - Of course, string operations such as splitting and pasting exist (see functions `paste()` `strsplit()` `grep()` etc and alternatively the whole `stringr` package)

2.4 Activity 3, vectors

Concerning the numbers of the cyclists seen before...

- What was the proportion of weekend trips compared to the whole, in each week?
- How many days were above average in each week? How about above total average?
- Which day had the maximum number of cyclists in each week?
- Predict the third week's cyclist numbers using two different methods:
  - average of the two weeks
  - same pattern of change between weeks

2.5 Factors

Factor is a rather "R-specific" data structure used to denote group membership

- "At the front" like a character vector
- "Behind the scenes" like a numeric vector with values 1,2,3,...
- Several functions (for plotting and analyses) treat factors in a special way
- Different groups are called levels
  - `nlevels()` for the number of levels
  - `levels()` for a list of the levels

2.6 Data frames

- The most important data structure: where data is actually kept
- A table made out of columns that are vectors or factors of the same length
- In RStudio you can look at the contents by left clicking the name in the workspace listing (first 1000 rows)
- Other general ways to inspect data frames: `dim()`, `names()`, `summary()`, `head()`, `tail()`

There are some example data sets that come with the installation of R. To bring these to your workspace use a command like

```
data(iris)
```

where you replace the name `iris` with the data set name. Use just `data()` without any arguments to see a list of available data sets.

Note: there are also other data structures that are superficially the same as the "original", base R data frame, such as `DataFrame` in some Bioconductor packages and `data.table` from the package with the same name. You can always find out if you are working with a base R data frame with the function `is.data.frame`, or with the function `is()`.

2.7 Accessing data frame columns and elements

- Accessing single columns by name: `dfname$colname` or dfname[["colname"]]
  - result is a vector / factor
  - note the double brackets: very much a different thing than single bracket
  - also note the presence or absence of quotation marks
- Accssing parts by index: `dfname[1:10,1:10]`
  - result is a vector/factor or a data frame depending on the request: single columns are "dropped in dimension"
  - a missing index means "all": `dfname[,2]` means the whole second column
- Also available: `subset(df,condition,select)` for more human friendly subsetting (not recommended to be used in complicated scripts, because of unreliable evaluation), an alternatively, `filter(df,condition1,condition2)` from `dplyr`.

2.8 Activity 4

Work with the example data set `ToothGrowth`

- Bring the data set to the workspace using `data(ToothGrowth)`
- Get familiar with the data set using the functions mentioned earlier
  - Also look at its manual page, e.g. with command `?ToothGrowth`
- Are the columns numeric, character or factor? Should they be something else?
- Find the mean of tooth growth in each of the 6 subgroups defined by the two delivery methods and three doses
  - Note that there are plenty of ways to go about this even with your currently available tools, and then plenty more with more sophisticated approaches. All are acceptable!

3 Functions and data

3.1 Review about objects and expressions

- Expressions (often) create objects out of other objects
  - Such as data frames and vectors
- Expressions are combinations of calculations, function calls and accessing operations
- Function calls look like `functname(...)`
- Accessing operations look like `objname$varname` or `objname[...]` or `objname[[...]]`
- The result of an expression is shown and lost unless it is saved in the workspace by assignment

3.2 Understanding help pages

- Every function (and data set, etc.) has a manual page, possibly a shared page with other closely related functions
  - The manual page can be reached by typing `?functname` in console or using the search bar on RStudio Help tab
- The page always has the same structure with some key points
  - Usage and Parameters: these show what parameters the function can take, in what order and their default values
  - Value: what comes out as a result (if anything)
  - See also and Examples: often very helpful in learning how to use things. You can always run the example using `example(functname)` to run the whole example code at the end of `functname`'s manual page.

More on the parameter listing: In R, parameters in function calls can be either named or unnamed. Named parameters will be matched by the name so they can be given in any order. Unnamed parameter values will be matched by order. These styles can be mixed when it is unambiguous.

Note! In RStudio pressing the tab key when writing the function call will help by both completing parameter names and bringing up the parameter list.

3.3 Activity 5, help pages

Learn how to use the function `sample()` to create random numbers:

- Create a Lotto row (7 numbers out of 39)
- Simulate a sequence of 10 coin tosses (you need to decide symbols to use for the heads and tails)
- Select a random subsample of size 20 of the `iris` data set

3.4 Optional Activity 5, review of handling objects

- (Take the iris data set in to use by `data(iris)` if you don't have it in the workspace already)
- Save the first column (`Sepal.Length`) as a vector called `SL`
- Save the first 5 elements of `SL` as another vector, call it `SL5`
- Calculate the mean of `SL5`
- Now all at once, without the intermediate assignments: calculate the mean of the first 5 `Sepal.Length` values

3.5 Reading data in to a data frame

- The workhorse function: `read.table()` (and behind that `scan()`)
  - aliases: `read.csv()`, `read.csv2()`
- Also available in RStudio: Import dataset button
- Behind all of these is the idea that the data is in a text file, arranged in to rows and columns just like the resulting data frame
  - the values appear on rows separated by a certain character, such as a comma (csv stands for comma separated values), forming columns
  - usually there is a header row giving the names for the columns
  - sometimes there are row names too (which is a different thing than an extra column)
  - pay attention to the existence of quotation marks and the decimal separator!
- Most (?) other programs can export data as a csv file
- Other sources (e.g. SQL databases) are beyond the scope of this course

3.6 Adding variables to data frames

- By an assignment: for example `dfname$newvar <- dfname$oldvar1*dfname$oldvar2`
  - Just like in an assignment in the workspace, existing variables can be overridden without a warning
  - It is also possible to use the functions `with()` and `within()`
  - Yet another way, from `dplyr` is to use `dfname <- mutate(dfname, newvar=oldvar1*oldvar2)` which will keep all the existing variables, or `transmute` which will drop the unused ones
- Common needs for new variables include changes of measurement units, derived variables, recombination of factor levels, binning continuous values in to factors (the function `cut()` will do this last bit easily)

3.7 Activity 6, reading data

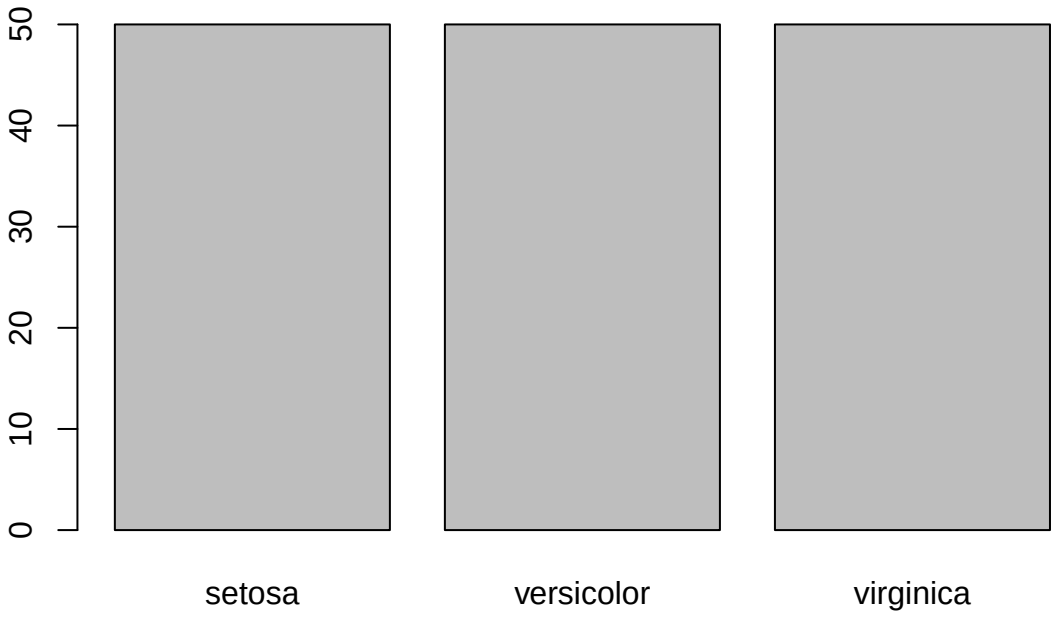
- Go to https://github.com/CSC-IT-Center-for-Science/data-stat-course/blob/master/datasets/weather-kumpula.csv (or just github.com, search for CSC IT center, click on Users results etc.)
- Download the file (you probably should right click on the "Raw" button and "Save target as" or similar, or left click it and "Save page as")
- Read it in to your RStudio session
- Explore the data in some (or all!) of the ways you now know

4 Basic data analysis

4.1 The `plot()` function

- `plot()` is a *generic* function: the result depends on what you use it on
  - Therefore the help page is unfortunately not very helpful
- One numeric vector: values against their index (not very common or useful)
- One factor: a barplot of frequencies (see also `barplot()` directly)

```
plot(iris$Species)
```



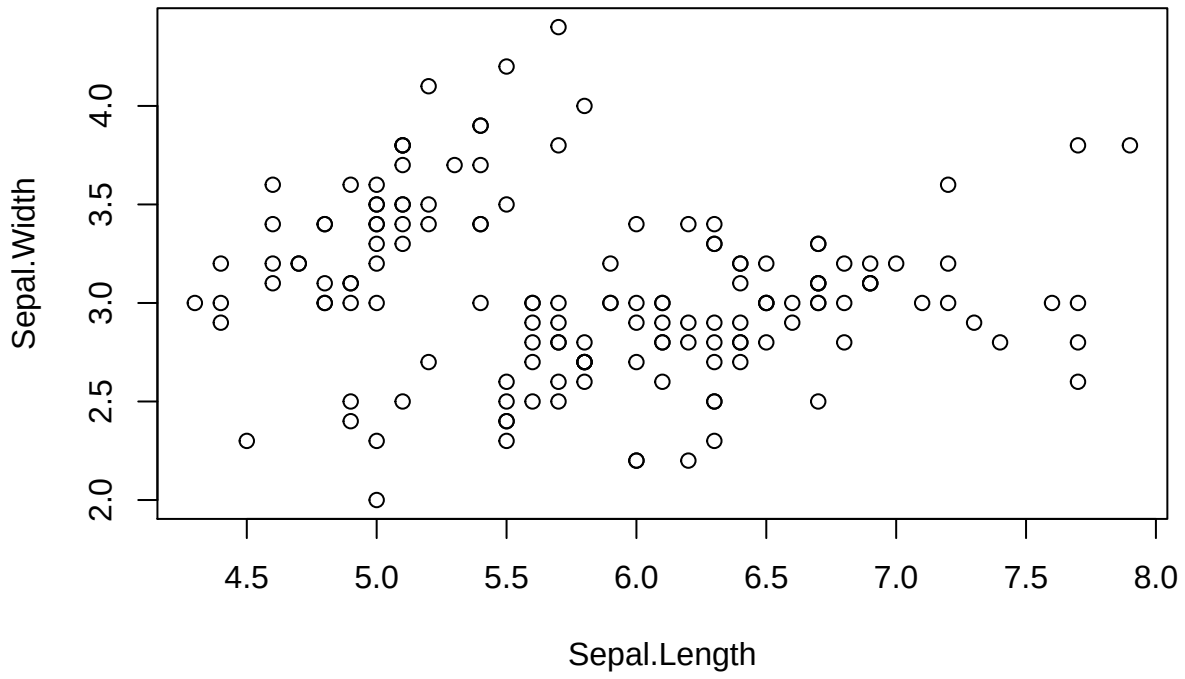
(not the smartest example, this one!)

- Two numeric vectors: a scatter plot
- A *formula:* depends. What is a formula anyway?

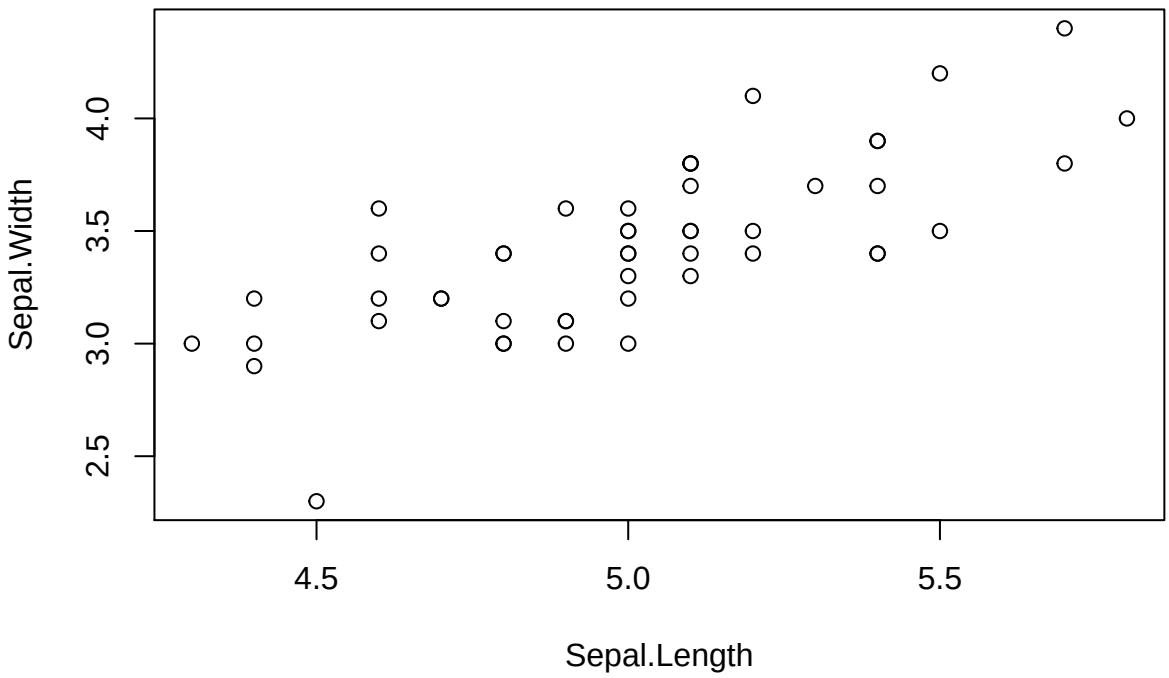
4.2 Formulas

- A formula is a description of dependence, or structure; "something by something else"
- Looks like this: `varname1 ~ varname2`
  - You can think of the tilde as the word "by"
  - There's a left hand side and a right hand side
- Usually comes with the possibility of naming the dataset the variables come from, and an optional subset rule, for example compare these:

```r
plot(Sepal.Width ~ Sepal.Length, iris)
```




```r
plot(Sepal.Width ~ Sepal.Length, iris, subset=Species=="setosa")
```



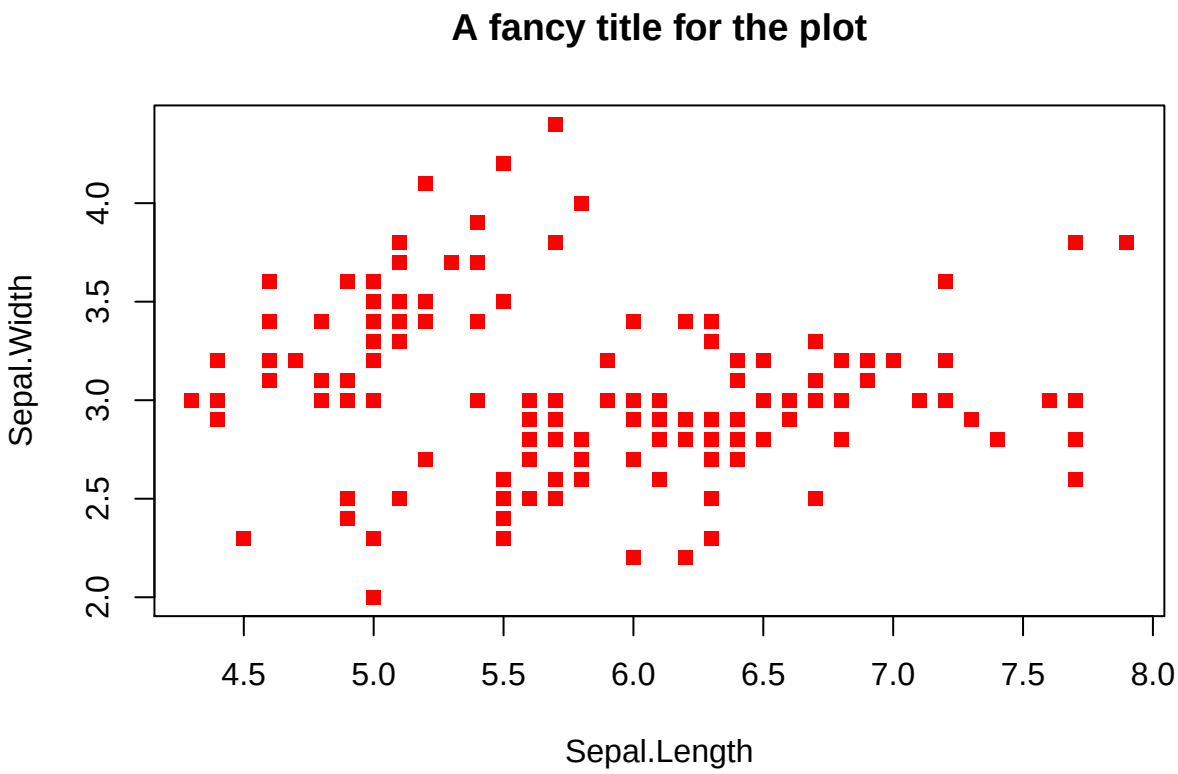
4.2.1 Plotting with formulas

- `plot(numeric~numeric)` will do a scatter plot, like those just above (the left hand side on y-axis)
- `plot(numeric~factor)` will do parallel boxplots by levels of the factor (see also `boxplot()` directly)
- `points()` and `lines()` will add points/lines on an existing plot
- . . . and then some

4.3 Graphical parameters

The listing of graphical parameters is given on the manual page of `par`. Some more are found on manual pages of `plot` and `plot.default`. Here is a shorter list of the more common ones. All of these can be used inside a plotting function, in this fashion:

```
plot(Sepal.Width~Sepal.Length,data=iris,col="red",pch=15,main="A fancy title for the plot")
```



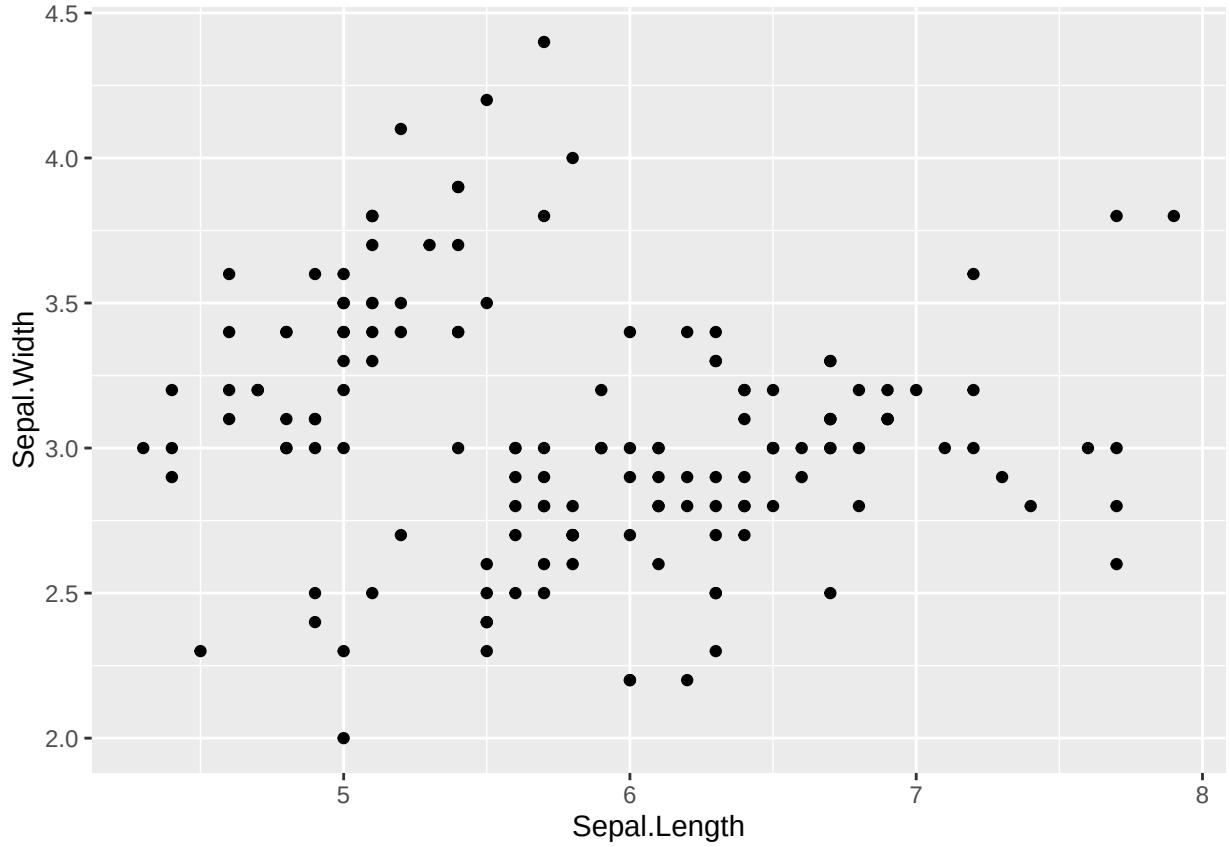
- `cex = 1.5`, size of the plot symbol relative to the usual 1.
- `col = "red"` or `col=5`, color of the symbol, line, etc. Can be given as a word (try `colors()` to see a list of named colors), as an index (referring to the current `palette()`), or rgb code... Can also be a vector of values in which case the values are used one at a time for the symbols/lines, and recycled if necessary. Allows for a quick way to plot factor levels with different colors.
- `lty="dashed"`, line type, one of "blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash"
- `lwd=1.5`, line width, similar to `cex`
- `pch=16`, plotting symbol, either an index (try `example(pch)` to see what they match) or a single character in quotes. Can be a vector.
- `type="l"`, one of p for points, l for lines, etc. and notably, n for no plotting which is useful for creating an empty "canvas" on which you can start to add stuff with e.g. `points()` and `lines()`, possibly level by level of a factor using different colors
- `xlab="label for x axis"`, similar for `ylab`
- `xlim=c(0,5)`, the limits for the x axis if you are not happy with the automatic choice, similar for `ylim`

4.4 The package ggplot2

The basic plotting provided by `plot()` and such considered above is the R's default option from the base package that comes with the basic installation and is therefore always available. Lately, an extra package called `ggplot2` has become another common choice and for good reasons. Some common things are far simpler to do with ggplot2 (and related packages) than with basic plotting. On the other hand, it uses a completely different philosophy (sort of similar to LaTeX in typesetting), a rather strange syntax and is in fact still partially a work in progress.

Here's a brief introduction on how to produce similar plots as were considered above. The package has to be installed for this to work.

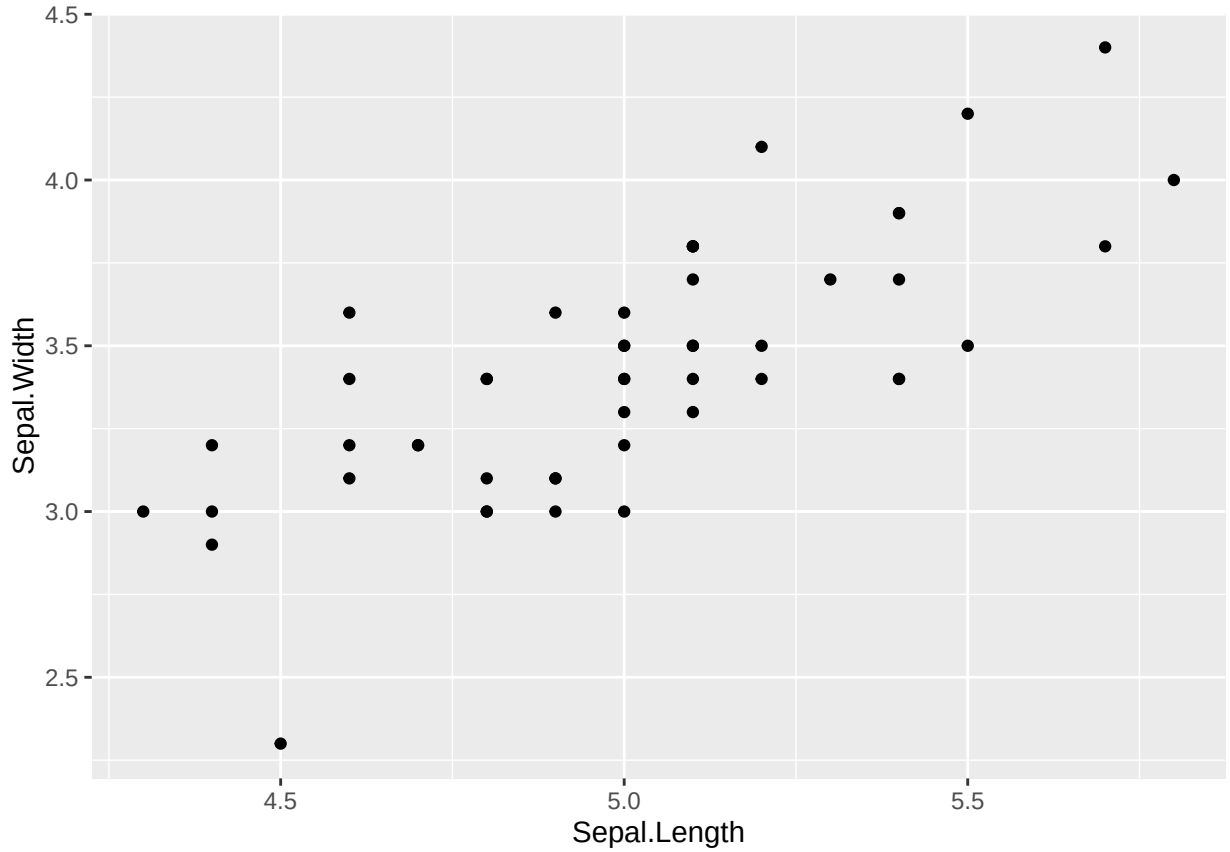
```
library(ggplot2) # so that you can actually use the installed package

ggplot(iris,aes(Sepal.Length,Sepal.Width))+geom_point()
```



Think of this as "I want to make a plot of the iris data set, using Sepal.Length and Sepal.Width, oh and also, there should be points."

Ggplot does not use a subset argument (*any more!*), so if you want to only use a subset of the data it has to be made explicitly, like here:

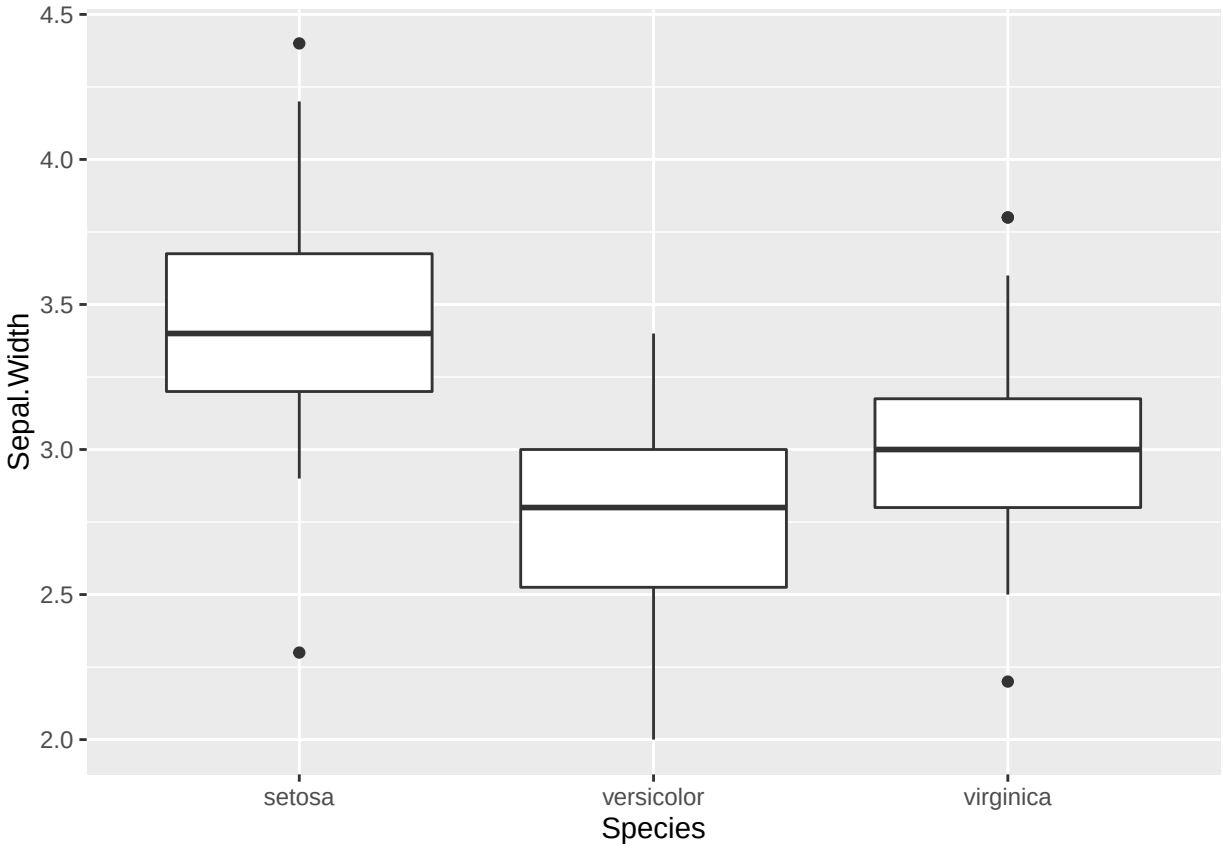
```
ggplot(subset(iris,Species=="setosa"),aes(Sepal.Length,Sepal.Width)) +
  geom_point()
```



(and of course, `filter` works as well.)

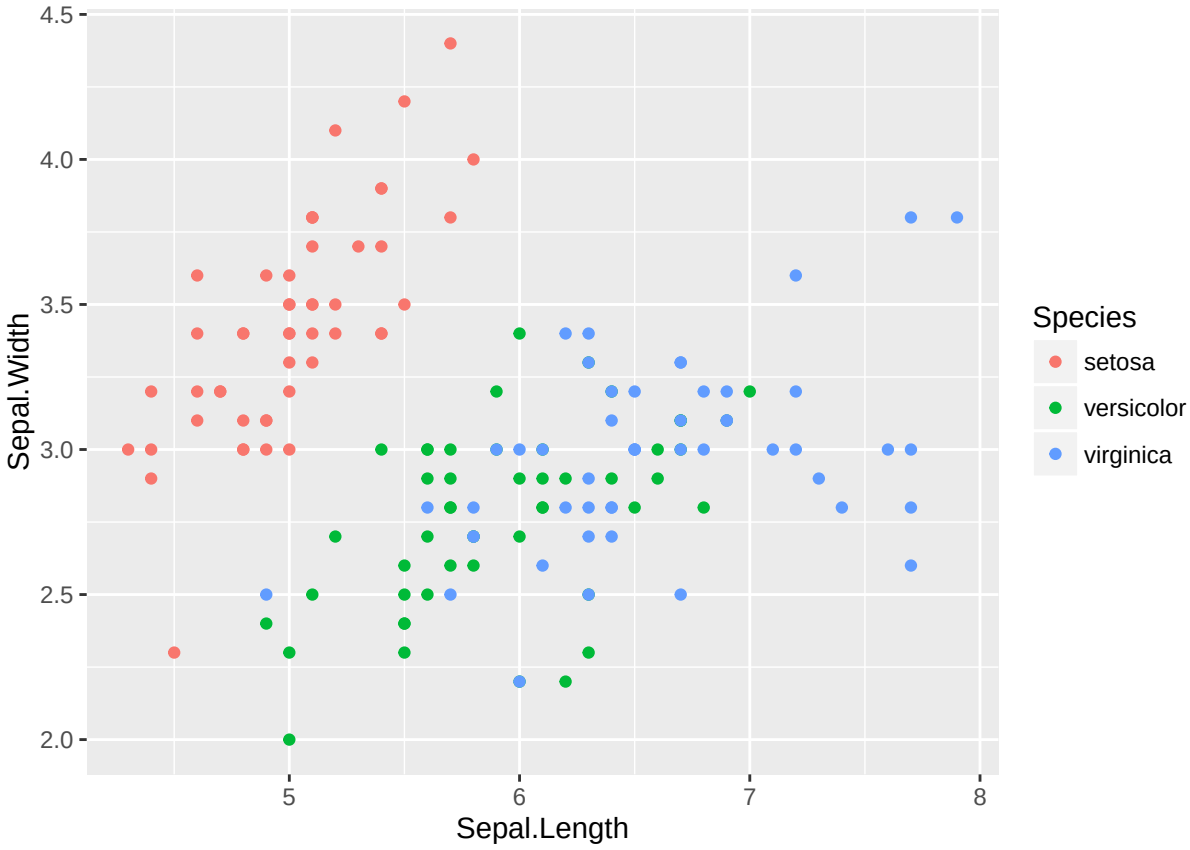
Boxplots by a factor (or a factor-like variable) happen by stating "oh and also, there should be boxplots":

```
ggplot(iris,aes(Species,Sepal.Width)) + geom_boxplot()
```



To create a scatter plot with the groups colored differently you add an extra wish about the aesthetics in the request, "there should be points that are colored by species":

```
ggplot(iris,aes(Sepal.Length,Sepal.Width)) + geom_point(aes(color=Species))
```



4.5 Activity 7, plotting

Try out plotting on the `iris` data set, the `ToothGrowth` or the weather data. For example, you could look at the relationship of air pressure and temperature, and if that is the same in summer and in winter.

4.6 Fitting linear models

- The function is called `lm()`
- You define the model as a formula
  - `response ~ predictor`
  - `response ~ predictor1 + predictor2`
  - Note that here the plus sign does not stand for summation but more like 'and': "response by this and this other thing"
- Note that whether the predictors are numeric vectors or factors now makes a big difference!
- Remember that the fitting function creates an object (a list this time)
  - Consider using `summary()` or `anova()` on the result

```
fit<-lm(Sepal.Width~Sepal.Length,data=iris)
summary(fit)
```

```
##
## Call:
```

```
## lm(formula = Sepal.Width ~ Sepal.Length, data = iris)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1095 -0.2454 -0.0167  0.2763  1.3338
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.41895    0.25356   13.48   <2e-16 ***
## Sepal.Length -0.06188    0.04297   -1.44    0.152
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4343 on 148 degrees of freedom
## Multiple R-squared:  0.01382,    Adjusted R-squared:  0.007159
## F-statistic: 2.074 on 1 and 148 DF,  p-value: 0.1519
```

```
fit2<-lm(Sepal.Width~Species,data=iris)
summary(fit2)
```

```
##
## Call:
## lm(formula = Sepal.Width ~ Species, data = iris)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -1.128 -0.228  0.026  0.226  0.972
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        3.42800    0.04804  71.359  < 2e-16 ***
## Speciesversicolor -0.65800    0.06794  -9.685  < 2e-16 ***
## Speciesvirginica  -0.45400    0.06794  -6.683 4.54e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3397 on 147 degrees of freedom
## Multiple R-squared:  0.4008, Adjusted R-squared:  0.3926
## F-statistic: 49.16 on 2 and 147 DF,  p-value: < 2.2e-16
```

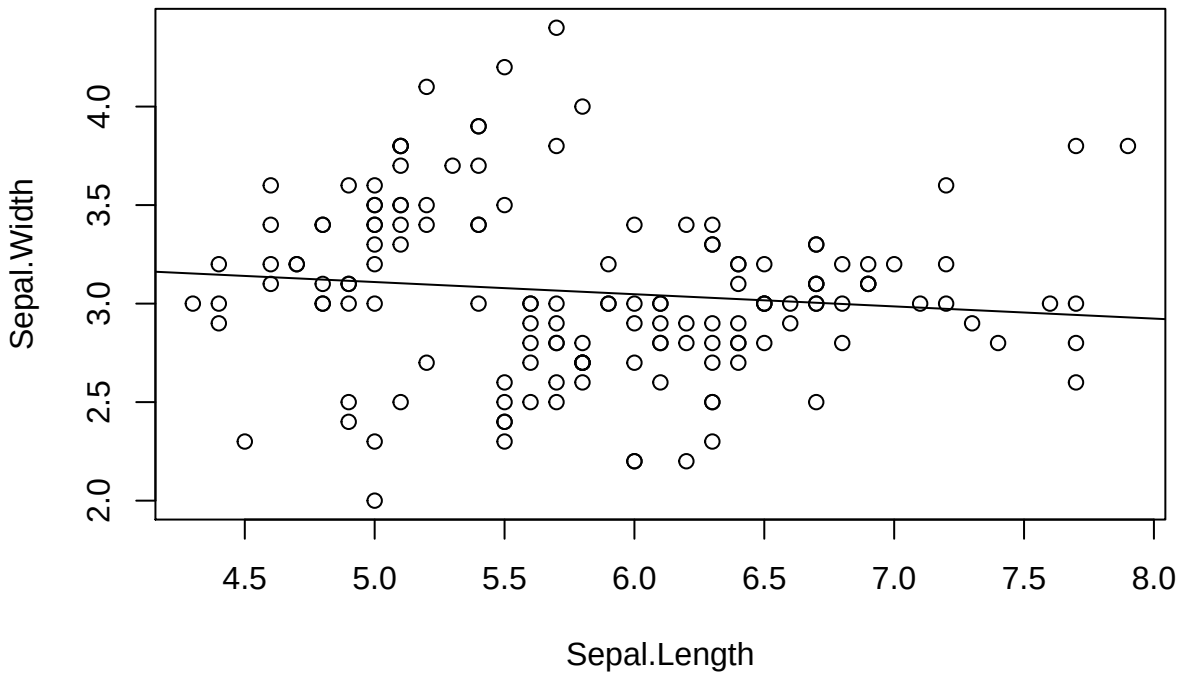
```
anova(fit2)
```

```
## Analysis of Variance Table
##
## Response: Sepal.Width
##            Df Sum Sq Mean Sq F value    Pr(>F)
## Species     2 11.345  5.6725   49.16 < 2.2e-16 ***
## Residuals 147 16.962  0.1154
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

4.6.1 `abline()`

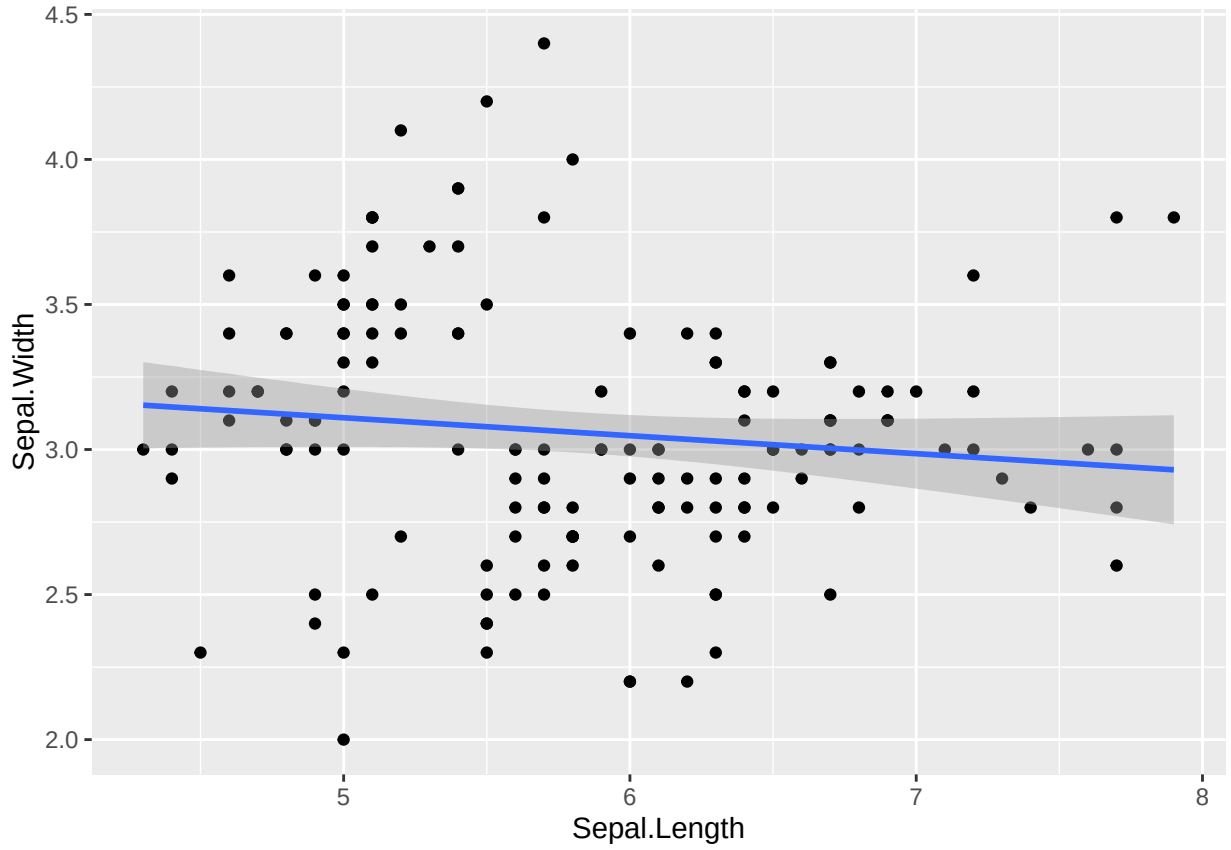
This function can be used to add straight lines on existing plots. Horizontal lines are drawn with the parameter-value pair such as `h=5`, where the value indicates at what y-value the line should be drawn. Vertical lines are drawn similarly with "v=5". In addition, `abline` can draw the regression line of a fitted linear model like this:

```
plot(Sepal.Width~Sepal.Length,data=iris)
abline(fit) #fit was created in the previous code chunk!
```



On the other hand, the same can be done using ggplot in a slightly different way:

```
ggplot(iris,aes(Sepal.Length,Sepal.Width))+geom_point()+stat_smooth(method="lm")
```



4.7 Testing hypotheses

- Difficult to give a general introduction as there are so many different tests
- However, one way or another you have to give the data to the testing function, maybe choose some other parameters
- The result is again a list, with some pretty printing
- T-test is performed by providing two vectors or by a formula with a two-level factor on the right hand side

```
t.test(Sepal.Width~Species, data=iris, subset=Species!="setosa")
```

```
##
##  Welch Two Sample t-test
##
## data:  Sepal.Width by Species
## t = -3.2058, df = 97.927, p-value = 0.001819
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.33028364 -0.07771636
## sample estimates:
## mean in group versicolor  mean in group virginica
##                    2.770                    2.974
```

4.8 Activity 8

Return again to any of the data sets seen, or try a new one: you can see a list of available data sets using `data()`. Use any and all of the techniques you have learned so far, or search for new! Think of a model to fit. Think of a hypothesis to test! If you can't think of anything, ask for a suggestion, or go see what someone else is doing and help them.

5 What next?

By now you should have a pretty good understanding of how to work with R and RStudio. But in this short introduction there was a lot of things left unsaid. For example, most of the more complicated and interesting methods relevant to your research are probably implemented in one of the thousands of packages available on CRAN or on Bioconductor or on github. You still probably need to put in some effort to learn to use them specifically. The following two points hold for all packages though:

- The packages need to be installed (once) on your own computer before you can use them. A small collection comes with the basic installation of R. You can see those on RStudio's Packages-pane.
- When you want to use a package you need to load it (each time you start a new session). A very small collection is automatically loaded. The loading command is `library(libname)` but the checkbox on RStudio Packages-pane does the same thing.

And here is a list of things you could learn still, in no particular order:

- Writing your own functions
- Efficient ways of doing something repeatedly or groupwise (apply-family, looping)
- Transformations on data frames (merge, reshape)
- Related to the previous two points: dplyr package
- Random number generation and distributions in general
- Matrices
- Parallel computing and how to use R on CSC's supercluster Taito
- Creating Shiny applications
- Creating R Markdown documents (such as this!)
- Other, suggestions?