



Fødevaredatanalyse

## Week 0

# Getting prepared for FDA

Assembled by Berit Østeby    [berit.oesteby@food.ku.dk](mailto:berit.oesteby@food.ku.dk)

#### Data sets used

dates.xlsx  
Mouse\_diet\_intervention.xlsx  
NewNordicDiet.xlsx  
Results Consumer Test.xlsx  
Results Panel.xlsx  
Wine.xlsx

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>RStudio Overview</b>	<b>3</b>
<b>3</b>	<b>Directory, packages and libraries</b>	<b>4</b>
3.1	Working Directory . . . . .	4
3.1.1	Checking the contents of your chosen working directory . . . . .	4
3.1.2	Associated error messages . . . . .	5
3.2	Packages, libraries and functions . . . . .	6
3.3	Associated error messages . . . . .	8
<b>4</b>	<b>Data set overview and assessment</b>	<b>9</b>
4.1	Data types and structures . . . . .	9
4.2	> <u>NA</u> (not available); Missing values in the dataset . . . . .	10
4.3	> <u>Error: Unexpected symbol in "..."</u> Dealing with invalid variable names . . . . .	12
4.3.1	Validity of variable names . . . . .	12
4.3.2	Associated error messages . . . . .	13
4.3.3	Renaming one (or a few) variable name(s) . . . . .	13
4.3.4	Substituting all spaces in, and adding prefixes to, all variable names . . . . .	14
<b>5</b>	<b>R notation</b>	<b>15</b>
5.1	Extracting a value or a set of values from a data frame . . . . .	15
5.1.1	Using the subset command . . . . .	16
<b>6</b>	<b>Introduction to selected commands</b>	<b>18</b>
6.1	Aggregate . . . . .	18
6.2	qplot . . . . .	19
6.2.1	Data types . . . . .	22
<b>7</b>	<b>Helpful sites</b>	<b>24</b>
<b>8</b>	<b>Exercises</b>	<b>25</b>
8.1	Getting started, nice and slow. . . . .	25
8.2	Coffee . . . . .	25
<b>9</b>	<b>Debugging the different week sets</b>	<b>26</b>
9.1	Week 1: Importing files, and calling on objects and functions . . . . .	26
9.1.1	Importing a data set, debugging function calls . . . . .	26
9.2	Debugging - Invalid variable names, PCA . . . . .	27
9.3	Debugging week 3 - NA's and R object type classes . . . . .	28
9.4	Debugging week 4 - R object type classes . . . . .	28

# 1 Introduction

The aims of this background and exercise set are

- that you get familiarised with RStudio, which will be used in the course, through exercises
- that you learn some tips which should optimise the learning of the program.
- that you learn to read some of the most common error messages, and how to solve them
- that you get used to using Google as a tool
  - to find solutions for error messages
  - as a source of inspiration if there is a certain thing you would like to do with your data set.
- that you get a basic knowledge of how to make a range of basic plotting functions in RStudio, and how you optimise the plots.
- that you get familiarised with some of the data sets that will be used later in the course

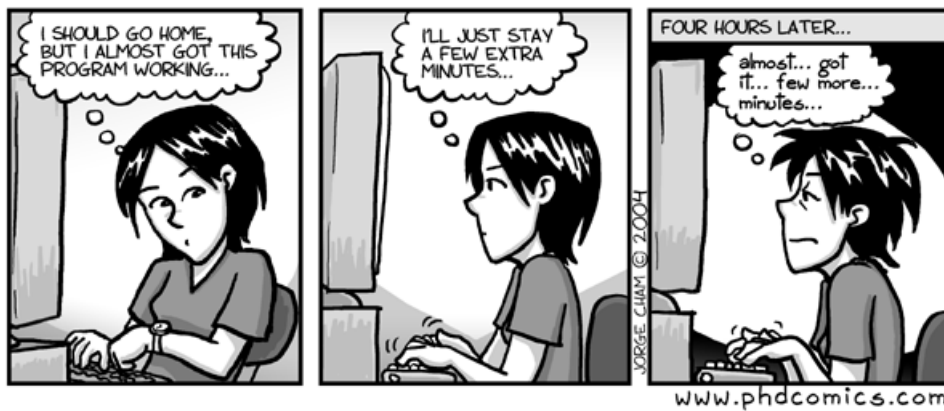


Figure 1: Doing the exercises in this set prior to the course start might well reduce the occurrence of sleep deprivation during the course weeks.

## 2 RStudio Overview

RStudio can be downloaded for free from this link: <https://www.rstudio.com/products/rstudio/download/>. Figure 2 shows the opening screen of RStudio, and the name and usages of the different frames. This figure will be referred to when describing the different frames in this exercise set, so take a minute to learn their names.

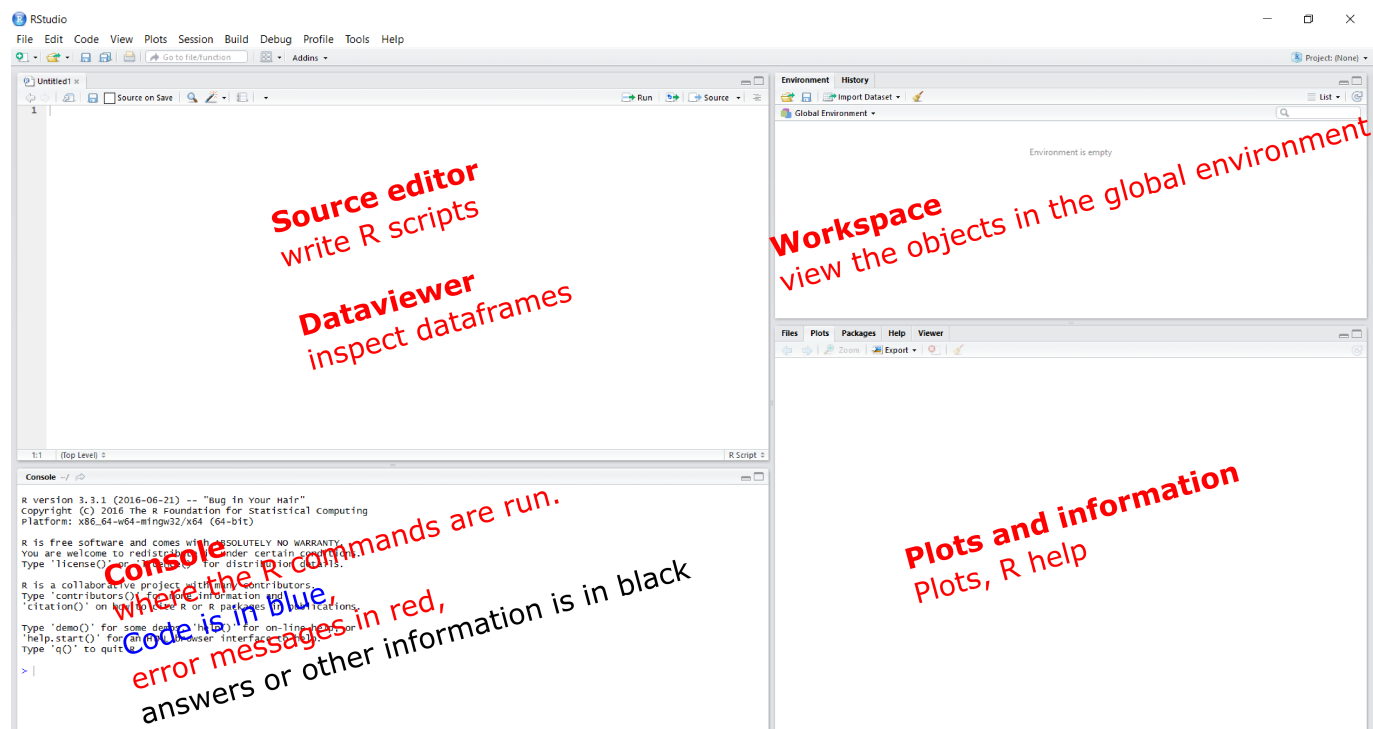


Figure 2: RStudio overview

## 3 Directory, packages and libraries

### 3.1 Working Directory

The working directory is the space from where R locates saved R scripts and data sets.



#### Working Directory analogy 🏠

A given working directory can be represented by a specific room in a house. You store your files in a specific room (for example, in the living room), and for R to find a given file, you need to tell it in which room it should look for it in. So, if you have stored a data set in the living room, you need to tell R that it should look in the living room. R will not find the file you put in the living room, in the bath room.

For this course, it is recommended that you create folders for the different weeks, in which you save your exercises and downloaded data sets. Every time you open RStudio, you need to define its relevant working directory. This can be done in several ways:

- Go to **Session** → **Set working directory** → **Choose directory...** (alternatively use the shortcuts `ctrl+shift+h` for this option), and choose the relevant folder. Now, the pathway for your folder will be shown in the console, like this:

```
> setwd("C:/Documents/FDA")
```

- If you are going to revisit a script later on, you can paste in the pathway code for your file into the script. The code can be copied from the console, if you do the option above:

So, in the start of your script copying the location as shown in Figure 3 will save you some time:

Source editor:

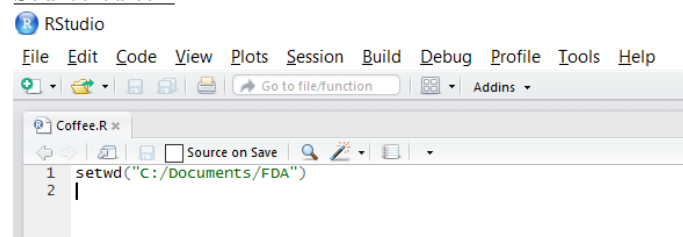


Figure 3: Command to save in the script to save time if a script is to be worked on more than one time.

#### 3.1.1 Checking the contents of your chosen working directory

The contents of your chosen working directory can be checked with the following command:

Source editor:

Example of console output:

```
list.files()
```

```
> list.files()
[1] "1_2.R"          "1_4.R"          "1_5.R"          "1_5_tutorial.R"
[5] "1_6.R"          "1_7.R"          "1_7_ny.R"       "1_8.R"
[9] "FDA_uge1.pdf"   "McDonaldsScaled.xlsx" "Results Consumer Test.xlsx" "Results Panel.xlsx"
[13] "wine.xlsx"
```

This can be useful if you want to

- check whether you're in the folder you thought you were
- check whether you have stored a data set in the right folder
- check the exact spelling of files in the folder.

### 3.1.2 Associated error messages

```
> Coffee <- read_excel("Results Consumer Test.xlsx")
```

Error: 'file.xlsx' does not exist in current working directory ('C:/Current working directory').

This error message says that the file you are asking for, can not be found in the working directory that you have defined. It can be caused by several reasons;

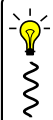
- being in the wrong working directory



#### Working Directory analogy 🏠

Ooops, your files are in the living room, but you have actually asked RStudio to work from the files present in the bedroom. Solution: ask RStudio to look in the living room instead.

- the relevant file has accidentally been stored in a different directory than the working directory



#### Working Directory analogy 🏠

You thought you stored your coffee-file in the kitchen, but accidentally, you stored it in the bathroom. Solution: move your coffee-file from the bathroom to the kitchen, and try again.

- misspelled name of the file which is called for. R is very sensitive to spelling. Just get used to it.

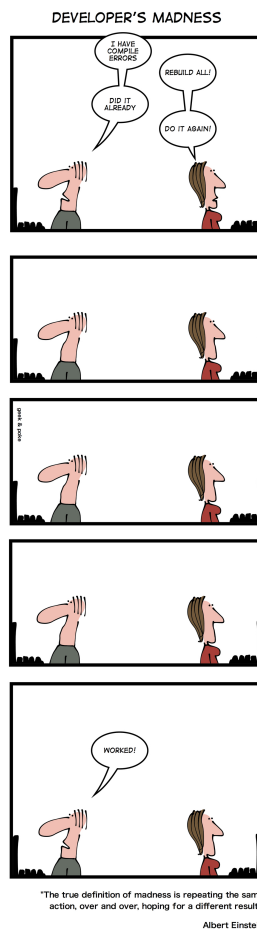


Figure 4: When getting compilation errors, try first to read and understand the error message - and if you don't know what to do in order to fix the error, Googling the error message may be a very good and time-saving idea.

## 3.2 Packages, libraries and functions

Packages are collections of functions, and necessary for many of the operations that will be made with R. Packages are installed through the following command

```
install.packages("package")
```

More packages can be installed at the same time, using the following command:

```
install.packages(c("package1", "package2"))
```

c is used for combining in several commands.

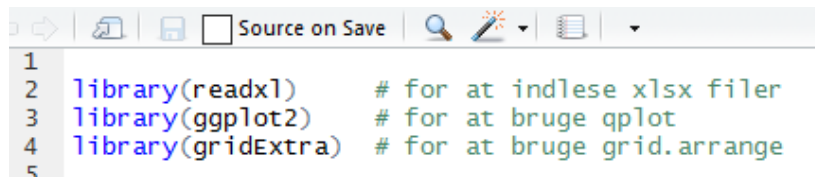
You normally only have to install a package once, i.e. you don't need to do it every time you open RStudio.

### Using the content of the package

In order to be able to use a function from a package, the relevant package needs to get loaded through the `library` command:

```
library(package)
```

This needs to be done every time you restart RStudio. It can therefore be useful to write the libraries you need in a given exercise, in the script itself. This would make the top of a script look something like in Figure 5.



The screenshot shows the top of an R script in the RStudio editor. The script contains three lines of code, each preceded by a line number (1, 2, 3). The code is as follows:

```
1  
2 library(readxl)      # for at indlese xlsx filer  
3 library(ggplot2)     # for at bruge qplot  
4 library(gridExtra)   # for at bruge grid.arrange  
5
```

Figure 5: Example of how library calls can be collected at the top of your script.

### Package installment analogy

Installing packages is similar to buy a toolbox. Once you've bought it (installed in your garage) it is there, and ready to be used, but every time you want to fix something broken or adjust something, you need to choose the right tool, as your toolbox hopefully is not limited to one screwdriver. For fixing a given challenge, you choose what fits for the job, being either a screwdriver, ruler or hammer.

In RStudio, the `library` command is used similarly, to tell RStudio the parts of the package needed for programming.



Figure 6: You can comment out stuff by putting a hash tag in front of it, as shown in Figure 5. This can be useful, as it will make it easier for you to understand what you did and why you did it when you revisit an R script you made earlier.



### 3.3 Associated error messages

We will read in some files with the `read_excel` function from the `readxl` package to exemplify the use of packages and some common error messages:

Here, the function runs as it should:

```
> Coffee <- read_excel("Results Consumer Test.xlsx")
>
```

- If you have forgotten to load the right package (as an analogy; you are asking RStudio to hang up a picture on the wall with a nail, but you haven't told RStudio to find a hammer in the toolbox):

```
> Coffee <- read_excel("Results Consumer Test.xlsx")

Error: could not find function "read_excel"
```

Solution: get the right package through library (alternatively also by loading a package)

```
> library(readxl)
> Coffee <- read_excel("Results Consumer Test.xlsx")
>
```

- The function is misspelled:

```
> Coffee <- read_excl("Results Consumer Test.xlsx")

Error: could not find function "read_excl"

> Coffee <- read_Excel("Results Consumer Test.xlsx")

Error: could not find function "read_Excel"
```

Solution: Write the function correctly (remember to distinguish between small and capital letters)

```
> Coffee <- read_excel("Results Consumer Test.xlsx")
>
```

#### Error message analogy

you want RStudio to find a hammer, but RStudio is a nazi on correct spelling, so it won't understand you if you ask it to find a Hammer, hemmer, hmmmer etc.

Solution: Spell it out right.

- Every now and then, functions in a package change, giving the following error message:

```
Warning message:
'functionX' is deprecated
```

Alternatives to the deprecated function can often be easily found, often in the error message, but if not they can be found on Google.

#### Error message analogy

This problem can be compared to your partner having made changes in your toolbox, without telling you. You \*thought\* you had a hammer, but no - your partner used it to fix your swimming pool and lost it. It might be a different hammer in one of your other toolboxes (the more expensive ones, that you have hidden away from your partner, because he does not know how to treat the tools with the respect they deserve), so you'll need to tell RStudio to look in a different toolbox. It might also be possible to tweak tools in your existing toolbox to get RStudio to do the job; i.e. get RStudio to use a ruler as a hammer. As RStudio is free, it has a lot of users, and the chance that any other of these users have bumped into similar problems is quite high.

## 4 Data set overview and assessment

### 4.1 Data types and structures

These are good to know as R sometimes will treat your data differently, depending on whether it is stored as characters or numeric. You can convert your data set from one type to another, but you should be careful in doing so. R has 6 vector types

- character: "a"
- numeric: 2, 15.5
- integer: 2L
- logical: true, false
- complex:  $i + 4i$

The vector type can be assessed by the following commands

- `class()`
- `typeof()`
- `length()`

The main data structures are

- vector: a collection of elements most commonly of mode character, logical, integer, or numeric  
The structures can be assessed by the following commands

- `typeof()`
- `length()`
- `class()`
- `str()`

- list
- matrix is an atomic vector with an extra dimension.

They can be constructed in a number of ways;

- matrices are filled column-wise by default:
- use the `byrow` argument
- bind columns and/or rows

- data frame: is a special type of list, where every element of the list has the same length.

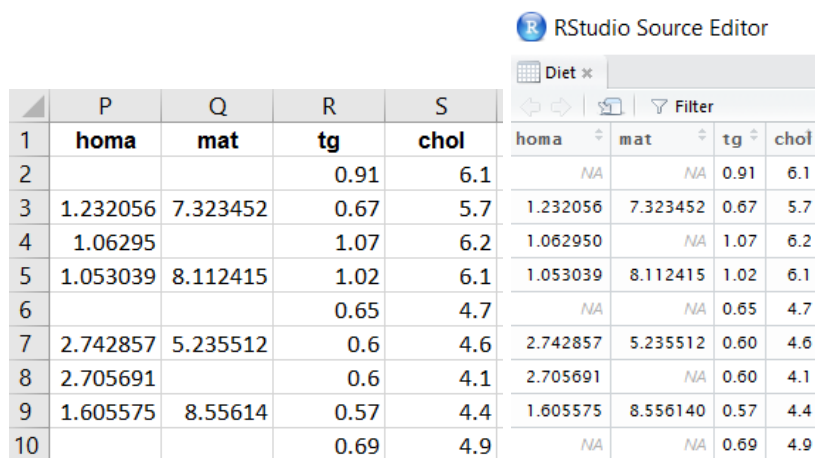
- `nrow()` #number of rows
- `ncol()` #number of columns
- `head()` #shows first 6 rows
- `str()` #structure of data frame

## 4.2 > NA (not available); Missing values in the dataset

`View(mydata)` gives an intuitive idea of differing lengths or missing values, as seen in Figure 7. (Instead of writing the `View` command, you can see the data frame by clicking on the file read in, in the Workspace.)

It happens that data sets have missing values for given variables. While these are seen in excel files as empty cells, R reads them as missing values in a matrix (Figure 7).

The method to exclude numbers, depends on how you are going to use the data.



	P	Q	R	S
	homa	mat	tg	chol
1				
2			0.91	6.1
3	1.232056	7.323452	0.67	5.7
4	1.06295		1.07	6.2
5	1.053039	8.112415	1.02	6.1
6			0.65	4.7
7	2.742857	5.235512	0.6	4.6
8	2.705691		0.6	4.1
9	1.605575	8.55614	0.57	4.4
10			0.69	4.9

Figure 7: How missing values are displayed in a spreadsheet vs. with `View` in R. (Data set: NewNordicDiet.xlsx)

- Localising NA's

The command `colSums(is.na(Dataset))` adds up all the NA's in the different columns.

Source editor:

```
library(readxl)

Diet <- read_excel("NewNordicDiet.xlsx")
colSums(is.na(Diet))
```

Console output:

```
homa    mat    tg    chol    hdl
148    300    1      1      1
ldl    vldl  energyday  proEpro  fatpro
1      1    441    441    441
```

- Exclude all rows with any missing values

If you only want to use complete rows, the rows including NA can be excluded from your data set by the use of `complete.cases`.

Source editor:

```
library(readxl)

Diet <- read_excel("NewNordicDiet.xlsx")
nrow(Diet)

Diet_clean <- Diet[complete.cases(Diet),]
nrow(Diet_clean)
```

Console output:

```
> nrow(Diet)
[1] 588

> Diet_clean <- Diet[complete.cases(Diet),]
> nrow(Diet)
[1] 139
```

Only

using complete cases dramatically reduces the amount of rows from 588 to 139. This solution is great for vectors, but less good for matrices, as it removes a whole row, even if only one variable measurement is missing.

- Calculating means/median etc. within a column

We have a column of values, but we are missing the values for some of the measurements. When trying to calculate the mean of these numbers, R replies with a missing (NA) value, telling us that our data is containing a missing value.

The command `sum(is.na(dataframe$column))` gives the number of NA values in the given column. By using the argument `na.rm=TRUE`, we tell R to ignore any missing values in the column.

Source editor:

```
mean(Diet$bm)

sum(is.na(Diet$bm))
mean(Diet$bm, na.rm=TRUE)
```

Console output

```
> mean(Diet$bm)

[1] NA

> sum(is.na(Diet$bm))

[1] 150

> mean(Diet$bm, na.rm=TRUE)

[1] 2792.456
```

by using the argument `na.rm=TRUE`, R ignores any cells containing missing values.

- 'Invisible' missing values

It might be hard to see in an excel sheet whether there are any additional, "active", rows of missing values, if they are placed at the end.

	E	F	G	H	I		Fat_Protein	cage_no	bw_w_minus1	bw_w0	bw_w1
1	Fat_Protein	cage_no	bw_w_minus1	bw_w0	bw_w1						
37	HF whey	8	14	17	16.7		HF whey	8	14.0	17.0	16.7
38	HF whey	8	10.7	14.7	14.6		HF whey	8	10.7	14.7	14.6
39	HF whey	8	13.2	16.4	16.1		HF whey	8	13.2	16.4	16.1
40	HF whey	8	11.3	15.3	16.2		HF whey	8	11.3	15.3	16.2
41	HF whey	8	11.4	16.3	16.5		HF whey	8	11.4	16.3	16.5
42							NA	NA	NA	NA	NA
43							NA	NA	NA	NA	NA
44							NA	NA	NA	NA	NA
45							NA	NA	NA	NA	NA
46							NA	NA	NA	NA	NA

Figure 8: The left picture shows the original spreadsheet, while the picture to the right is a screen shot from RStudio (Data set: Mouse\_diet\_intervention.xlsx).

The solution here is to make a subset of your data set, by extracting the rows that you are using from the original data set:

```
Mouse <- read_excel('Mouse_diet_intervention.xlsx')
View(Mouse)

# Extracting the 40 rows of the dataset containing values
MouseN <- Mouse[c(1:40),]
```

What we fill inside the square brackets is what we want R to extract from our data set. Everything in front of the comma, relates to the rows, while everything after the comma, relates to our columns. Here we are asking R whether it would like to extract all rows from row no. 1 to row no. 40, and as the space after the comma is empty, we ask R to include all columns. Read more about extracting rows and columns in Section 5.

## 4.3 > Error: Unexpected symbol in "..."

### Dealing with invalid variable names

#### 4.3.1 Validity of variable names

##### Valid variable names

- start with a letter or a dot
- do not contain any spaces

##### Invalid variable names

- start with a number
- contain spaces
- contain math operators such as /, which occasionally will be interpreted as math and not characters

So, how do you know, whether your data set contain invalid variable names?

The author of this set has not found a straight forward way to do this - so here are two ways you can recognise it:

- look through the variable names using the command `names(mydata)` (Figure 9)
- wait until you run into one of the error messages shown below.

If the data set contains invalid variable names, as shown in the column names in Figure 9, RStudio might complain.

	Wine labels	class	Country	Ethyl acetate	2/3-Methylbutanal	Ethanol
1	ARG-BNS1	1	Argentina	8.146505	0.004797360	0.011651037
2	ARG-DDA1	1	Argentina	9.511960	0.001647777	0.139456345
3	ARG-FFL1	1	Argentina	7.975901	0.003802121	0.005393887
4	ARG-FLM1	1	Argentina	7.800659	0.001921981	0.039641227
5	ARG-ICR1	1	Argentina	7.947105	0.001912794	0.099581718
6	ARG-SAL1	1	Argentina	8.714034	0.002505292	0.043381915

```

> names(wine)
[1] "wine labels"
[3] "Country"
[5] "2/3-Methylbutanal"
[7] "Ethyl propanoate"
[9] "Propyl acetate"
[11] "Ethyl butyrate"
[13] "2,3-pentanedione"
[15] "2-Butanol"
[17] "Ethyl pentanoate"
[19] "Ethyl 2-butenate"
[21] "3-Methyl -1-butanol"
[23] "3-Methyl-3-buten-1-ol"
[25] "Ethyl pyruvate"
[27] "Hexyl acetate"
[29] "3-methylcyclopenten-1-carboxylic acid (uncertain id.)"
[31] "Ethyl heptanoate"
[33] "1-Hexanol"
[35] "Nonanal"
[37] "Ethyl octanoate"
[39] "1-octen-3-ol"
[41] "Decanal"
[43] "Vitispirane"
[45] "Ethyl 2-hydroxycaproate"
[47] "2-Pentanol (uncertain id.)"
[49] "Ethyl decanoate"
[51] "Butanoic acid, 3-methyl- (uncertain id.)"
[53] "Ethyl 9-decenoate"
[55] "Methyl 2-hydroxybenzoate"
[57] "Hexanoic acid"
[59] "Ethyl 3-methylbutyl butanedioate"

```

```

"class"
"Ethyl acetate"
"Ethanol"
"Ethyl 2-methyl-propanoate"
"2-Methylpropyl acetate"
"Ethyl 2-methylbutyrate"
"Ethyl 3-methylbutyrate"
"3-Methylbutyl acetate"
"1-Butanol"
"Amylpropionate"
"Ethyl hexanoate"
"1-Pentanol"
"2,3-Dihydrofuran (uncertain id.)"
"3-Hydroxy-2-butanone"
"3-Methyl-1-pentanol"
"Ethyl lactate"
"3-Hexenol"
"2-Butoxy- ethanol"
"Acetic acid"
"Furfural"
"Benzaldehyde"
"Propanoic acid"
"1-Octanol"
"Butanoic acid"
"3-Methylbutyl octanoate"
"Diethyl succinate"
"Calacorene (uncertain id.)"
"2-Phenylethyl acetate"
"Benzyl alcohol"

```

Figure 9: View or clicking the read in data set shows invalid variable names. This data set contains loads of invalid variable names.

### 4.3.2 Associated error messages

We try to make a `qplot`, but get an error message

Source editor:

```
Wine <- read_excel('Wine.xlsx')
View(Wine)

p1 <- qplot(data=Wine, Furfural, Ethanol)
p2 <- qplot(data=Wine, Ethyl acetate, ethanol)
```

Console output:

```
> p1 <- qplot(data=Wine, Furfural, Ethanol)
> p2 <- qplot(data=Wine, Ethyl acetate, ethanol)
Error: Unexpected symbol in "p2 <- qplot(data=Wine, Ethyl acetate"
```

There are two solutions to this:

1. changing your chosen syntax;  
preferred when you want to present the column names in your plots (See Section 5.1).
2. changing the invalid names to valid names  
can be used when the grammatical correctness of the column names are off less importance.

### 4.3.3 Renaming one (or a few) variable name(s)

Starting on an exercise asking you to code with one of the variables with spaces in it, it might seem easiest to change the single variables that you need in excel, and save it as a new. And this works perfectly fine. You can also change variable names in R as shown below.

```
#Changing the name of the variable in column 4
names(Wine)[4] <- 'Ethyl.2.methyl.propanoate'
```

#### 4.3.4 Substituting all spaces in, and adding prefixes to, all variable names

When you realise that Morten keeps asking for new variables to compare, several of them containing spaces ..you'll look for a way to efficiently remove or replace all the spaces in the variable names. We can use the command `gsub`.

```
# Substituting all spaces in variable names with a dot
names(Wine) <- gsub(" ", ".", names(Wine))
```

A prefix can also be added to all the variable names, to avoid R complaining about names starting the wrong way.

```
# Defining all variable names to start with a_
colnames(Wine) <- paste("a", colnames(Wine[,]), sep = "_")
```

	Wine labels	class	Country	Ethyl acetate	2/3-Methylbutanal	Ethanol
1	ARG-BNS1	1	Argentina	8.146505	0.00479736	0.01165104
2	ARG-DDA1	1	Argentina	9.51196	0.001647777	0.1394563
3	ARG-FFL1	1	Argentina	7.975901	0.003802121	0.005393887

	Wine.labels	class	Country	Ethyl.acetate	2/3-Methylbutanal	Ethanol
1	ARG-BNS1	1	Argentina	8.146505	0.00479736	0.01165104
2	ARG-DDA1	1	Argentina	9.51196	0.001647777	0.1394563
3	ARG-FFL1	1	Argentina	7.975901	0.003802121	0.005393887

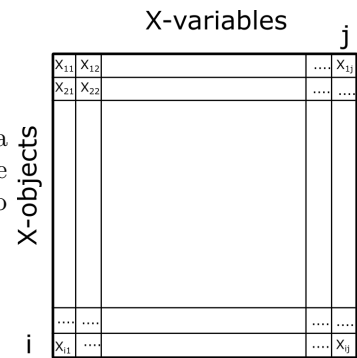
	a_Wine.labels	a_class	a_Country	a_Ethyl.acetate	a_2/3-Methylbutanal	a_Ethanol
1	ARG-BNS1	1	Argentina	8.146505	0.00479736	0.01165104
2	ARG-DDA1	1	Argentina	9.51196	0.001647777	0.1394563
3	ARG-FFL1	1	Argentina	7.975901	0.003802121	0.005393887

Figure 10: Top figure shows the original variable names. In the middle, the spaces in variable names are replaced with dots. At the bottom, a prefix has been added to all the variable names

## 5 R notation

### 5.1 Extracting a value or a set of values from a data frame

Square brackets;  $df[i, j]$  are used to extract a value or set of values from a data frame, this works by writing the data frame's name followed by a pair of square brackets. The indexes tell R which values to return. R will use the first index to subset the rows of the data frame and the second index to subset the columns.



The advantages of sub setting a data set are

- to extract only the rows or columns relevant for you, reducing your data set in size and making it easier to work with
- to make it possible to do calculations, due to difference in lengths in between different parameters

Subsetting can be done in a few ways, and we'll show that with a real data set:

	Wine labels	class	Country	Ethyl acetate	2/3-Methylbutanal	Ethanol	Ethyl propanoate	Ethyl 2-methyl-propanoate	Propyl acetate	2-Methylpropyl acetate,	Ethyl butyrate	Ethyl 2-methylbutyrate	2-Methylbutyl acetate
1	ARG-BNS1	1	Argentina	8.146505	0.004797360	0.011651037	0.1744942	0.10555782	0.06980001	0.06713103	0.1944734	0.02084708	0.0000000
2	ARG-DDA1	1	Argentina	9.511960	0.004797360	0.011651037	0.2698809	0.16469137	0.07644078	0.08687492	0.1845937	0.03329328	0.0000000
3	ARG-FFL1	1	Argentina	7.975901	0.004797360	0.011651037	0.2235151	0.14991512	0.06966254	0.07393891	0.1463870	0.02303831	0.0000000
4	ARG-FLM1	1	Argentina	7.800659	0.004797360	0.011651037	0.2379087	0.15472910	0.07049397	0.06903591	0.2050895	0.02708130	0.0000000
5	ARG-ICR1	1	Argentina	7.947105	0.001912794	0.099581718	0.2597944	0.14865604	0.06380468	0.07446050	0.2072410	0.03117191	0.0000000
6	ARG-SAL1	1	Argentina	8.714034	0.002505292	0.043381915	0.2242742	0.14072456	0.04959497	0.09691696	0.1799086	0.03925861	0.0000000
7	AUS-CAV1	2	Australia	7.448486	0.003032742	0.016059005	0.3374560	0.14914934	0.05520893	0.04424401	0.2108030	0.04723972	0.0000000
8	AUS-EAG1	2	Australia	7.347678	0.002279393	0.139749870	0.2969021	0.09014709	0.11717298	0.06591905	0.2039159	0.01773638	0.0000000
9	AUS-HAR1	2	Australia	8.120381	0.002329275	0.130278988	0.2318927	0.12675206	0.07713586	0.04852565	0.1974210	0.03149089	0.0000000
10	AUS-IB41	2	Australia	8.793289	0.002224879	0.078379292	0.2218921	0.15336706	0.09708472	0.06232333	0.1729105	0.02983736	0.0000000
11	AUS-KIL1	2	Australia	11.065081	0.003101257	0.251759064	0.3205486	0.41101069	0.04645483	0.11424976	0.2143714	0.13380257	0.0000000
12	AUS-KIR1	2	Australia	9.906680	0.001751569	0.043467972	0.4681868	0.32766096	0.11905133	0.09958243	0.2344413	0.06984530	0.0000000
13	AUS-NUG1	2	Australia	7.782126	0.001300266	0.198259543	0.3576069	0.11359247	0.17179633	0.10464639	0.2256802	0.02588836	0.0000000
14	AUS-SOC1	2	Australia	8.669339	0.003107816	0.004205524	0.3341900	0.25097597	0.08811798	0.11324690	0.2238087	0.05695031	0.0000000
15	AUS-TGH1	2	Australia	8.048740	0.002087554	0.137686111	0.3398740	0.12475039	0.08776693	0.05682283	0.1897665	0.02820311	0.0000000
16	AUS-VAF1	2	Australia	9.730056	0.001456612	0.204650169	0.4222450	0.21210996	0.15819995	0.09401661	0.2511838	0.05009667	0.0000000
17	AUS-WBL1	2	Australia	8.708729	0.002434400	0.018819241	0.3713302	0.17024227	0.09200559	0.05851991	0.2288707	0.03886806	0.0000000
18	AUS-WES1	2	Australia	10.506057	0.001581742	0.092312869	0.3439428	0.14284193	0.10332681	0.06467705	0.2347263	0.03711495	0.0000000
19	CHI-CDD1	3	Chile	10.174825	0.003123636	0.328637082	0.2110648	0.19796589	0.07561129	0.06840427	0.1584983	0.03954664	0.0000000
20	CHI-CDM1	3	Chile	8.408178	0.001988995	0.023890816	0.1427522	0.10076303	0.08677508	0.08396911	0.1712643	0.01484752	0.0000000
21	CHI-CMO1	3	Chile	8.924974	0.003330979	0.163480821	0.2754740	0.14497268	0.07131028	0.06886175	0.2002443	0.04080054	0.0000000

Figure 11: Some rows and columns of the data set.

The data set `Wine.xlsx` looks like this. The objects are represented by different samples in the rows, while dependent and independent arguments are represented in the columns.

Different syntaxes can be used:

- `dataframe$variable`  
This option is valid for
  - Data frames
  - Lists

This is the easy option if you know the name of the variable and the variable has a valid variable name.

If the variable name is invalid, you can be sneaky, and get around this problem by placing single " around the name (see example below).



RStudio returns all the values in the column as a vector. This `$` notation is incredibly useful because you will often store the variables of your data sets as columns in a data frame. From time to time, you'll want to run a function like `mean` or `median` on the values in a variable. In R, these functions expect a vector of values as input, and `dataframe$variable` delivers your data in just the right format.

### Blank spaces/indexes: Extracting the whole set of rows and/or columns

Suppose you want to make a subset of all the Argentinian wines, and you want to include all the variables.

```
Wine <- read_excel('Wine.xlsx')
Wine_Arg <- Wine[1:6,]
```

### Extracting columns/rows by including the ones you want

But maybe you are only interested in the ethanol content in general in wine. Then you can extract the ethanol column from all the rows (tip: the column number is found by holding the mouse cursor over the variable name, as seen in Figure 11):

```
Wine_ethanol <- Wine[,5]
Wine_ethanol_name <- Wine$Ethanol
```

By using the `$` notation, you don't need to type out the whole name of the variable; the variables will pop up in a list, and as you type in the start of your variable name, the relevant variables on the list get limited down:

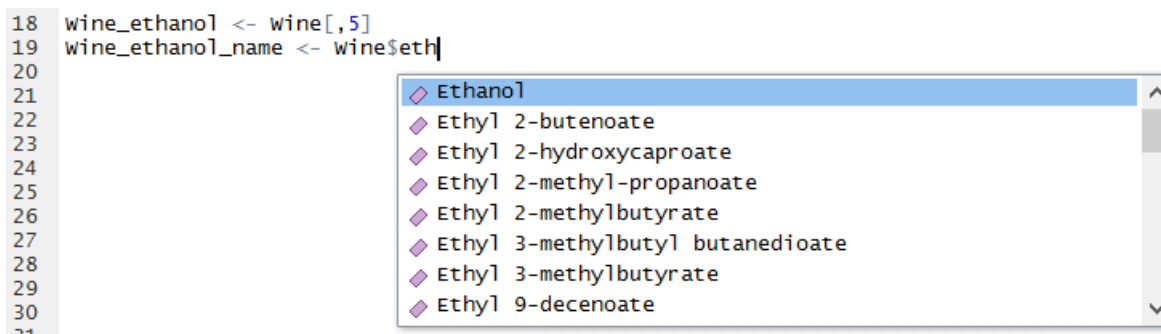


Figure 12: Handy auto completion

The auto completion can save you a bit of time, as it ensures the variable name is written down in a valid form; both spelling, capital vs. small letters, and: If the variable name is invalid (as described in Section 4.3), the name will be surrounded by quotation marks, when it is chosen from the auto completion list:

```
Wine_23methylbutanal <- Wine$'2/3-Methylbutanal'
```

### Extracting columns/rows by excluding the ones you do not want

Instead of choosing the rows or columns we want to include in our data set, we can choose which ones we would like to exclude.

Let's say we only want the response variables in this case, so we want to exclude the first 3 columns. This is defined by calling the rows or columns you want to exclude negatively:

```
Wine_response <- Wine [,-(1:3)]
```

#### 5.1.1 Using the subset command

In R the command `subset` is used to filter the data in a data frame based on a chosen criteria or criteria.

##### One criteria

Let's pick out all the Argentinian wine, which we also did earlier with a different approach:

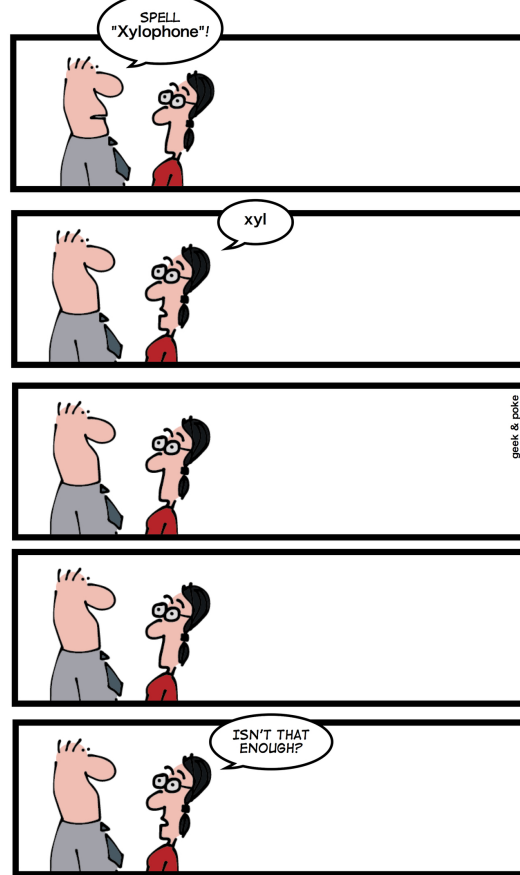
```
Wine_Arg2 <- subset(Wine, Country == 'Argentina')
```

##### Multiple criteria

Let's say, all the Argentinian wine are not relevant for whatever reason. We only want the Argentinian wine with a value less than 0.1 for Ethanol.

```
Wine_Arg3 <- subset(Wine, Country == 'Argentina' & Ethanol < 0.1)
```

## SIMPLY EXPLAINED



AUTO COMPLETION

Figure 13: Auto completion is very handy - but taking note of the names of the different variables can also be useful sometimes.

## 6 Introduction to selected commands

### 6.1 Aggregate

Aggregate is a very useful function - and it will be used throughout the course. What does it do?

If we want to ask R about any function, we can ask it for `help` :

Source editor:

```
help("aggregate")
```

Shown in Plots&information frame:

aggregate (stats)

R Documentation

#### Compute Summary Statistics of Data Subsets

##### Description

Splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.

Let's try to visualise this. You have this data set with coffee judges, `Results Panel1.xlsx`, where each judge has tested coffee of a certain temperature, several times.

	Sample	Assessor	Replicate	Intensity	Sour	Bitter	Sweet	Tobacco	Roasted	Nutty	Chocolate
1	31C		1	9.30	6.90	6.75	4.50	10.50	7.95	3.90	7.35
2	31C		1	8.70	8.10	7.95	4.35	9.60	8.85	4.80	6.90
3	31C		1	9.75	8.70	10.20	3.90	10.20	10.20	4.80	7.50
4	31C		1	11.70	10.95	11.40	3.15	11.55	10.05	3.45	5.85
5	31C		2	8.70	5.70	11.40	6.15	10.65	8.85	1.95	8.10
6	31C		2	8.70	9.30	10.35	4.95	11.70	10.95	5.40	10.35
7	31C		2	7.95	9.00	11.25	1.20	11.10	8.85	4.80	12.00
8	31C		2	10.65	10.05	12.75	1.05	12.30	10.50	1.20	11.40
9	31C		3	8.25	8.85	11.25	4.80	12.75	2.70	2.40	8.25
10	31C		3	9.15	8.25	13.35	4.50	13.50	4.80	5.70	8.70
11	31C		3	15.00	15.00	15.00	0.00	15.00	7.35	5.40	12.00
12	31C		3	4.50	6.90	11.40	5.40	13.05	11.70	0.90	7.65
13	31C		4	3.90	7.80	6.90	2.10	10.35	4.05	1.50	6.45
14	31C		4	9.30	6.30	10.80	1.35	10.80	4.50	2.10	10.50
15	31C		4	8.55	7.65	7.95	3.30	7.35	5.55	1.35	7.95
16	31C		4	1.35	2.85	4.35	1.20	3.30	1.20	4.05	3.75
17	31C		5	6.90	5.40	10.80	2.70	11.55	9.90	7.50	5.85
18	31C		5	9.30	6.60	9.00	1.95	10.50	11.85	3.75	9.45
19	31C		5	6.30	8.10	8.70	2.40	11.85	7.65	7.65	9.45
20	31C		5	5.85	5.25	10.35	6.45	9.60	9.75	5.85	3.75
21	31C		6	10.05	9.15	8.70	6.30	8.85	7.50	4.20	7.35
22	31C		6	8.70	7.95	10.35	3.75	11.70	7.05	3.30	9.30
23	31C		6	9.75	13.50	10.65	3.15	12.15	5.70	3.60	7.50
24	31C		6	10.05	12.60	7.95	3.45	10.35	4.80	2.85	8.55
25	31C		7	6.45	10.35	6.90	1.65	10.65	6.45	4.05	7.80
26	31C		7	4.35	5.70	4.20	5.55	5.70	6.75	4.20	7.95
27	31C		7	5.85	6.75	5.10	1.80	9.15	6.60	0.60	6.00
28	31C		7	9.30	5.70	9.00	1.95	7.80	9.60	4.20	5.40
29	31C		8	11.40	13.65	10.80	1.50	12.15	9.75	2.70	8.55
30	31C		8	4.80	10.05	6.75	5.40	7.20	9.00	7.05	10.80
31	31C		8	9.15	10.35	12.75	1.80	7.35	9.30	5.55	10.80
32	31C		8	12.15	7.05	8.55	3.45	13.20	12.60	1.80	4.65
33	37C		1	10.05	10.05	8.70	3.60	9.60	8.70	3.15	4.50
34	37C		1	10.65	10.50	8.10	5.10	9.60	10.20	5.10	5.55
35	37C		1	9.60	9.45	10.80	3.75	10.65	10.35	4.05	5.55

Subgroup 3: Replicates

Subgroup 2: Individual judges

Subgroup 1: Coffee temperature

Figure 14: As shown here, the data set can be divided in several subsets; firstly, the coffee has been served at different temperatures. Secondly, several judges have tasted the coffee at a given temperature, giving as second subset. Thirdly, each judge have tasted a coffee at a given temperature several times, giving a third subset.

The columns from Intensity to Chocolate are response variables, i.e., which score a judge has given on the coffee. A higher value of Chocolate, means the judge thought this coffee was quite chocolatey in its taste.

Lets say we want to find the mean value for the response variable, for each temperature. I.e., we want one mean value for intensity, sour, etc, and the mean value is calculated based on all judges and their replicates.

Source editor:

```
CoffeeMean <- aggregate(CoffeeRP,  
                        by=list(CoffeeRP$Sample),  
                        FUN="mean")
```

Shown in Dataviewer:

Group.1	Sample	Assessor	Replicate	Intensity	Sour	Bitter	Sweet	Tobacco	Roasted	Nutty	Chocolate
1	31C	NA	4.5	2.5	8.306250	8.451562	9.487500	3.285938	10.420312	7.884375	3.829687
2	37C	NA	4.5	2.5	8.620313	9.075000	8.873437	4.143750	9.979688	8.095312	3.914062
3	44C	NA	4.5	2.5	8.517188	7.087500	9.281250	4.659375	8.878125	8.882812	4.410938
4	50C	NA	4.5	2.5	10.528125	7.589062	10.115625	3.046875	9.529687	9.660937	4.743750
5	56C	NA	4.5	2.5	9.946875	8.268750	10.598437	2.868750	10.120313	9.693750	3.829687
6	62C	NA	4.5	2.5	10.110938	7.790625	9.632812	3.314062	8.835938	9.787500	4.195312

In the `aggregate` command, the first thing in the parenthesis is that we tell R which object we want to be used; here the data set that we have called `CoffeeRP`. Next, `by=list` defines which columns we want to 'save' information about. In this question, I only want a statistical value about the temperature, so I am not interested in keeping information about judges or their replicates.

If you want to 'save' information about more than one column, for example you want a mean score from each judge for each coffee temperature, you just add them inside the brackets, like so: `by=list(CoffeeRP$Assessor, CoffeeRP$Sample)`. With `FUN` you decide the summary statistics you want to calculate.

R shows the results as shown above, where it has given each subset the characteristic name of which it made statistics about.

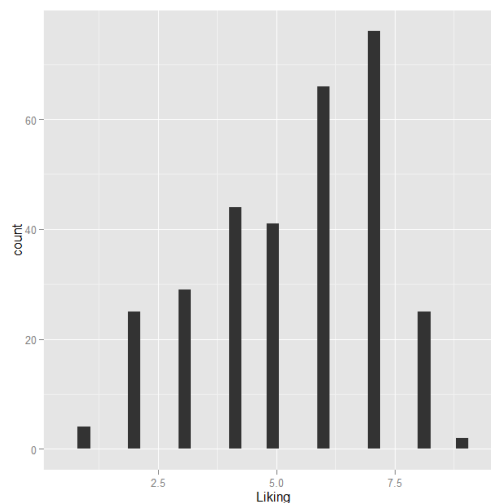
## 6.2 qplot

1) Importing data from excel:

```
> Coffee <- read_excel('Results Consumer Test.xlsx')
```

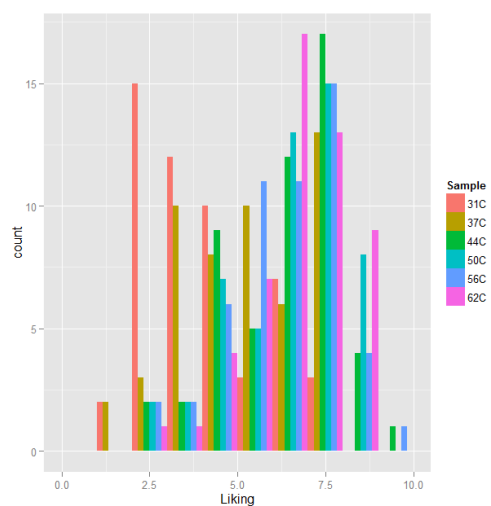
2) Make a histogram

```
> qplot(data=Coffee, Liking)
```



Try to modify the colour, the bin width, the transparency and the main title on the plot.

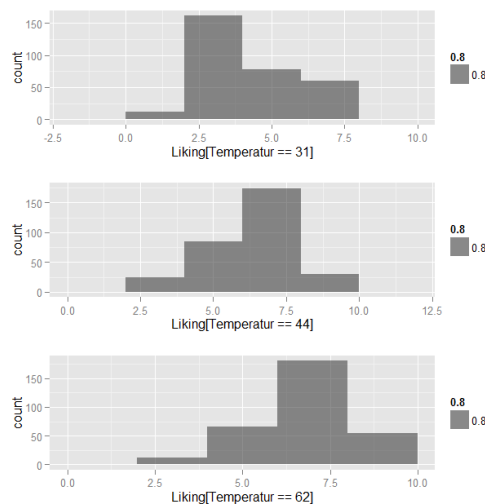
```
> qplot(data=Coffee, Liking, fill=Sample, position='dodge', binwidth=1)
```



Interpreting this figure, how does the temperature affect the liking?

If you do not want to overlay the histograms, it is possible to plot them as individual panels:

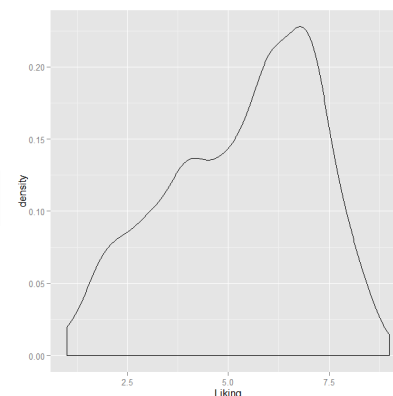
```
> p1 <- qplot (data=Coffee, Liking [Temperatur==31], geom="histogram",alpha=0.8, position="dodge",binwidth=2)
> p2 <- qplot (data=Coffee, Liking [Temperatur==44], geom="histogram",alpha=0.8, position="dodge",binwidth=2)
> p3 <- qplot (data=Coffee, Liking [Temperatur==62], geom="histogram",alpha=0.8, position="dodge",binwidth=2)
> grid.arrange(p1,p2,p3, ncol=1)
```



## Densitogramplot

A densitogram is a smoothed extension of the histogram; represents the same type of information. The bin width in the histogram controls the resolution, whereas the counterpart is the degree of smoothing. By adding a single option to the `qplot()`, the plot is changed to a densitogram:

```
> qplot(data=Coffee, Liking, geom="density")
```

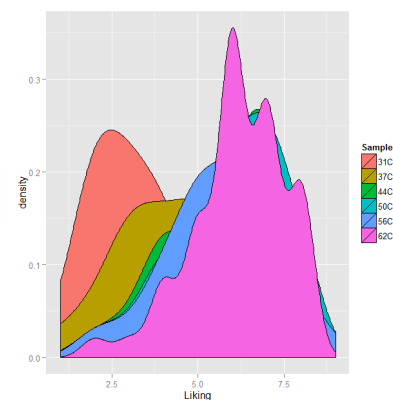


Try to modify the smoothing of the densitogram by changing the option `adjust=...`

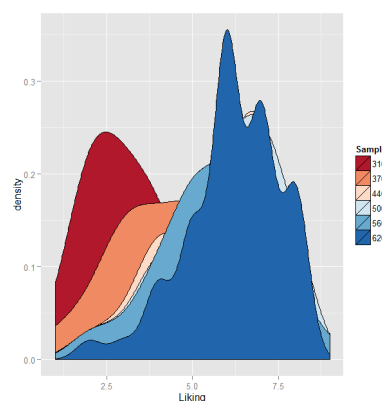
Try to make the smoothing very refined (e.g. `adjust=0.3`). Does this reflect the underlying distribution? What is a suitable smoothing option for these data?

Adding additional information:

```
> qplot(data=Coffee, Liking, fill=Sample, geom="density")
```

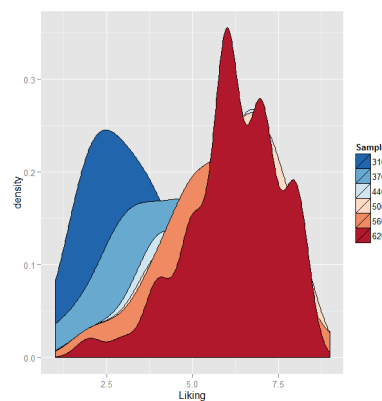


```
> qplot(data=Coffee, Liking, fill=Sample, geom="density")  
> +scale_fill_brewer(palette="RdBu")
```



Manually specifying the colors:

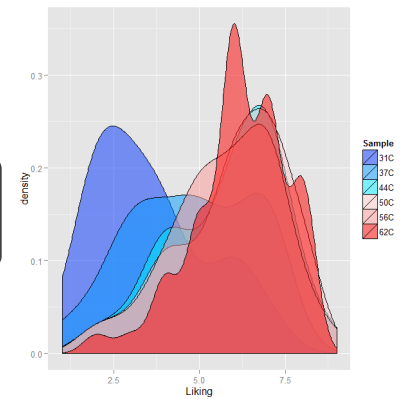
```
> qplot (data=Coffee, Liking, fill=Sample, geom="density")  
> +scale_fill_manual(values=rev(brewer.pal(6,"RdBu")))
```



The plot is not optimal, as only the densitogram for 62 degrees is fully visible. Try to adjust the transparency by `alpha=...` to get a better version of the same plot.

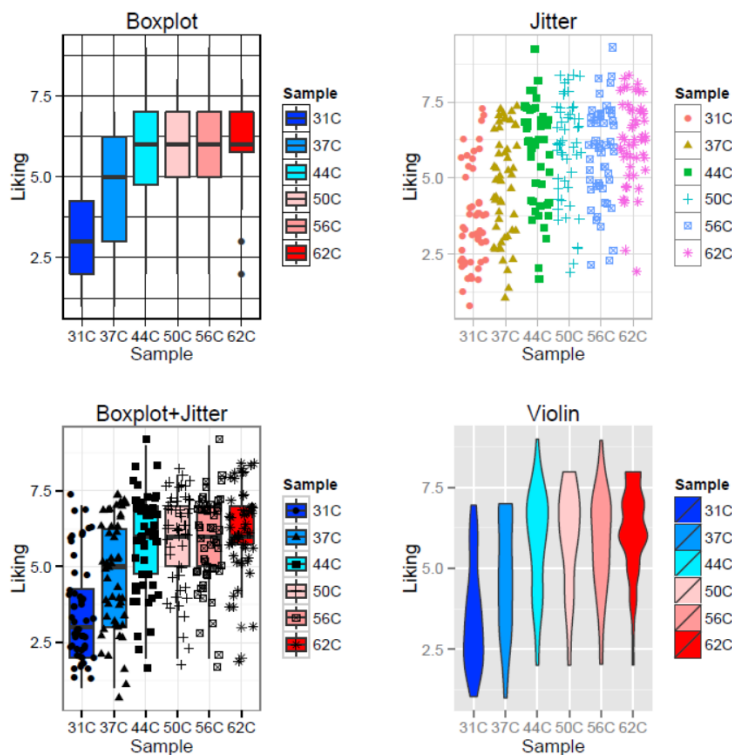
Can also specify the exact colours:

```
> colPalette <- c('#0033FF','#0099FF','#00EEFF','#FFCCCC','#FF9999','#FF0000')
> qplot(data=Coffee, Liking, fill=Sample, geom="density", alpha=I(0.5))
> + scale_fill_manual(values=colPalette)
```



Boxplot, jitterplot and violinplot

```
> c11 <- scale_fill_manual (values=colPalette)
> p1 <- qplot(Sample, Liking, data=Coffee, geom="boxplot", fill=Sample)+ theme_linedraw()+ggtitle("Boxplot")+c11
> p2 <- qplot(Sample, Liking, data=Coffee, geom="jitter", color=Sample, shape=Sample)+theme_light()+ggtitle("Jitter")+c11
> p3 <- qplot(Sample, Liking, data=Coffee, geom=c("boxplot", "jitter"), fill=Sample, shape=Sample, main="Boxplot+Jitter")+theme_bw()+c11
> p4 <- qplot(Sample, Liking, data=Coffee, geom="violin", fill=Sample, main="Violin") + theme_gray()+ c11
#Skriver ud figurer i pdf
> pdf ("Box_jitter_violin.pdf")
> grid.arrange(p1,p2,p3,p4,ncol =2)
> dev.off()
```



Things to notice from the code:

- the title on the individual plots can be inferred using either `qplot(...,main='The Title')` or by 'adding' the title; `|qplot() + ggtitle('The Title')`--
- The color scale is predefined and simply added to the individual plots
- The background of the plots are different, and is inferred by adding `+ theme_XXX()` (default: `theme_gray()`)
- In the *Jitter* plot, the colors do not work. Try to use `scale_color_manual()` instead of `scale_fill_manual`, and see what happens.

## 6.2.1 Data types

### Treating numerical data as factors

Decimal values are called numerics in R, and is the default computational type (see also Section 4.1). However, sometimes we would prefer our numbers not to be treated as a continuous range of decimal values, but rather like factors.

Consider the following data set:

	Sample	Temperature	Assessor	ServingOrder	TemperatureJudgment	Liking	Intensity	Sour	Bitter	Sweet	Male	Female
1	31C	31	1	6	2	3	4	3	4	2	1	0
2	31C	31	2	6	2	3	7	5	8	3	1	0
3	31C	31	3	6	1	3	2	1	4	6	0	1
4	31C	31	4	6	1	2	5	6	4	3	1	0
5	31C	31	5	6	2	2	2	3	2	1	1	0
6	31C	31	6	6	2	4	3	4	2	1	1	0
7	31C	31	7	6	2	2	5	8	7	3	0	1
8	31C	31	8	6	1	2	2	3	4	5	0	1
9	31C	31	9	6	1	1	5	6	4	3	1	0
10	31C	31	10	6	4	4	3	1	5	1	0	1
11	31C	31	11	6	1	3	1	0	1	7	0	1

Figure 15: The top rows in the data set

It makes sense that the response variables can take on any decimal number in a given range for the assessors to choose from.

It does *not* make sense to treat the different judge, who are each assigned by a number, as decimal numbers, as we only have integers for the different judges. We would prefer to treat the numbers designated for the judges, as factors. This can be done 2 ways, either it can be defined in the start, with the command `as.factor` or you can define it later in the function, by surrounding the relevant variable with `factor(variable)`.

#### Visual explanation of the differences - (Figure 16)

```
library(gdata) #needed by rename.vars
library(gridExtra) #use for grid.arrange
library(readxl)

CoffeeP <- read_excel("Results Panel.xlsx")

CoffeeAG <- aggregate (CoffeeP,by=list(CoffeeP$Assessor,CoffeeP$Sample),FUN ="mean")
# renaming some variables
CoffeeAG <- rename.vars(CoffeeAG,c("Group.1","Group.2"),c("Judge","Temp"))

p1 <- qplot(data=CoffeeAG,Temp,Intensity,group=Judge,color=Judge)+geom_line()
p2 <- qplot(data=CoffeeAG,Temp,Intensity,group=Judge,color=factor(Judge))+geom_line()
grid.arrange(p1, p2, ncol=2)
```

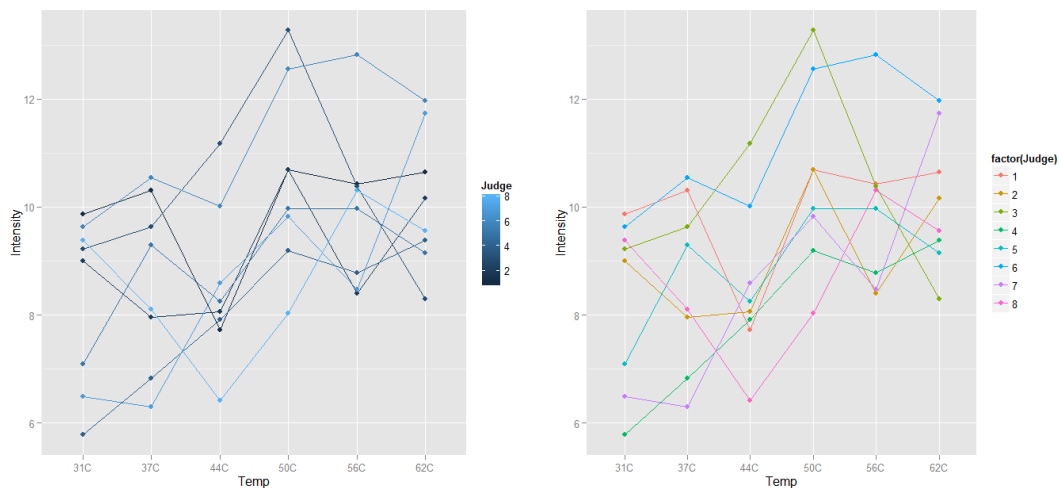


Figure 16: In the figure to the left, the numbers representing the judges are treated as a continuous scale, whereas on the figure to the right, the numbers are treated as factors.

## Telling R to treat numerical string as factor from the start - (Figure 17)

```
library(gdata) #needed by rename.vars
library(gridExtra) #use for grid.arrange
library(readxl)

CoffeeP <- read_excel("Results Panel.xlsx")
# Telling R to treat Assessor as factors
CoffeeP$Assessor <- as.factor(CoffeeP$Assessor)

CoffeeAG <- aggregate (CoffeeP,by=list(CoffeeP$Assessor,CoffeeP$Sample),FUN ="mean")
# renaming some variables
CoffeeAG <- rename.vars(CoffeeAG,c("Group.1","Group.2"),c("Judge","Temp"))
head(CoffeeAG)

p1 <- qplot(data=CoffeeAG,Temp,Intensity,group=Judge,color=Judge)+geom_line()
p2 <- qplot(data=CoffeeAG,Temp,Intensity,group=Judge,color=factor(Judge))+geom_line()
grid.arrange(p1, p2, ncol=2)
```

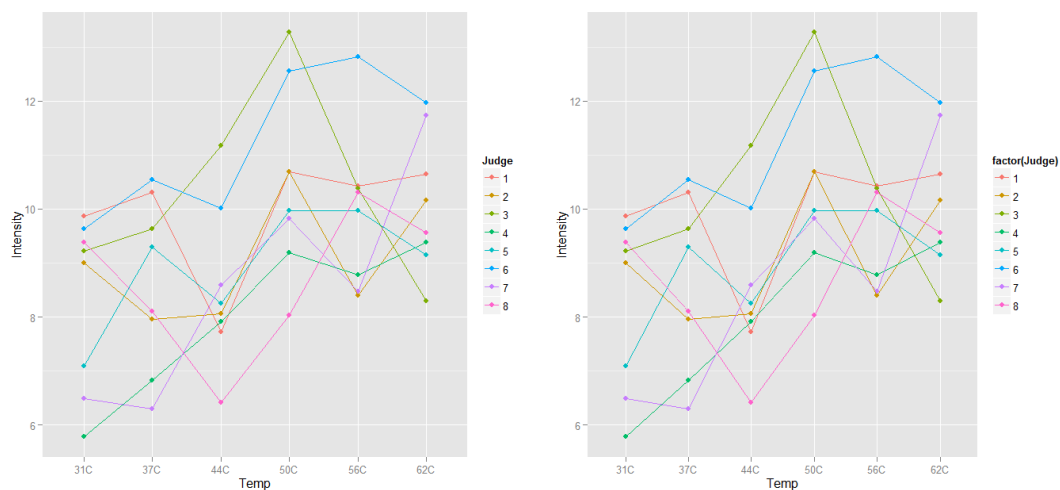


Figure 17: As we defined assessors as factors in the start, R will treat them as factors all along, and therefore we see no difference in these plots.



## 7 Helpful sites

- Examples for use of different commands: <http://www.statmethods.net/>
- Cheatsheets for functions and commands: <https://www.rstudio.com/resources/cheatsheets/>

## 8 Exercises

### 8.1 Getting started, nice and slow.

1. Create a new folder, and call it `FDA_week0`
2. Download all the files given on the title page of this exercise set, and save it in the folder.
3. Open RStudio, and set your FDA folder to be the current working directory. Find the `code` for this folder, and copy it into a script that you save with your choice of name.
4. What command do you use to check the content of your FDA folder?
5. Use Google to find the packages required in order to use the following functions. What do the different functions allow you to do?
  - `qplot`
  - `rename.vars`
  - `melt`

### 8.2 Coffee

1. Read in `Results Consumer Test.xlsx`  
**Get an overview of the data set:**
2. How many rows and columns are there in the data set?
3. What vector type is your data set?
4. What is the data structure?
5. Write a command to find the the value of row 3 and column 5.  
**Making subsets:**
6. Make subsets of each coffee serving temperature

## 9 Debugging the different week sets

Below are some debugging exercises for the different data sets that you will use during the course.

### 9.1 Week 1: Importing files, and calling on objects and functions

For some of you, coding in RStudio may seem simple. The aim with these debugging exercises is to train you to analyse the errors RStudio gives you, and to give you some tools to use to avoid issues when coding in RStudio. Many of the debugging exercises throughout the course will be related to the data sets used in other exercises given in the same week, so you might find it sensible to do the debugging-exercises first, to get to know the data sets and their potential issues, before the struggle starts.

#### 9.1.1 Importing a data set, debugging function calls

Data set: Results Consumer Test.xlsx

1. Why won't RStudio read in the file in the following cases?

```
> Coffee          <- read_excel("Results Consumer Test.xlsx")

Error: could not find function "read_excel"

> Coffee          <- read_excel("Results Consumer Test.xlsx")

Error: 'file.xlsx' does not exist in current working directory ('C:/Current working directory').
```

2. Why is the view-function not working?

```
> Coffee          <- read_excel('Results Consumer Test.xlsx')
> view(Coffee)

Error: could not find function "view"
```

3. What is missing when you call for the `aggregate` to do its calculations?

```
> CoffeeRP <- read_excel('Results Panel.xlsx')
> CoffeeAG <- aggregate(by=list(CoffeeRP$Assessor,CoffeeRP$Sample), FUN='sd')

Error in is.ts(x): argument "x" is missing, with no default
```

## 9.2 Debugging - Invalid variable names, PCA

Data set: Wine.xlsx

1. Why doesn't RStudio want to make the second plot I ask for below?

```
> Wine <- read_excel('Wine.xlsx')
> qplot(data=Wine, Furfural, Ethanol)
> qplot(data=Wine, Ethyl acetate, Ethanol)
Error: unexpected symbol in "qplot(data=Wine, Ethyl acetate"
```

- Come with 2 suggestions of how you could solve it, and discuss advantages and disadvantages with the two methods.

2. Here is a screen shot of other variable names from the dataset

2/3-Methylbutanal	Ethanol	Ethyl propanoate	Ethyl 2-methylpropanoate	Propyl acetate	2-Methylpropyl acetate,	Ethyl butyrate	Ethyl 2-methylbutyrate	2,3-Pentanedione	Ethyl 3-methylbutyrate	2-Butanol	3-Methylbutyl acetate
-------------------	---------	------------------	--------------------------	----------------	-------------------------	----------------	------------------------	------------------	------------------------	-----------	-----------------------

- Give examples of 2 invalid names (invalid for different reasons), and 2 valid names. Feel free to use Google.
- Do a litmus test on your suggestions by trying to make a plot with the suggested variables.
- How would you treat the different invalid names, so that you could call for the variables in RStudio? Come with 2 suggestions, and discuss advantages and disadvantages with them.
- As a researcher within food chemistry, and a heavy user of RStudio to plot your data, how would you suggest naming the variables to avoid running into these issues in Rstudio? What are the advantages and disadvantages here?

3. Why is RStudio unhappy with the code below?

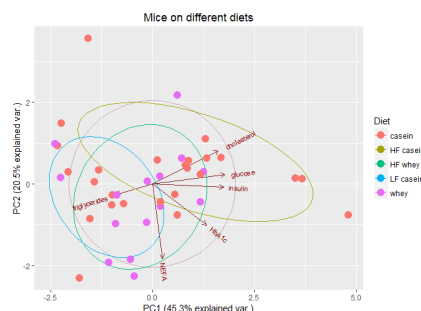
```
> Mouse <- read_excel('Mouse_diet_intervention.xlsx')
> Mouse <- Mouse[c(1:40),]
> Mouse_m <- Mouse [,c(5,23:28)]
> MousePCA <- prcomp(Mouse_m, center=TRUE, scale.=TRUE)
Error in colMeans(x, na.rm=TRUE): 'x' must be numeric
```

- What should be changed in the code in order for it to work?

4. Save this exercise till when you have learned more about PCA plots in the course.

We make a PCA plot on the mouse dataset, and want to group the different mouse based on the explaining variables. Why do the dots come in 2 colours, while the ellipses come in 3?

```
> Mouse <- read_excel('Mouse_diet_intervention.xlsx')
> Mouse <- Mouse[c(1:40),]
> Mouse_m <- Mouse [,c(23:28)]
> MousePCA <- prcomp(Mouse_m, center=TRUE, scale.=TRUE)
> M <- ggbiplot(MousePCA, obs.scale=1, var.scale=1, groups=Mouse$Fat_Protein, ellipse=TRUE, circle=TRUE)
> M <- M + guides(color=guide_legend('Diet'))
> M <- M + geom_point (size=I(4), aes(color=Mouse$diet_protein))
> M <- M + ggtitle ('Mice on different diets')
> print(M)
```



- How would you change the code in order to make the dots represent the same variety in the data as the ellipses?

### 9.3 Debugging week 3 - NA's and R object type classes

Data sets: `Mouse_diet_intervention.xlsx`, `MaramaBeanOil0x.xlsx`

1. We want to find the mean of the body weight of the mice at week 0. Why does RStudio reply with NA?

```
> Mouse <- read_excel('Mouse_diet_intervention.xlsx')
> mean(Mouse$bw_w0)
[1] NA
```

- Suggest 2 solutions on this problem.
- How can you, in RStudio, localise all the columns containing NA's?

2. We want to make this plot, which includes the variables `Marama.oil$Temperature` and `Marama.oil$Light`, but RStudio replies with an error message.

```
> library(ggplot2)
> library(readxl)
> library(gridExtra)
> Marama <- read_excel('MaramaBeanOil0x.xlsx')
> Marama.oil <- subset(Marama, Product=='oil')
> qplot(data=Marama.oil,
+ Temperature:Light, sqrt(PV),
+ geom=c('boxplot', 'jitter'))+
+ facet_grid(.~Month)+theme_bw()
Error in Temperature:Light : NA/NaN argument
In addition: warning messages:
1: In Temperature:Light :
  numerical expression has 28 elements: only the first used
2: In Temperature:Light :
  numerical expression has 28 elements: only the first used
3: In eval(expr, envir, enclos) : NAs introduced by coercion
4: In Temperature:Light :
  numerical expression has 28 elements: only the first used
5: In Temperature:Light :
  numerical expression has 28 elements: only the first used
6: In eval(expr, envir, enclos) : NAs introduced by coercion
```

- Why is RStudio unhappy with the command?  
Hint: check the storage class of your values.
- Suggest two ways in which you can change the coding so that the `qplot` can be made. What are the advantages and disadvantages of the two different solutions?

### 9.4 Debugging week 4 - R object type classes

Dataset: `dates.xlsx`

1. We want to make a mean of the responses from the different judges on the response variable `aroma`.

```
> Dates <- read_excel('dates.xlsx')
> Dates_s<-Dates[c(2:24),c(2:6)]
> rowMeans(Dates_s)
Error in rowMeans(Dates_s) : 'x' must be numeric
> |
```

- What is the problem?
- How would you solve it? Do you have to solve it one variable at a time, or can you find a more time efficient way to change all the variables?