Introduction to `R`

11 August 2020

Contents

`R`

`R` is a programming language and free software environment that you can use to get a computer to generate, import and manipulate data. `R` performs mathematical and statistical operations and produce high quality vizualisations. `R` outputs of all these procedures in any format you require, e.g., *.pdf*, *.html* or as *Microsoft Word* files.

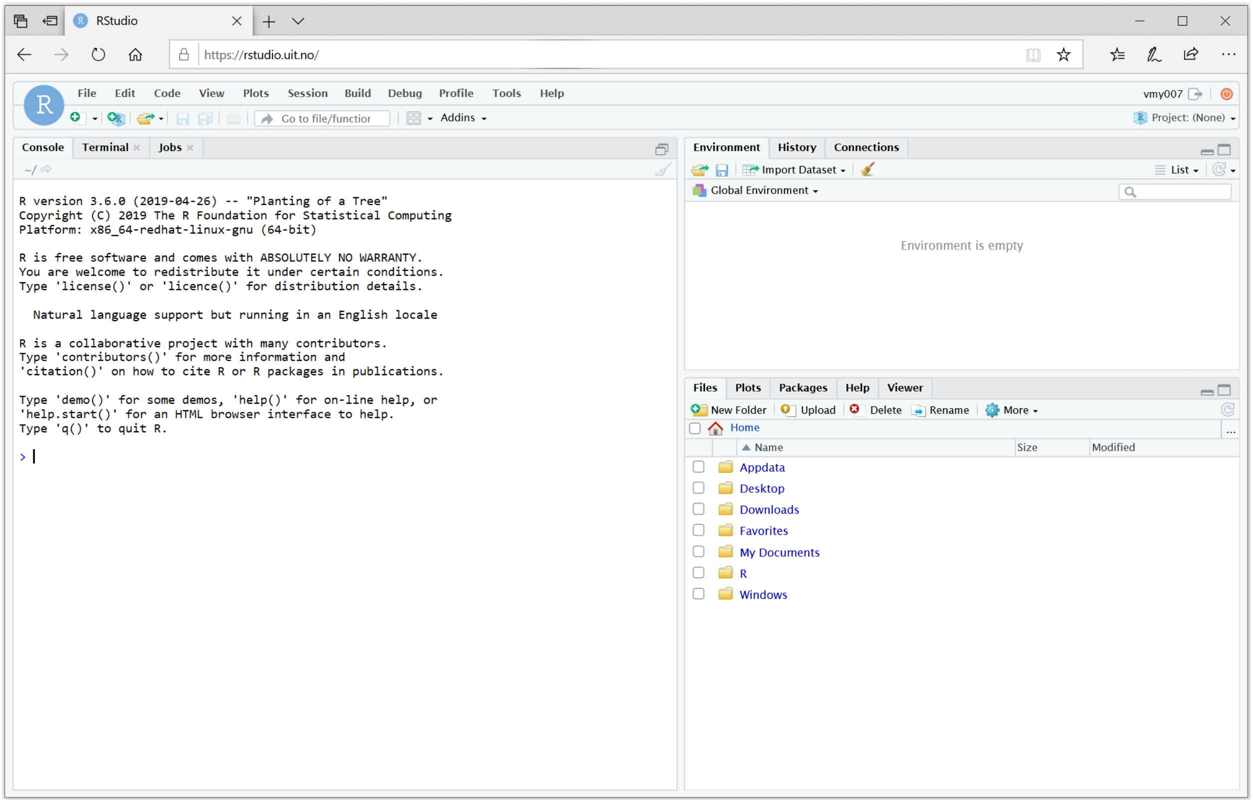
Between the R language and the computer is a software that interprets R code and communicates your instructions to the computer's operating system and hardware. Due to the popularity of R, there are many such interpreters. You will be using one that is called RStudio.

Most likely you will use RStudio as a web service. This means that you can use the software, and run code through a web browser. At **UiT**, go to **https://rstudio.uit.no** and log on using your ordinary **UiT** username and password. You need an Internet connection and a recent web browser. No other software installation is required. If off campus, you need to use the **VPN** service, and map your personal **UiT** drive. Follow the instructions *here in norwegian* and *here in english*.

It's also possible to install R and RStudio on your computer. Based on your operating system, follow the links to *R download*, and *RStudio*. Note that this requires knowledge on how to install software.

RStudio

RStudio has a toolbar and is divided into four panes or sections. The first time you open RStudio, you will se an image similar to this with only three panes.



```
R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-redhat-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Figure 1: **RStudio as displayed in the browser the first time**

Go to the toolbar and start a new R script (`File - New File - R Script`).

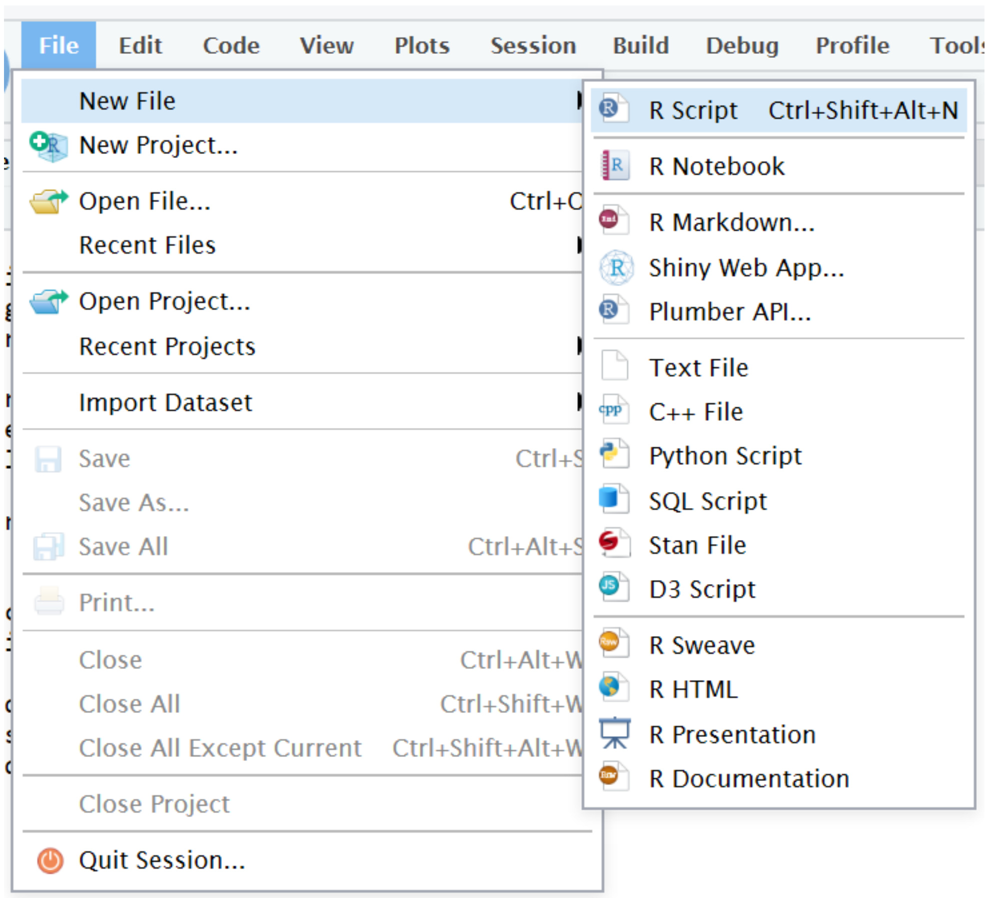
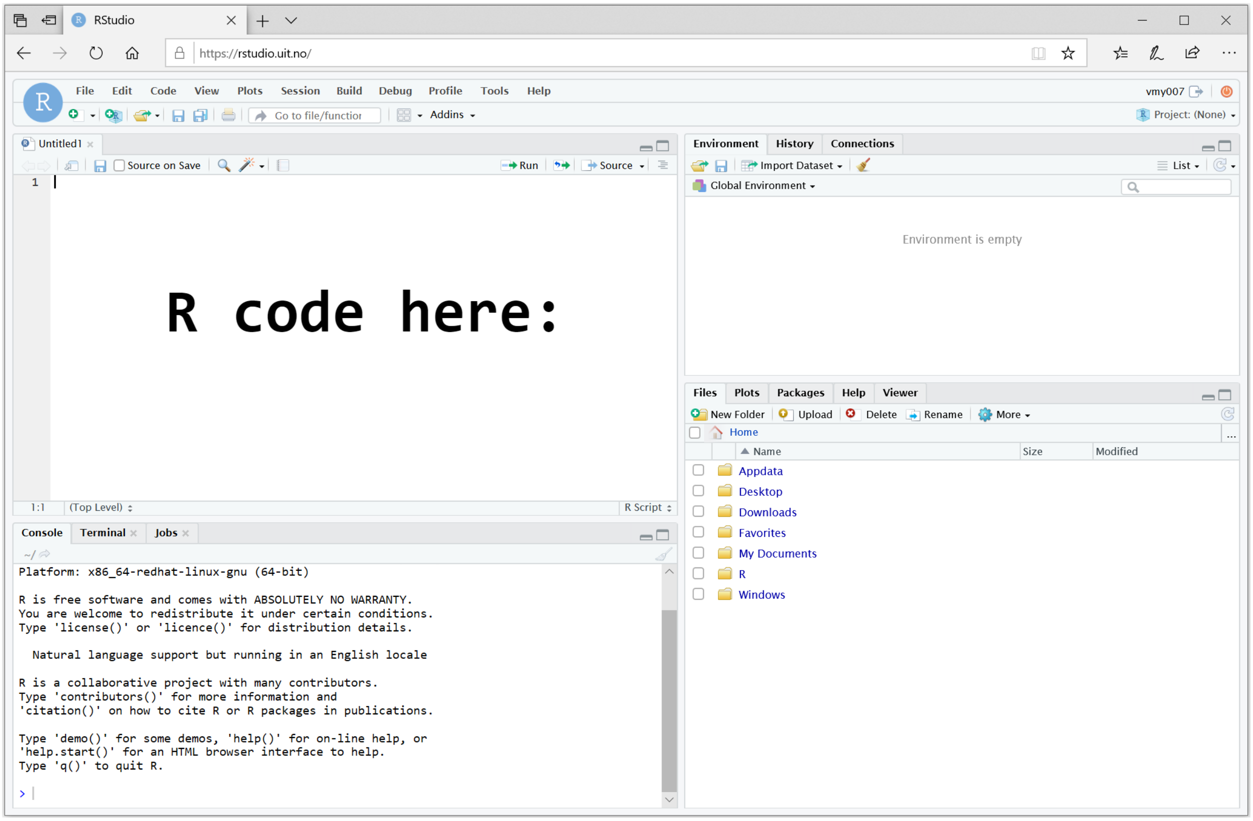


Figure 2: **Creating a new R Script**

You will now have four panes in your RStudio session. This is the standard layout of RStudio.



```
Platform: x86_64-redhat-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

Figure 3: **RStudio with code window**

1. The *Source pane* (upper left) provides a text editor for writing and running R code. Write R commands commands for immediate use or later re-use, after saving. If you do not see a *Source pane* in your session, create one using: `File - New File - R Script`.

2. The *Environment/History/Connections* pane (upper right). The *Environment* gives an overview of all the data, vectors and functions that is in the current workspace. *History* maintains an chronological record of your previous executed `R` commands.

3. The *Files/Plots/Packages/Help* pane (lower right), allows you to access files, plots, documentation of commands (help) as well as to load in new specialized software called packages.

4. The *Console* pane (lower left) where the output of your commands ends up when you `Run` them. Here you can also type commands directly for immediate output.

RStudio Cheat sheets

Once you start RStudio, you should start by downloading the *RStudio IDE Cheat Sheet* (Figure 2). This can also be accessed once RStudio is active from the `Help` toolbar.

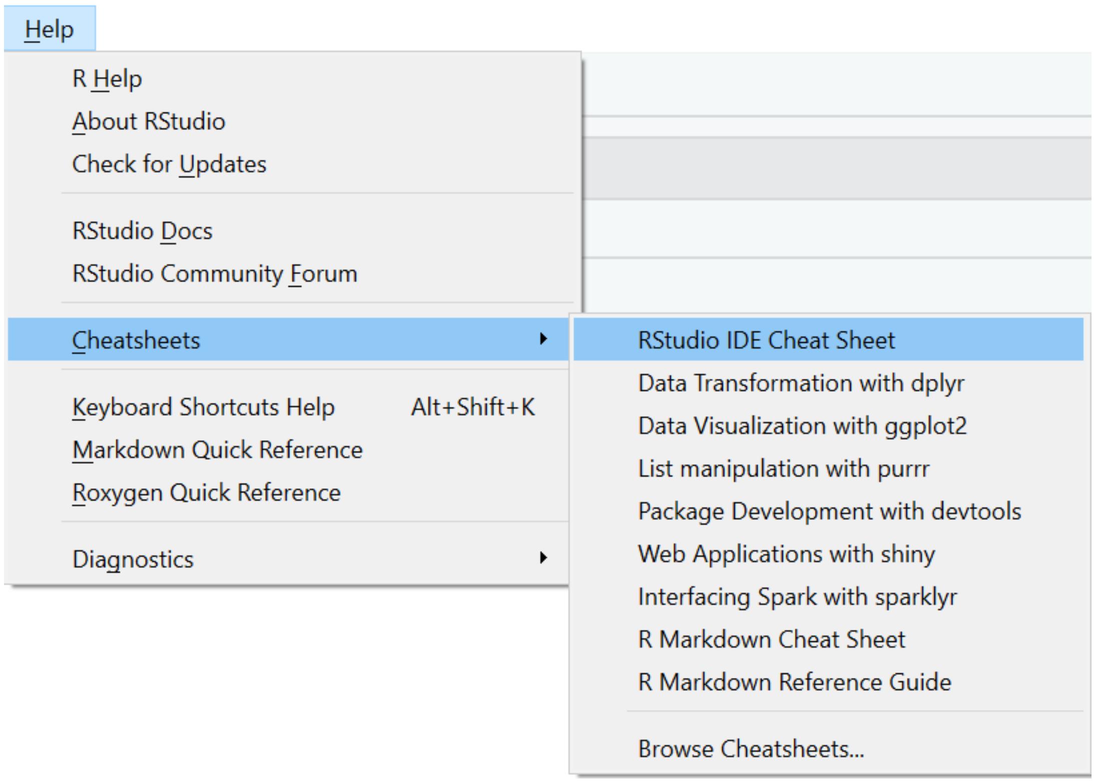

Help
R Help
About RStudio
Check for Updates
RStudio Docs
RStudio Community Forum
Cheatsheets
Keyboard Shortcuts Help Alt+Shift+K
Markdown Quick Reference
Roxygen Quick Reference
Diagnostics
RStudio IDE Cheat Sheet
Data Transformation with dplyr
Data Visualization with ggplot2
List manipulation with purrr
Package Development with devtools
Web Applications with shiny
Interfacing Spark with sparklyr
R Markdown Cheat Sheet
R Markdown Reference Guide
Browse Cheatsheets...

Figure 4: **Using Help to find RStudio Cheat Sheet**

The cheat sheet will help you to remember the basic setup and functionality of RStudio.

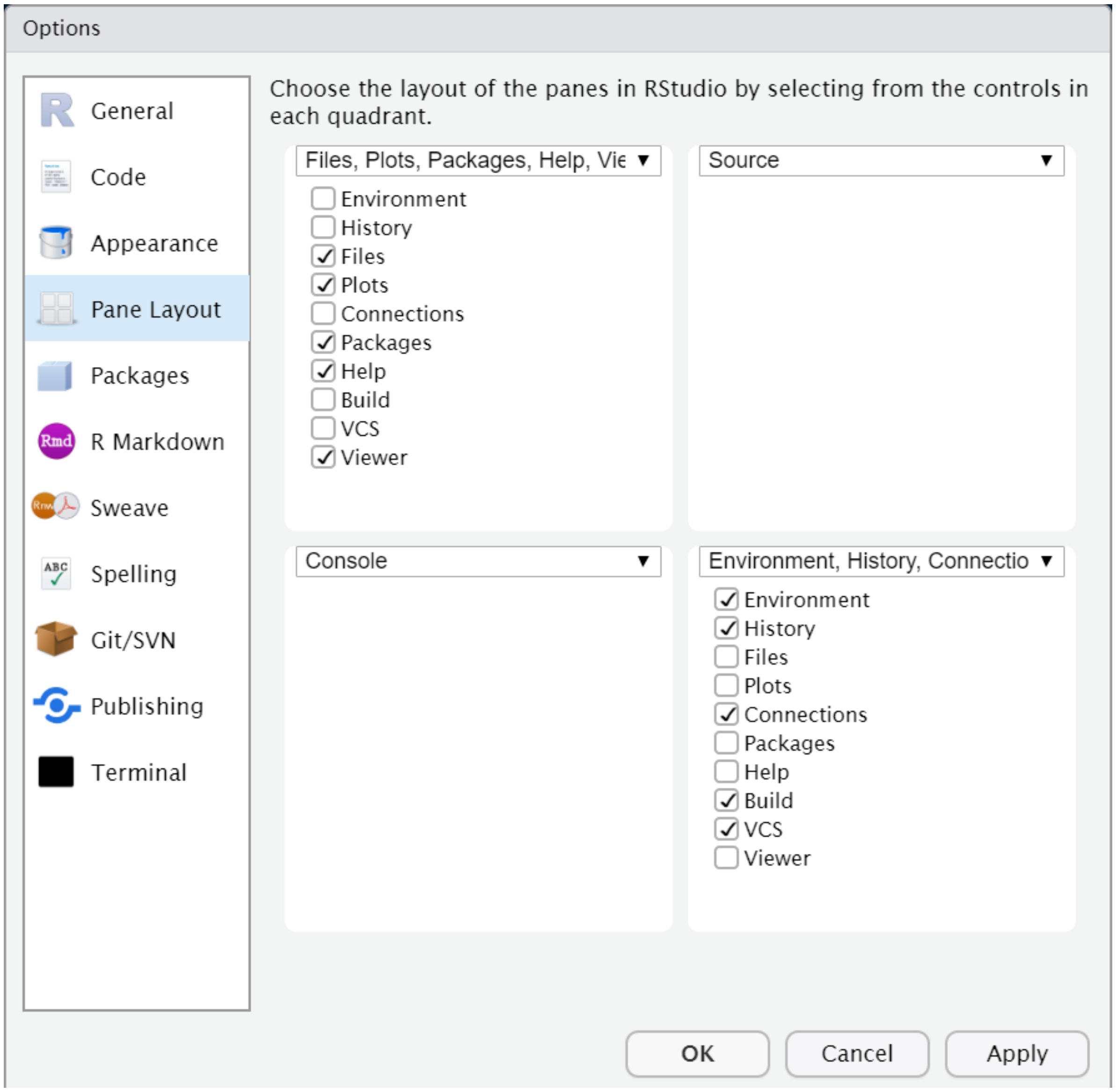
Changing RStudio layout and skins

If you go to `Tools - Global Options` and select `Pane Layout`on the left, the following window will appear:

Here you can arrange the layout of your RStudio session. My preferred layout is shown in the following figure. Note that you can also change what `Tabs` to display in the *toolbar* of the panes.

If you select `Apperance` on the left, the following window will appear:

Here, you can change skins depending on your preferred taste.


Options

Choose the layout of the panes in RStudio by selecting from the controls in each quadrant.

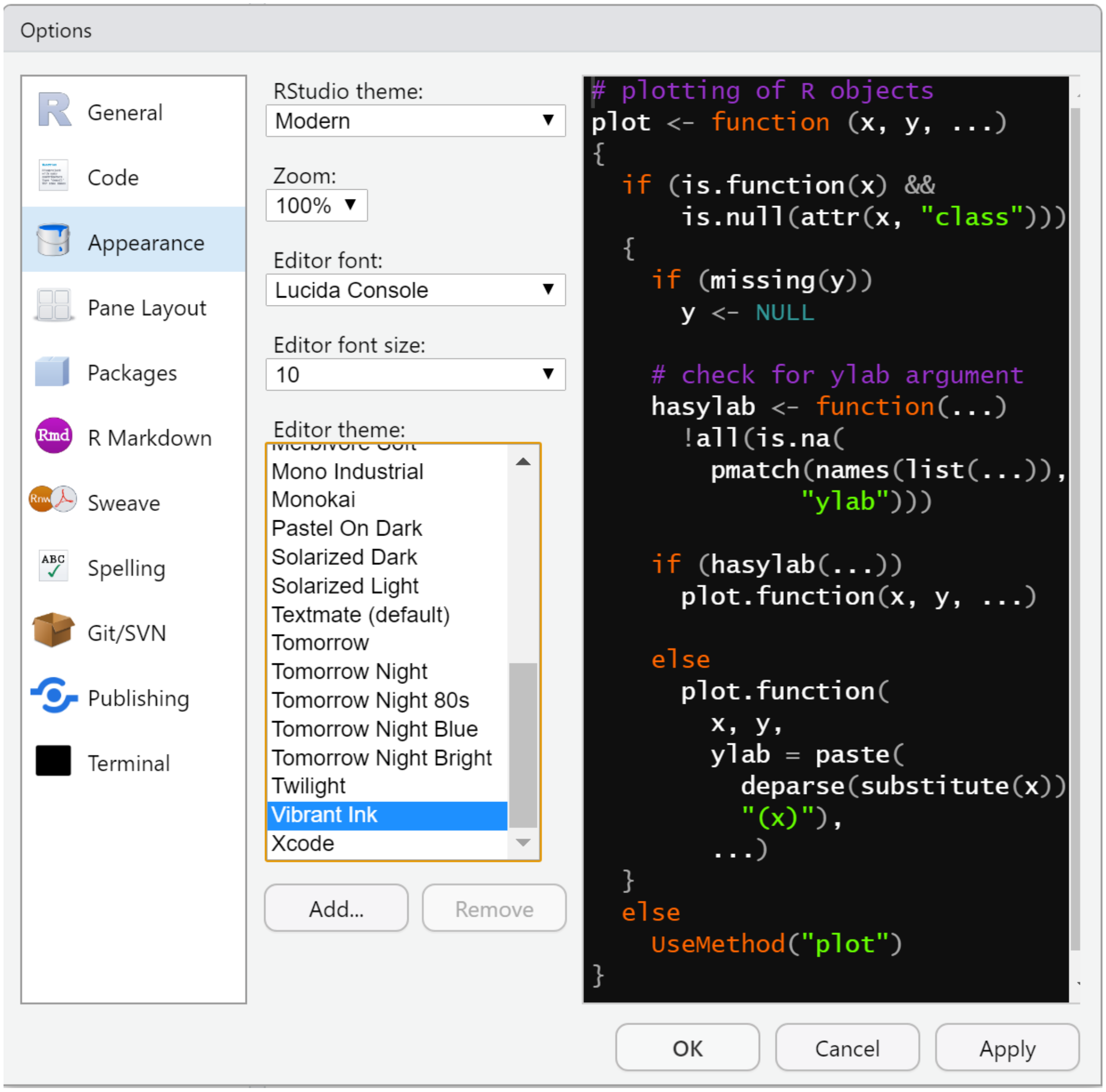
Files, Plots, Packages, Help, Vie ▼
- [ ] Environment
- [ ] History
- [✓] Files
- [✓] Plots
- [ ] Connections
- [✓] Packages
- [✓] Help
- [ ] Build
- [ ] VCS
- [✓] Viewer

Source ▼

Console ▼

Environment, History, Connectio ▼
- [✓] Environment
- [✓] History
- [ ] Files
- [ ] Plots
- [✓] Connections
- [ ] Packages
- [ ] Help
- [✓] Build
- [✓] VCS
- [ ] Viewer

Figure 5: **Changing the Pane Layout**



RStudio theme: Modern

Zoom: 100%

Editor font: Lucida Console

Editor font size: 10

Editor theme:

```
Merbivore Soft
Mono Industrial
Monokai
Pastel On Dark
Solarized Dark
Solarized Light
Textmate (default)
Tomorrow
Tomorrow Night
Tomorrow Night 80s
Tomorrow Night Blue
Tomorrow Night Bright
Twilight
Vibrant Ink
Xcode
```

```
# plotting of R objects
plot <- function (x, y, ...)
{
  if (is.function(x) &&
        is.null(attr(x, "class")))
  {
    if (missing(y))
      y <- NULL

    # check for ylab argument
    hasylab <- function(...)
      !all(is.na(
        pmatch(names(list(...)),
              "ylab")))

    if (hasylab(...))
      plot.function(x, y, ...)

    else
      plot.function(
        x, y,
        ylab = paste(
          deparse(substitute(x)),
          "(x)"),
        ...)
  }
  else
    UseMethod("plot")
}
```

Figure 6: **Changing the RStudio skin**

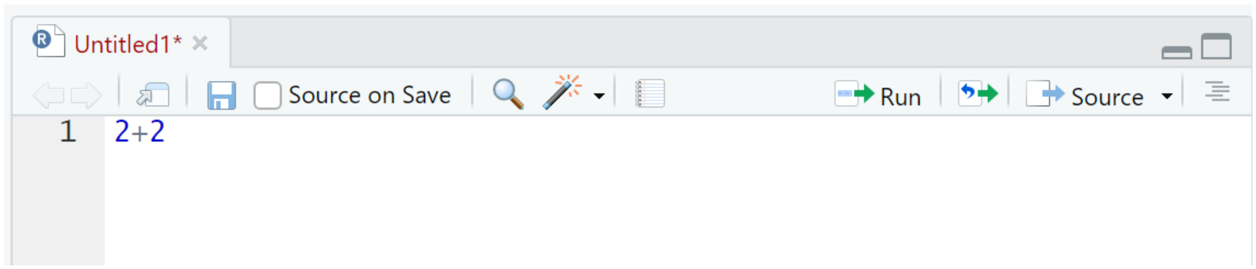
Start coding!

Go to the toolbar and start a new R script (`File - New File - R Script`).

In your Untitled1* code window, type in:

```
2+2
```

There are two ways to run this code. Press the `Run` button on the script toolbar. Make sure the cursor is "blinking" on the line you want to run.



```
2+2
```

Figure 7: **Run Code**

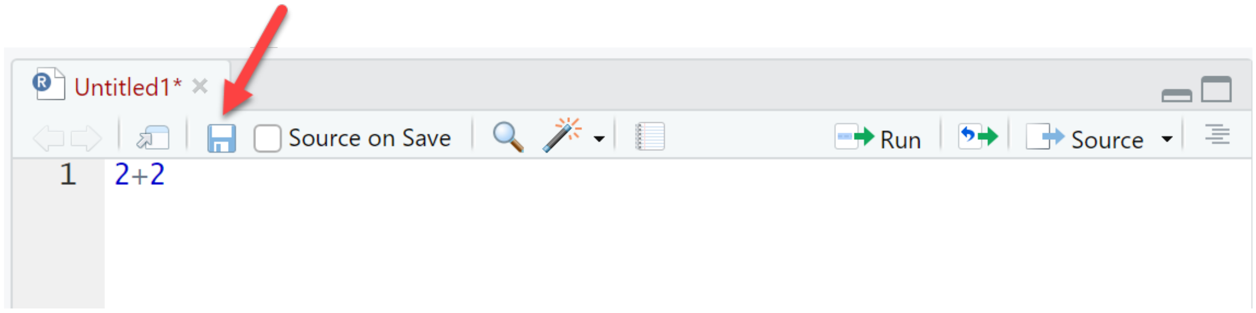
The other option is to use the keyboard, use Ctrl+Return keys on Windows, and Cmd + Return keys on macOS.

In the `Console` the output will now be dispalyed as:

```
[1] 4
```

Save code

To save the snippet of `R` code, just use the "diskette" icon, or from the toolbar: `File - Save`.



```
2+2
```

Always save your work, and use file names that are informative.

Working with `Objects`

The entities R operates on are technically known as objects. Some examples are vectors, matrices, lists, functions and data. Objects used and created in an `R` session is shown in the `Environment` pane.

Use the assignment `<-` operator to create an object using code.

```
a <- 1

b <- 2

a + b
```

```
[1] 3
```

We have now created two objects, labeled `a` and `b` with the respective values 1 and 2, and then we added them. Notice that an assignment does not print a value in the `Console`. Instead, we store it in our `R` session for later use in what we might call a variable.

You can also find the objects in a session in the `Console` using code, using the `ls()` function.

```
ls()
```

```
[1] "a" "b"
```

Note that each time you run code you can update an object, i.e. change it. Let's change `b` to 3.

```
b <- 3

a + b
```

```
[1] 4
```

One of the most used functions in `R` is `c()`. We we use it to `combine` objects or values:

```
c(a,b)
```

```
[1] 1 3
```

Or create a new object, a vector `ab`:

```
ab <- c(a,b)

ab
```

```
[1] 1 3
```

A simple sequence of numbers, from zero to nine, i.e., a vector.

```
x <- 0:9

x
```

```
 [1] 0 1 2 3 4 5 6 7 8 9
```

Sum of the vector `x`.

```
sum(x)
```

```
[1] 45
```

Important note!

Note that `R` is case sensitive! This means that there is a difference between code written using caps or not. Hence, `A` and `a` are different symbols and would refer to different objects. Also when giving names to objects, stick with plain letters and numbers.

Basic mathematical functions

Here are some basic mathematical functions.

Function	Expression	Code	Output
Addition	$2+2$	`2+2`	4
Subtraction	$2-2$	`2-2`	0
Multiplication	2×2	`2*2`	4
Division	$\frac{2}{2}$	`2/2`	1
Square root	$\sqrt{2}$	`sqrt(2)`	1.4142
Exponents	2^2	`2^2`	4
Euler's number	e	`exp(1)`	2.7183
Exponential function	e^2	`exp(2)`	7.3891
Natural logarithm	$ln(e^2)$	`log(exp(2))`	2
Pi	π	`pi`	3.1416
sine	$\sin()$	`sin(pi/2)`	1
cosine	$\cos()$	`cos(pi)`	-1
Absolute value	$\|-2\|$	`abs(-2)`	2
Factorial	$3!$	`factorial(3)`	6

Basic statistical functions

R contains wide range of statistical functions.

The arithmetic mean ($\bar{x}$) of x.

```
mean(x)
```

```
[1] 4.5
```

The median of x.

```
median(x)
```

```
[1] 4.5
```

The sample variance (σ^2) of x.

```
var(x)
```

```
[1] 9.166667
```

The standard deviation (σ) of x.

```
sd(x)
```

```
[1] 3.02765
```

Standard scores (z-scores) of x. For each value of x the mean of x is subtracted, and divided by the standard deviation of x, i.e. $z = (x - \bar{x})/\sigma$.

```
scale(x)
```

```
            [,1]
 [1,] -1.4863011
 [2,] -1.1560120
 [3,] -0.8257228
 [4,] -0.4954337
 [5,] -0.1651446
 [6,]  0.1651446
 [7,]  0.4954337
 [8,]  0.8257228
 [9,]  1.1560120
```

```
[10,]  1.4863011
attr(,"scaled:center")
[1] 4.5
attr(,"scaled:scale")
[1] 3.02765
```

The standard scores `z` has a mean of zero and a standard deviation of 1. Lets check this.

```r
z <- scale(x)
mean(z)
```

```
[1] 0
```

```r
sd(z)
```

```
[1] 1
```

Some summary statistics of `x`.

```r
summary(x)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   0.00    2.25    4.50    4.50    6.75    9.00 
```

Working with functions

There is an interesting story about the famous mathematician Carl Friedrich Gauss, who as an elementary student in the late 1700's, amazed his teacher with how quickly he found the sum of the integers from 1 to 100. Gauss recognized that he had 50 pairs of numbers summing to 101, when he added the first and last number in the series, the second and second-last number, and so on. For example: (1 + 100), (2 + 99), (3 + 98), creates 50 pairs that has a sum of 101. Hence, the sum of the integers from 1 to 100 is $50 \times 101 = 5050$.

There is a general formula for the sum of consecutive numbers: $n(n + 1)/2$. We will create our own function, called `gauss` that uses this formula. The `function()` uses `n` as an argument, and the function itself is inside the curly brackets `{}`. Here is our function:

```r
gauss <- function(n) {
  return(n*(n+1)/2)
  }
```

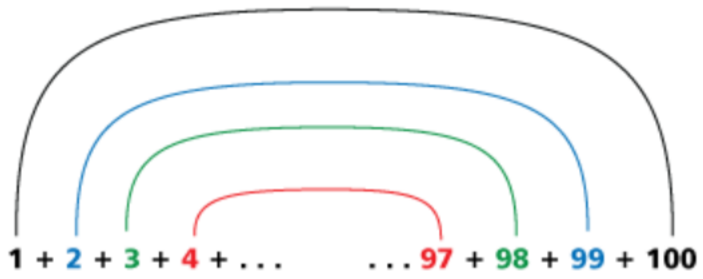


Figure 8: **Gauss' number pairs**

```
gauss(100)
```

```
[1] 5050
```

Note that we used `return()` to evaluate and define the function. The name we selected for the function, `gauss`, is arbitrary. We could have given it any name we like. However, it's a good practice to give functions names that are easy to read and write and remind you what they are about. We used `n` as an argument, we could have used `x`, `y` or any naming we preferr. Also, note that if you give a new function a name, you are allowed to give it any name, also a name of an existing `R` function. This means that you are allowed to "owervrite" any function in an `R`session! This flexibility is both a blessing and a curse, and is a base for much confusion.

Remember we used the function `pi` above to get the constant 3.1416. Now, let's rename our function above.

```
pi
```

```
[1] 3.141593
```

```
pi <- function(n) {
  return(n*(n+1)/2)
  }
pi(100)
```

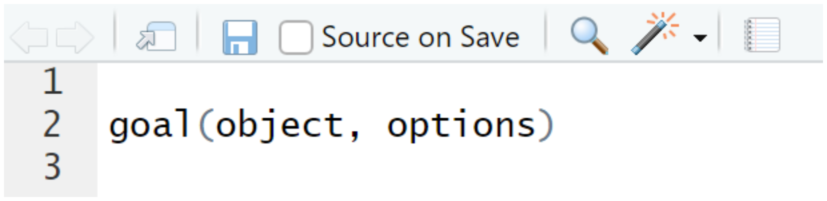
```
[1] 5050
```

```
pi
```

```
function(n) {
  return(n*(n+1)/2)
}
```

The `base R` function `pi` does not exist in this session any more! It has been replaced by our own function. You have to restart your `R` session to get back the default settings.

All functions in `R` is a verb with the *goal* of what we want to do with the *object* enclosed in parenthesis, using commas to separate *options*.



```
goal(object, options)
```

Figure 9: **Structure of `R` function**

Notice that our `Environment` now contains 7 objects, two values, `a` and `b`. Two vectors, `x` the integer values from 0 to 9, and `ab` the combination of `a` and `b`. We also have a data object `z` with the standardized values of `x`, and finally the `gauss` and `pi` functions that we created.

We can print the objects in our session in the console by using the `ls()` function.

```r
ls()
```

```
[1] "a"     "ab"    "b"     "gauss" "pi"    "x"     "z"
```

If we compactly would like to display the structure of an arbitrary `R` object, use the `str()` function.

```r
str(z)
```

```
 num [1:10, 1] -1.486 -1.156 -0.826 -0.495 -0.165 ...
 - attr(*, "scaled:center")= num 4.5
 - attr(*, "scaled:scale")= num 3.03
```

Shows that `z` is a 10 by 1 numerical Data object.

Extending R with packages

R ships with some basic functions called `base R`. However, it is extended by packages (read functions) written by collaborators all over the world. Compare this to your smartphone where you can download apps to get more functionality. Packages in `R` adds new functionality to `R`, and you have to load them ("start them") each time you need them.

Have a look at DataCamp's R Packages: A Beginner's Guide for an introduction to `R` packages. When this document was compiled (2020-08-11) there were 16031 packages available for download. There is a task view available at https://cran.r-project.org/web/views/. Some useful topics for economists would be econometrics, experimental design, finance, optimization and time series. But packages from other areas might be useful for a specific task.

What we need to know is how to load a packag in an `R` session, this one should suffice:

```r
library(package_name)
```

Note that when a package loads, you can see in the `Console` if there are some current function names that are written over by the new package you loaded. In `R` the last function loaded rules the old one's!

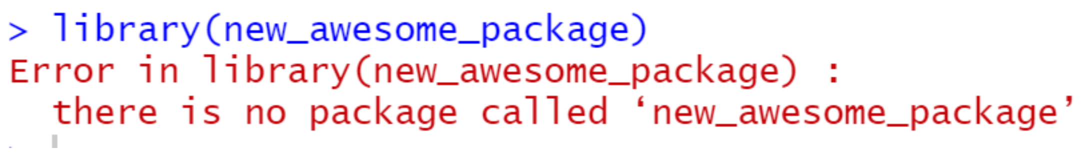
Note that there is a trick to write code that allows you to direct to the package and function directly, without loading it:

```r
package_name::function_nam(object)
```

Here, the two colons, `::` points to the function in a package, even when it isn't loaded in the current session. This is useful if you are using two packages in a session that overwrite each other functions.

Installing new packages

If you try to use a package that is not installed, you will get an error message in the console.



```
> library(new_awesome_package)
Error in library(new_awesome_package) :
  there is no package called ‘new_awesome_package’
```

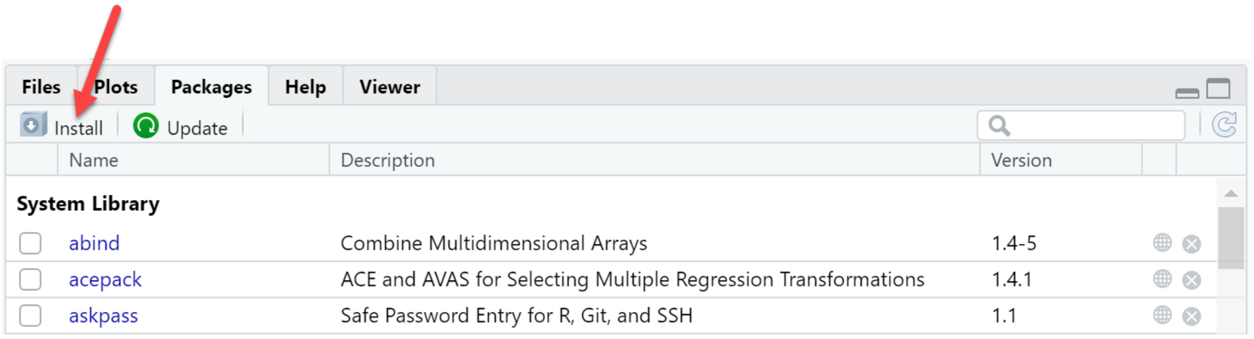
Figure 10: **Error message when trying to load uninstalled package**

The solution is to install the package you need by using the `install.packages()` function, and having the package name in brackets (" ").

```r
install.packages("new_awesome_package_name")
```

You can also use the `Packages` toolbar.

If you would like to know what packages are loaded in a specific session run:



	Name	Description	Version
	System Library		
☐	abind	Combine Multidimensional Arrays	1.4-5
☐	acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
☐	askpass	Safe Password Entry for R, Git, and SSH	1.1

Figure 11: **Using the Packages toolbar to administer packages**

```
sessionInfo()
```

The last thing you need to remember is that each time you restart R or start a new R session, you need to load the packages again. This seems cumbersome, but you will get used to it. A final bit of advice is to restart your R session each time you start on a new task. This clears the memory of your R session.

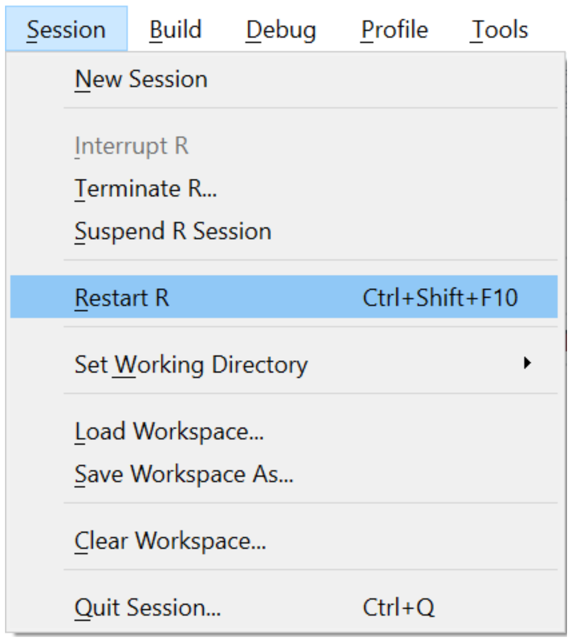


Figure 12: **Restart R session**

Reproducible research

The goal of reproducibility is the ability for your work to be easily recreated by others but most importantly for "future self." Code connects data and instructions in order to reproduce your analysis and results. Several packages in R are dedicatet to this purpose. They can be split into groups for: literate programming, pipeline toolkits, package reproducibility, project workflows, code/data formatting tools, format convertors, and object caching. There is also a task view for reproducible research.

The best way to accomplish reproducibility is to start scripting your analyses. Scripting is the practice of writing code to a file in order to perform certain tasks or calculations. Rather than producing "one-off" analysis, script your work so you can reference the exact method in the future, re-run the same method using other data, and easily share your processes with colleagues, collaborators and the public.

Research is considered to be reproducible when the exact results can be reproduced if given access to the original data, software, and code. We will be focusing on making the analysis we do reproducible through R scripts. A fun "horror" movie about reproducibility can be found at "Non-reproducible workflows: a horror movie".

Some general guidelines are:

- script all your work
- comment (use `#` first) the steps in your code, especially for "future self"
- limit the number of packages being used when possible
- keep up to date with `R` software versions and package updates

Tips:

To find the path to your current `R` session, use:

```r
getwd()
```

To change the path to the folder you would like in `rstudio.uit.no`, use:

```r
setwd("~/path/to/folder")
```

If your are running RStudio on your laptop use:

```r
setwd("C:/path/to/folder")
```

More on reproducible research here and here.

Exercises

Write `R` code that solves:

1) $8/2(2+2)$

2) Let `x` be a vector of the numbers from 1 to 9. Create a new vector `zx` of the standardized values of `x`, using the `mean()` and `sd()` functions.

3) Create a new function, call it `znew` that creates the standardized values of the vector `x`, using the `mean()` and `sd()` functions.

```r
getwd()

knitr::purl("01_functions.Rmd")
```