

Computational Reproducibility

Technology has greatly improved how scientists work. The internet has made it easy to share information – including data, materials, and code – and new software and online platforms have emerged to facilitate the workflow of scientists. One important goal of a scientific workflow is to make sure that the final work you publish is computationally reproducible. **Computational reproducibility** means that when you use the **same data** as in the published article, you can reproduce the **same results**. We could consider this a minimum standard of any workflow. However, meeting this standard requires training. When I was a Ph. D. student, we had a problem known as ‘data rot’. When I submitted an article for publication and received the reviews after several months, I could not always easily reproduce my own analyses. Sometimes, ‘data rot’ had eaten away at either my data or my analysis code, and it no longer worked.

Obviously, there is no such thing as ‘data rot’. The problem was that I had not recorded the exact steps I used to analyze my data, and I could therefore not reproduce my data analysis. In this assignment, you will learn what a computationally reproducible workflow looks like, and how you can share computationally reproducible results with your published paper. The goal of this assignment is for someone else (or for yourself, one year from now) to be able to take your data, run your code, and get exactly the same results as you reported in your work.

Although there are multiple ways to achieve a fully reproducible workflow, in this assignment I will introduce you to what I believe is one emerging standard for a fully reproducible workflow. You will set up a GitHub account, create a new GitHub repository, link this repository to RStudio, and learn how to use version control to save the history of all changes you've made while working on your files. This will allow you to use a version control system as you are programming in R, which stores previous versions of files. You will then create an R Markdown document, which allows you to write a completely reproducible data analysis script (including figures), that you can export as an HTML or PDF file (and we will briefly discuss how to create APA formatted documents in a new R package called papaja). You will then learn how to archive your GitHub repository – data, code, and any other files – on the Open Science Framework, and link to these files in the final version of your manuscript. Finally, we will take a look at Code Ocean, a novel online platform that allows you to share computationally reproducible code online, making it easy for others to run your code. You will not learn how to become an experienced programmer in this assignment – that would require substantially more time. But you will experience what a fully reproducible workflow looks like. After this assignment, you will still need a lot of training. But improving your programming skills and learning a fully reproducible workflow will be worth it!

Getting software and code to work on your system might be a challenge, and regrettably, I can't offer ICT support. Differences between Windows, Linux, and Apple operating systems means that you might need to search the internet for solutions to problems you run into. Software will update, packages update, and different people

might encounter different errors at different times. **For this reason, this assignment will live online in a google doc, and I would appreciate it if you could provide feedback about any errors you experience.**

If you get stuck, you can check what you did against what the assignment should look like by visiting the public versions of part of this assignment:

GitHub repository: https://github.com/Lakens/reproducibility_assignment,

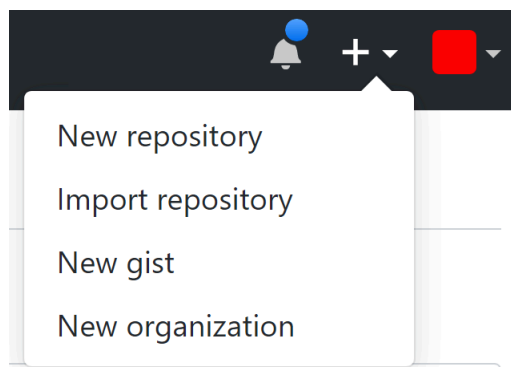
OSF project: <https://osf.io/jky8s/>

Code Ocean: [Capsule](#)

Step 1: Setting up a GitHub repository

If you haven't created a GitHub account before, do so now. Go to <https://github.com/> and create an account. Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. GitHub is a web-based hosting service for version control using Git. Git is free open source software. GitHub is a company that allows people to easily use Git. Other ways to use Git exist, but GitHub is currently the most widely used (and it's free).

Once you have an account, you can create a new repository. A **repository** is a collection of folders and files that make up your project. In the top-right of the GitHub page, click the + symbol, and select 'New repository' from the dropdown menu.





The first thing to do is name your repository. When it comes to naming folders and files, it is important to follow **best practices for file naming**:

- Keep names short (because you will have to type less), but clear (so others will understand the names). "data_analysis_project" is easier to understand for others than "dat_an_prjct".
- Do not use spaces (spaces in variable names make it more difficult to refer to variables in your code). Options include:
 - o Points: this.is.a.file.R (recommended in most code style guides)
 - o Underscore: this_is_a_file.R (this is my personal favorite despite of what style guides say)
 - o Camelcase: ThisIsAFile.R
 - o Dashes: this-is-a-file.R
 - o No spaces: thisisafire.R

- If you want to number multiple sequential files, do not use 1_start, 2_end, but use leading zeros whenever you might number more than 10 files, so for example 01, 02, etc., or 001, 002, etc.
- Do not use special characters such as \$#&*{}: in file names.
- If you want to use date information, use the YYYYMMDD format.

If you are interested in using a consistent style when writing code, you might want to look at code style guides (such as [Google's R Style guide](#)).

Let's name our repository: reproducibility_assignment

Owner	Repository name
 Lakens ▾	/ reproducibility_assignment 

You can add a short description (e.g., 'This is an assignment to practice an open and reproducible data analysis workflow'). You can make the repository public or keep it private (if you use the academic or paid version- if you are a researcher you can get a [free developer plan](#)).

Click the checkbox before 'Initialize this repository with a README'. A readme file is a useful way to provide a more detailed description of your project, that will be visible when people visit your GitHub project page.

You will also be asked if you want to add .gitignore file (you can ignore this for now) and a **license**. Adding a license is a way to easily communicate how other people can use the data, code, and materials that you will share in your GitHub repository. Note that not making a choice about a license, is also a choice: if you do not add a license your work is under exclusive copyright by default. You can [learn more about licenses](#), but for now, a simple choice is the MIT license, which puts only very limited restrictions on reuse, but more restrictive licenses also exist. You can select the choice of license (such as the MIT license) from the dropdown menu. It lets people do anything they want with your code as long as they provide attribution back to you and don't hold you liable. There are also [creative commons licenses](#) that you can use when you are sharing something else than software, such as research materials (for example, this educational material is shared under a CC-BY-NC-SA 4.0 license).

[Create repository](#)

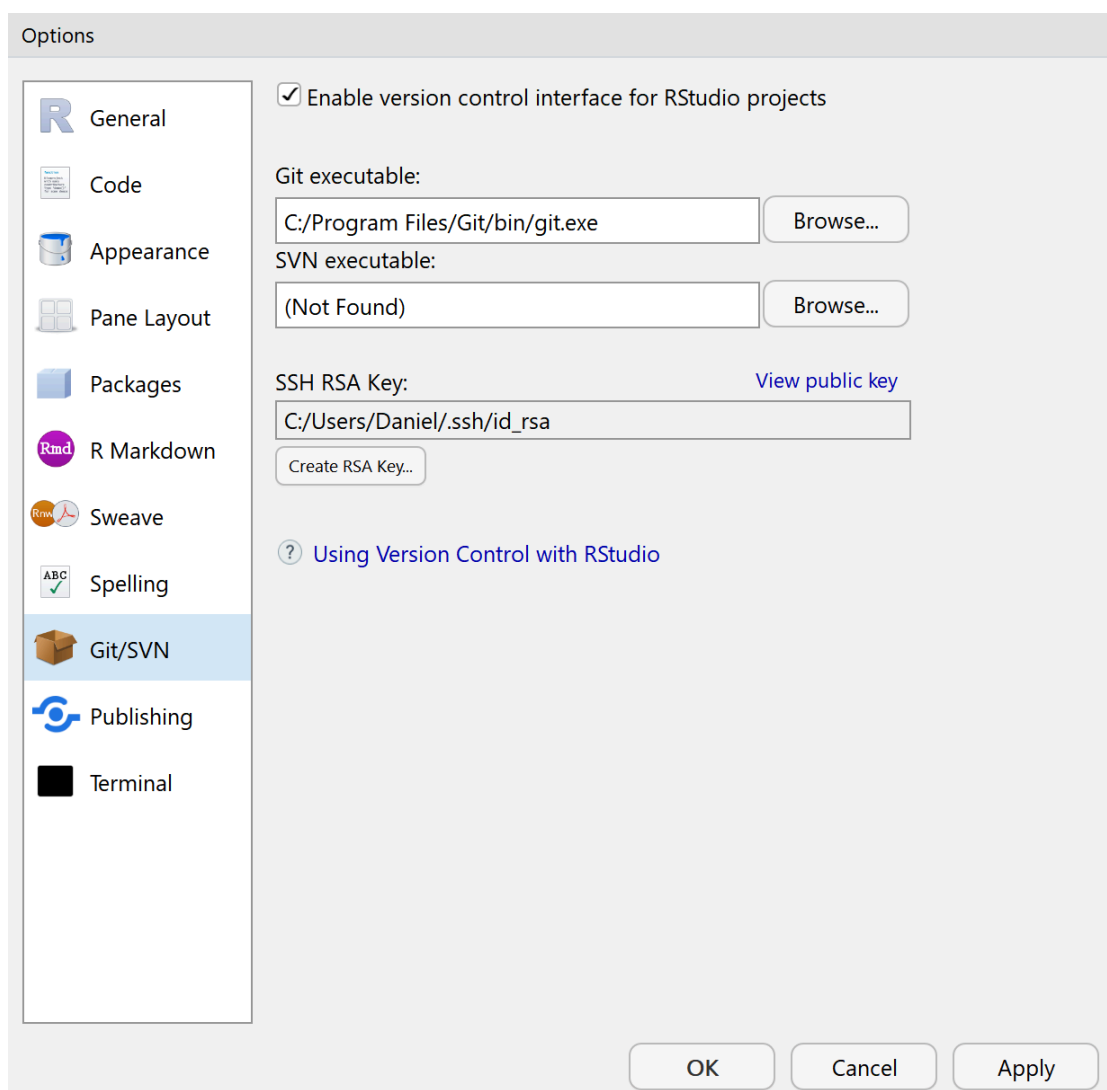
You are now ready to create the repository. Click

It might feel unintuitive, but it is important to remember that you are not expected to directly interact with your new GitHub repository through the GitHub website. The repository page will give you information about the contents of the repository, and the history of the files in the repository, but it is not that easy to add files or download files directly through the website. You will use other software to interact with your GitHub repository.

Step 2: Cloning your GitHub repository into RStudio

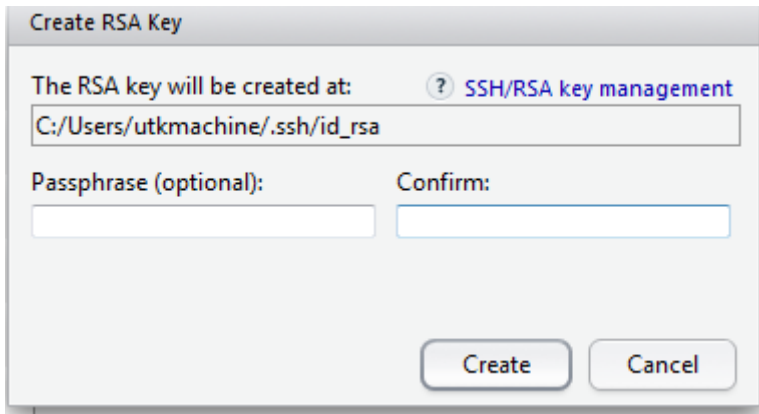
To allow RStudio to work together with GitHub, you first need to set up the system. A detailed explanation for different operating systems is provided [here](#). First, download Git: <https://git-scm.com/downloads> for your operating system, and install it (you can accept all defaults during the installation process). If you haven't done so already, download and install R: <https://cran.r-project.org/>, and download and install the free version of RStudio (scroll down for the installers): <https://www.rstudio.com/products/rstudio/download/>.

In RStudio, go to Tools > Global Options, and select the Git/SVN menu option.



Check if the Git executable has been found automatically. If not, you will need to click the 'Browse...' button and find it manually.

Click the 'Create RSA Key...' button. A window will appear, and you can click 'Create':

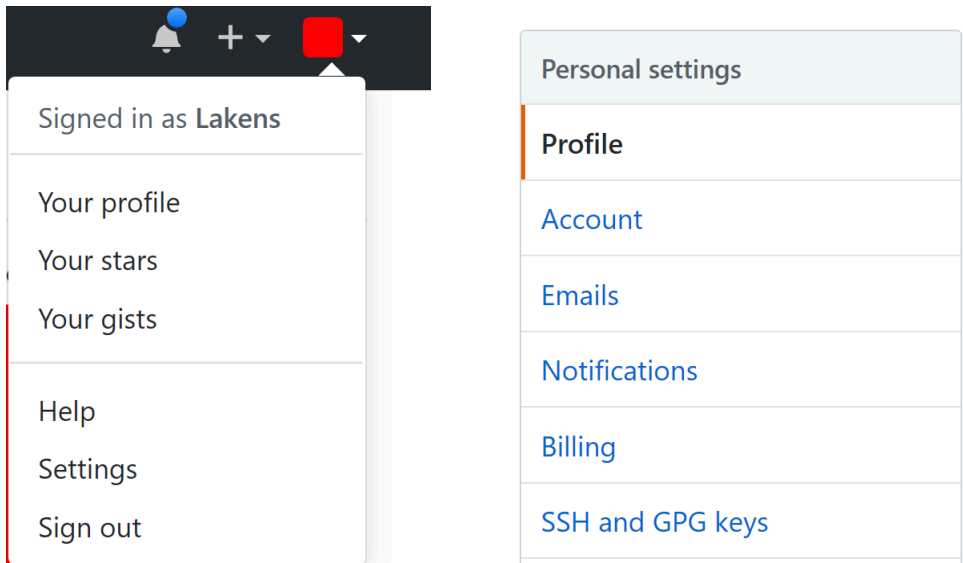


A new window will open:

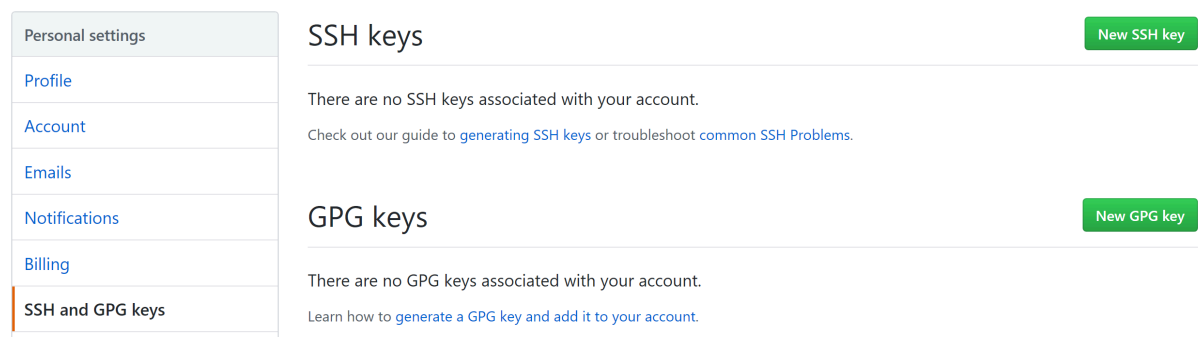


You can close the window. Click the blue hyperlink 'View public key'. A window will appear, telling you that you can use CTRL+C to copy the key. Do so.

Go to GitHub, and go to settings and then select the option SSH and GPG keys:

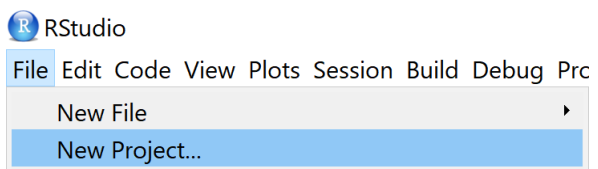


Click 'New SSH key'

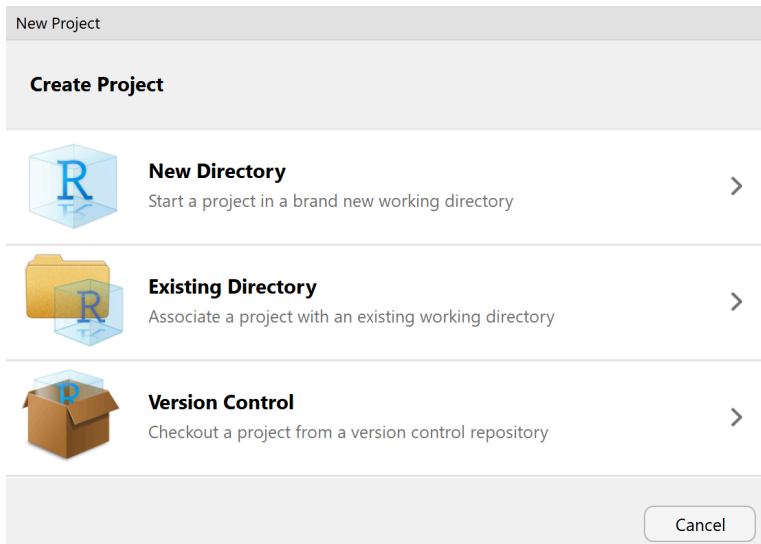


Enter a name (e.g., RStudio) and paste the key in the correct window. Click 'Add SSH Key'. This will allow you to push code to GitHub repositories without having to enter your GitHub login name and password every time. You are now ready to create a **version controlled project** in RStudio.

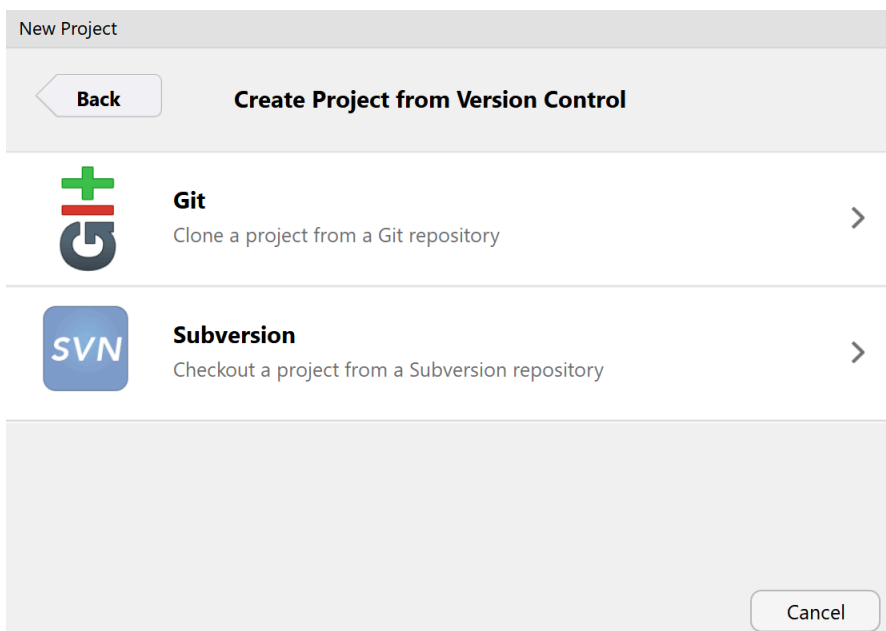
In RStudio, go to File>New Project:



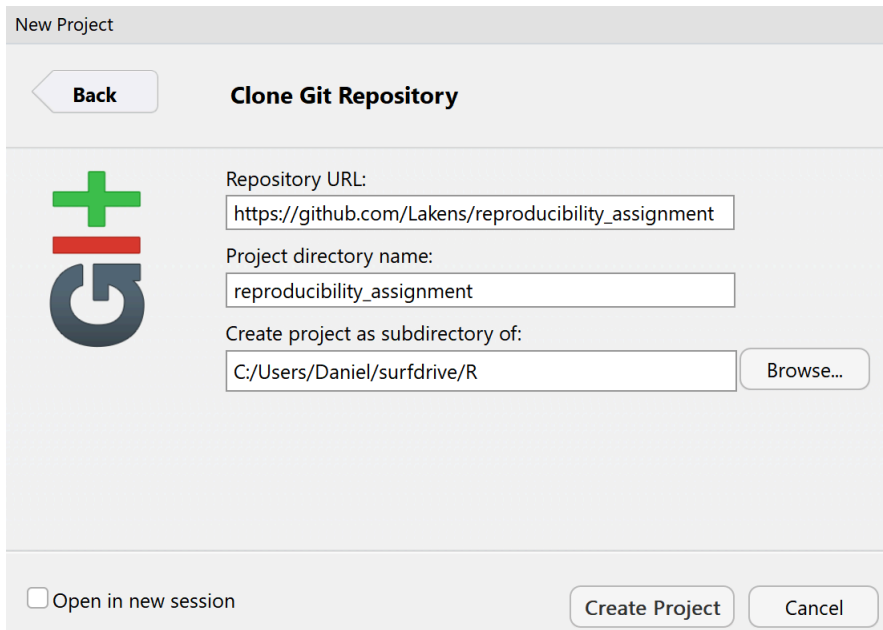
You will see three choices. Choose the 'Version Control' option:



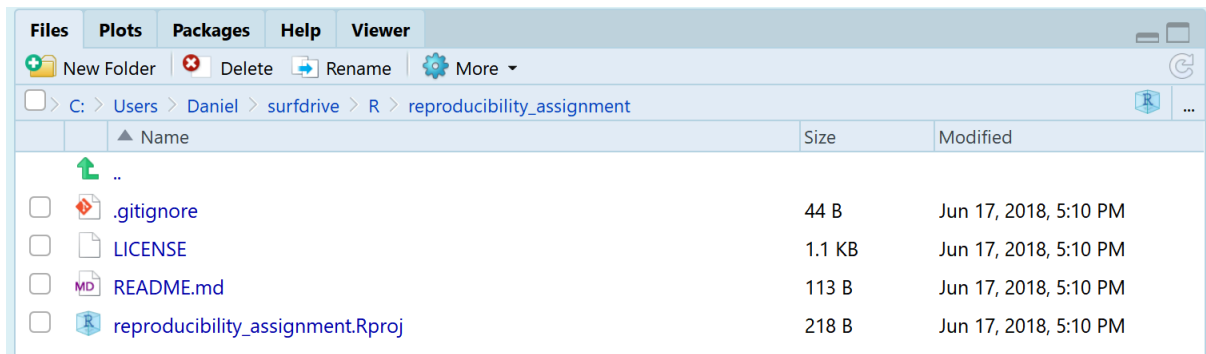
Choose the 'Git' option:



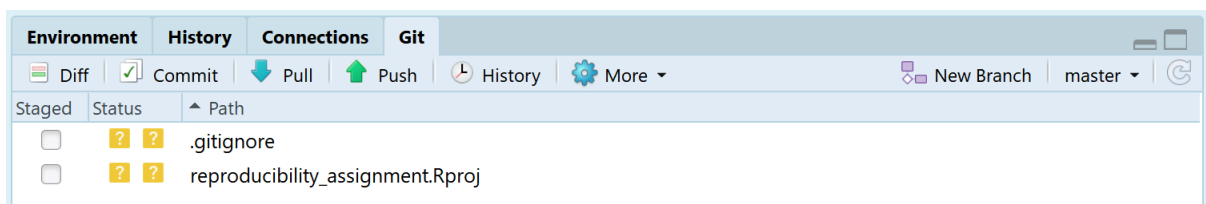
We will be cloning the online GitHub repository you created, which will create a local copy of all files. You can copy-paste the URL from your GitHub repository (e.g., https://github.com/Lakens/reproducibility_assignment). If you copy-paste this URL in the top field, this will automatically create a Project directory name that is similar to the name you gave your project on GitHub. You can select a folder on your computer by clicking the 'Browse' button to indicate where you want to save the local copy of your repository.



Click 'Create Project'. R will quickly download some files, and open the new project. You will see that the project creation was successful because the 'Files' tab in the RStudio interface shows that you have downloaded some files from our GitHub repository (the README.md and LICENSE files). RStudio also created a reproducibility_assignment.Rproj file and a .gitignore file. The **project file** is used to store information about the project, and that is required to use GitHub.



We can also see this is a version control project in the top right of the interface, where there is now a 'Git' tab. If you click it, you will see:



We see a range of buttons, such as the Diff, Commit, Pull, and Push buttons. These are used to interact with GitHub. Many computer programmers interact with GitHub through the command line, such as:

```
$ git commit -m "This is a git commit message"
```

Learning to use git through the command line is not needed for most people who just want basic version control. In this assignment I will explain a reproducible workflow that does not require you to use the command line, because everything can be done using menu options in RStudio. To make the analysis code reproducible, we will use a R Markdown file to analyze our Stroop data.

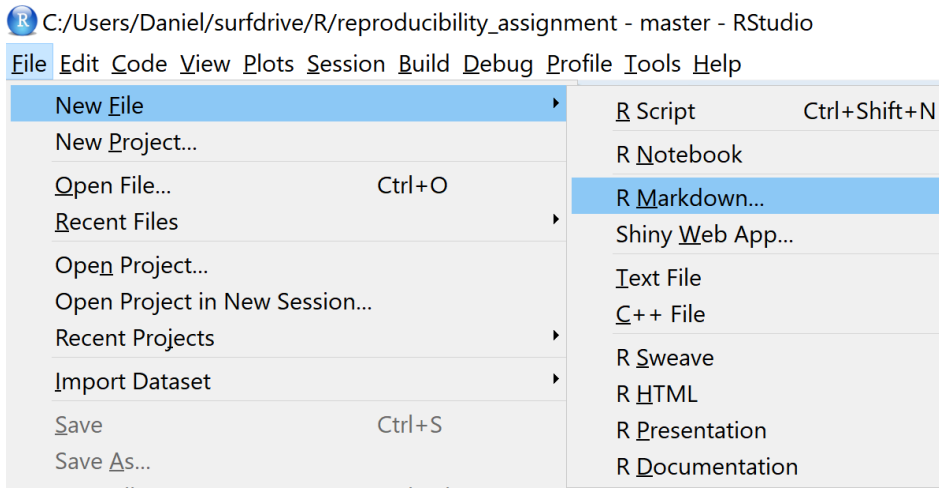
Step 3: Creating an R Markdown file

R Markdown files provide a way to save and execute code, while also allowing you to create reports of your data analysis (and even full scientific articles that you can submit for publication!). A complete introduction to R Markdown is [available here](#), and a cheat sheet is [available here](#). The main strength of R Markdown is that it allows you to create a fully reproducible document. This means that you can compile the document into a HTML file or PDF document that people can read like normal text. The R Markdown file also contains code that performs the analyses each time the document is compiled. Instead of copy-pasting values from your analysis software into a word document, you combine code and text to create a manuscript where every number or figure can be traced back to the exact code that generated it.

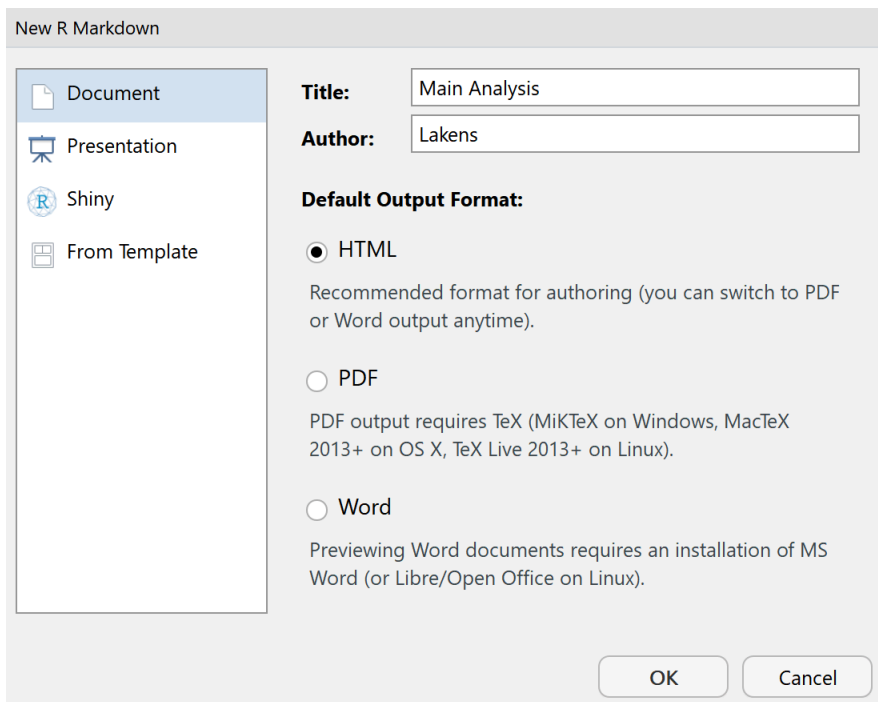
You can still make errors in the analysis if you use R Markdown files. The important difference is that your programming errors will be stored in the R Markdown document. Because the document is reproducible, all errors are reproducible as well. It is impossible to prevent all errors, but it is possible to make them reproducible. This will make it easier to identify and correct errors. I understand that you might worry about others seeing your errors if you allow them to see exactly what you have done. But we all make mistakes, and it is important for science to be able to identify and correct these mistakes. An important aspect of moving to a more reproducible workflow is learning to **accept that we all make errors** (and we should give credit to the people who actively try to correct their mistakes).

Let's start by creating a new R Markdown document in RStudio by clicking New File > R Markdown...

If this is the first time you are using R Markdown, you will be asked to install packages R requires. RStudio knows which packages it needs to start using R Markdown, so install all packages that are recommended.



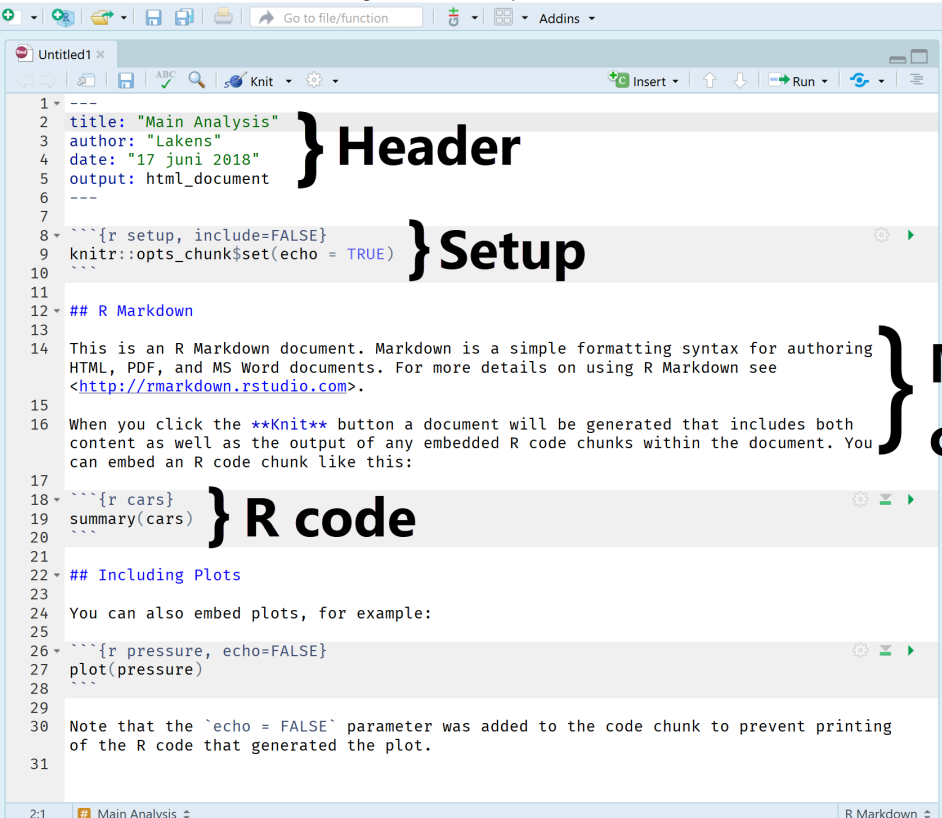
This gives you a new window where you can specify the title of your RMarkdown document, and an author name. Enter the title 'Main Analysis', and feel free to change the Author subfield in anything you prefer. RMarkdown files can be converted (or 'knitted') into a html file, a PDF document, or a word document. To generate PDF files you need to install MiKTeX which we won't do for this assignment ([a good tutorial how to install MiKTeX is available here](#)). So leave the default output format to HTML



Let's start by saving the new file: Click CTRL+S, and save the file under the name 'main_analysis.Rmd'. Because you are working in an RStudio project, the file will automatically be saved in the same folder as all other files in this project. If you look at

the files tab in the bottom right pane, you will see the new file appear. Now let's take a look at the R Markdown file.

The R Markdown file by default includes several sections to get you started. First, there is a header section. In the header section, there is code that determines how the final document is rendered. This section is sensitive, in the sense that it needs to be programmed exactly right – including spaces and tabs – so it is not recommended to change it too much without looking up detailed documentation on how to do so. An R Markdown file is fed to knitr software, which creates a normal markdown file, which then uses pandoc software to generate the specific document you requested. All of this happens automatically.




The screenshot shows the RStudio interface with an R Markdown file open. The file content is as follows:

```
1 ---
2 title: "Main Analysis"
3 author: "Lakens"
4 date: "17 juni 2018"
5 output: html_document
6 ---
7
8 ```{r setup, include=FALSE}
9 knitr::opts_chunk$set(echo = TRUE)
10 ```
11
12 ## R Markdown
13
14 This is an R Markdown document. Markdown is a simple formatting syntax for authoring
15 HTML, PDF, and MS Word documents. For more details on using R Markdown see
16 <http://rmarkdown.rstudio.com>.
17
18 When you click the Knit button a document will be generated that includes both
19 content as well as the output of any embedded R code chunks within the document. You
20 can embed an R code chunk like this:
21
22 ```{r cars}
23 summary(cars)
24 ```
25
26 ## Including Plots
27
28 You can also embed plots, for example:
29
30 ```{r pressure, echo=FALSE}
31 plot(pressure)
32 ```
33
34 Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing
35 of the R code that generated the plot.
```

Large black curly braces on the right side of the code editor group the sections into four categories:

- Header**: Lines 2-5
- Setup**: Lines 8-9
- Markdown code**: Lines 14-16
- R code**: Lines 18-20

The header is followed by a set-up section where you can define general options for the entire file. Then, we see the two main sections Markdown code and R Code. **Markdown code** is a markup language in plain text formatting syntax that can be easily converted into HTML or other formats. **R code** is used to analyze data or create figures. To see the final result of this code, hit the  Knit button in the toolbar at the top of the pane.

A new window will appear that allows you to view the HTML file that was created. You see the formatted HTML document that combined both text and the output of R code.

C:/Users/Daniel/surfdive/R/reproducibility_assignment/main_analysis.html

main_analysis.html Open in Browser Find Publish

Main Analysis

Lakens
17 juni 2018

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.


When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed      dist
## Min.   : 4.0    Min.   : 2.00
## 1st Qu.:12.0    1st Qu.: 26.00
## Median :15.0    Median : 36.00
## Mean   :15.4    Mean   : 42.98
## 3rd Qu.:19.0    3rd Qu.: 56.00
## Max.   :25.0    Max.   :120.00
```

Including Plots

You can also embed plots, for example:



The plot shows a single data point represented by a small circle at the top right corner of a rectangular frame. The y-axis is labeled with the number 800.

Close the window – we are now ready to analyze our data.

Step 4: Reproducible Data Analysis in RStudio

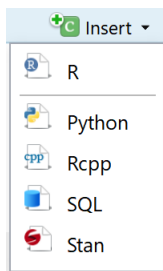
Delete all text from `##` R Markdown on down – only keep the header and set-up sections of the default document.

First, we need to analyze some data. We will download this data directly from an existing GitHub repository I created. Students in an introduction to psychology course performed a simple Stroop experiment, and named the colors in a congruent trial (e.g., the word 'red' written in a red font) and incongruent trial (e.g., the word 'red' written in a green font). The time they took to name all words was recorded in seconds (e.g., 21.3

seconds) for both the congruent and incongruent trial. There are four columns in the dataset:

- Participant Number
- Response Time for Congruent Stimuli
- Response Time for Incongruent Stimuli
- Year of Data Collection

Click the button '+C Insert' to insert code – a drop down menu will be visible. Select R.



In the R Markdown file, you'll see a new section of R code that start with `{r}` and end with `}```. You can also just create these sections be manually typing in these two lines.`


Copy-paste the code below – make sure to get all the text – and paste in between the start line and the end line of the R code chunk. Note that there are 2 lines of code (starting with `stroop_data` and `write_table`) and there are no hard returns or breaks – when copy-pasting these lines of code, you might need to correct this depending on your device.

```
stroop_data <-  
read.table("https://raw.githubusercontent.com/Lakens/Stroop/master/stroop.  
txt", sep = "\t", header = TRUE)  
  
write.table(stroop_data, file = "stroop.csv", quote=F, row.names=F)
```

After copy-pasting the text, the code should look like the screenshot below (again, be aware of any difficulties when copy-pasting text from a PDF file into RStudio):

A screenshot of R code in a code chunk. The code is enclosed in a light gray box with a header `{r}` and a footer `}```. The code inside is:

```
stroop_data <- read.table("https://raw.githubusercontent.com/Lakens/Stroop/master/stroop  
p.txt", sep = "\t", header = TRUE)  
  
write.table(stroop_data, file = "stroop.csv", quote=F, row.names=F)  
  
}```
````

This code creates a data.frame called 'stroop_data' that contains data, and then saves this data in a .csv file called 'stroop.csv'. Click the Knit button:  Knit to look at the document. You should see something like:

Main Analysis

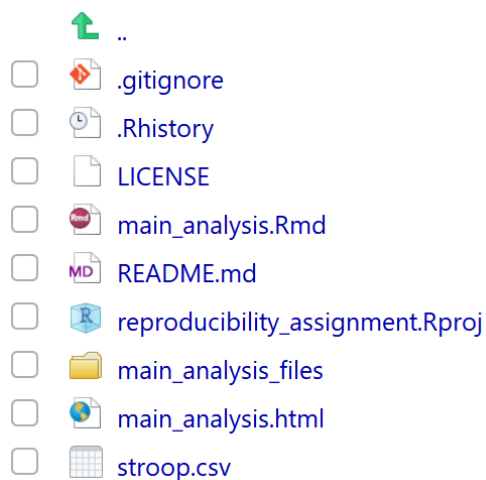
Lakens

17 juni 2018

```
stroop_data <- read.table("https://raw.githubusercontent.com/Lakens/Stroop/master/stroop.txt", sep = "\t", header = TRUE)

write.table(stroop_data, file = "stroop.csv", quote=F, row.names=F)
```

This might not look very impressive – but the real action is in the file pane in the bottom right part of the screen. Close the window showing the HTML output and look at the file pane. You should now see a bunch of files:



One file is stroop.csv – this is our data file of the Stroop data, that we downloaded from the internet, and saved to our project folder, using R code.

There is really no need to keep downloading the file from the internet when we can also just load it from the local folder. So let's change the code. We won't completely delete this code – we will just **comment it out** by placing a # in front of it. This way, we can still remember where we downloaded the code from, but we won't use the code.

Because it is always important to **provide comments in the code you write**, add the explanation:

```
#run only once to download the data
```

above the line where we downloaded the code. Then, select the lines of code in the chunk, and click CTRL+SHIFT+C (or COMMAND+SHIFT+C on a mac). This should add # in front of all lines, making it comments instead of code that is executed every time. You should end up with:

```

```{r}
#run only once to download the data
stroop_data <- read.table("https://raw.githubusercontent.com/Lakens/Stroop/master/stroop.txt", sep = "\t", header = TRUE)
#
write.table(stroop_data, file = "stroop.csv", quote=F, row.names=F)
```

```

Now we need to add a line of code that we will run, and with which we will load the stroop.csv dataset from the local folder. Underneath the last commented out line of code, but within the R code block, add:

```
stroop_data <- read.csv("stroop.csv", sep = " ", header = TRUE)
```

```

```{r}
#run only once to download the data
stroop_data <- read.table("https://raw.githubusercontent.com/Lakens/Stroop/master/stroop.txt", sep = "\t", header = TRUE)
#
write.table(stroop_data, file = "stroop.csv", quote=F, row.names=F)

stroop_data <- read.csv("stroop.csv", sep = " ", header = TRUE)
```

```

Click CTRL+S to save the file. Knit the file. We see:

Main Analysis

Lakens

17 juni 2018

```

# #run only once to download the data
# stroop_data <- read.table("https://raw.githubusercontent.com/Lakens/Stroop/master/stroop.txt",
#   sep = "\t", header = TRUE)
#
# write.table(stroop_data, file = "stroop.csv", quote=F, row.names=F)

stroop_data <- read.csv("stroop.csv", sep = " ", header = TRUE)

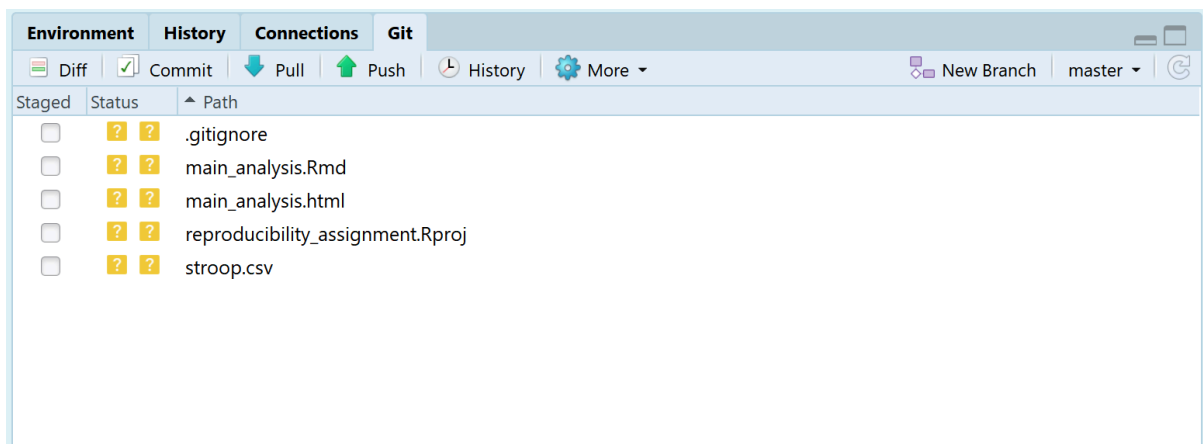
```

Close the HTML file. We've done quite a lot of work. It would be a shame if this work was lost. So this seems to be the perfect time to save a version of our R Markdown file, not just locally, but also on GitHub.

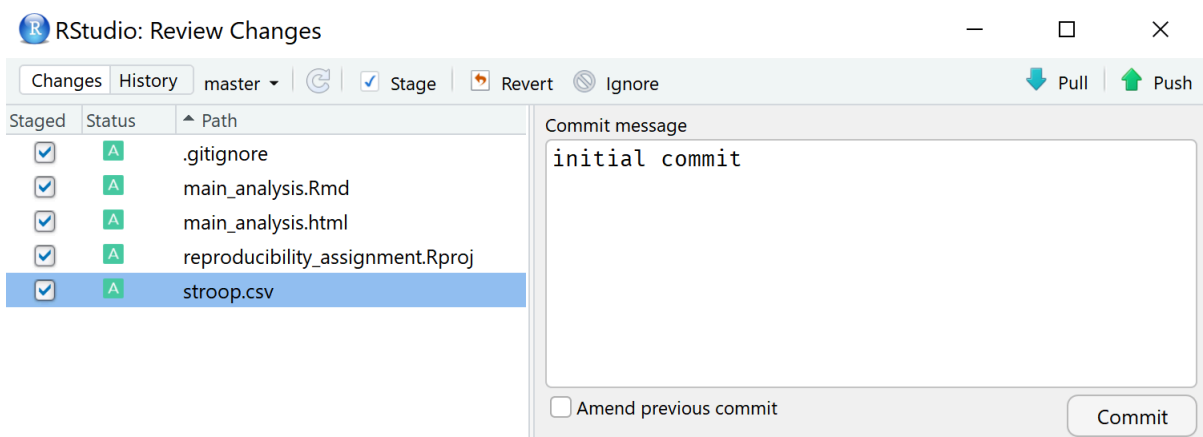
Step 5: Committing and Pushing to GitHub

It is time to store our changes in the cloud, on GitHub. This process takes two steps. First, we record the changes to the repository. This is called a **'commit'**. You don't need an internet connection to 'commit' files because we are just recording the changes locally. However, then you want to make sure that the updated files and recorded changes are also stored on GitHub. This will require you to **push** the files to GitHub.

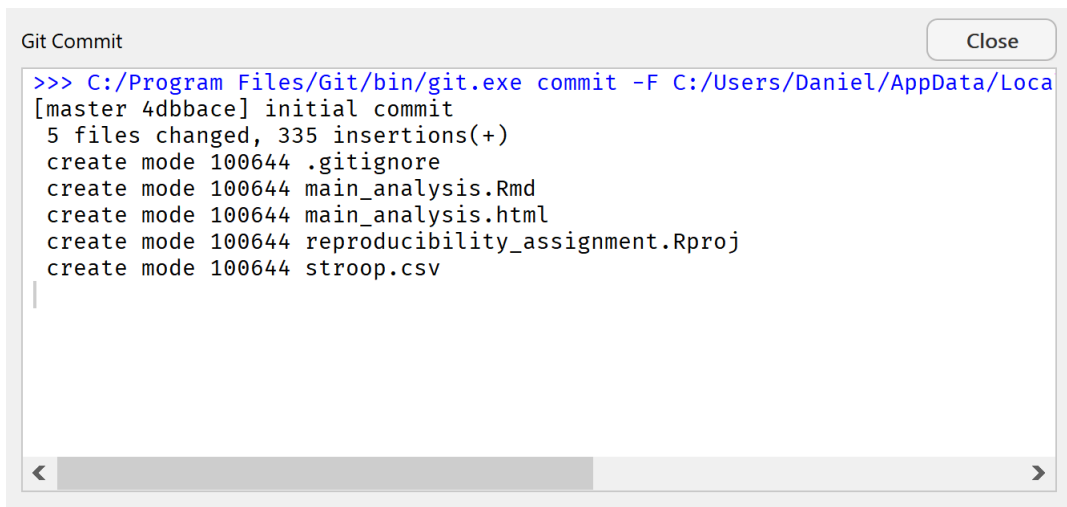
If we look at the Git tab in the top right pane in RStudio, we see the Commit button, the Push button, and a bunch of files. The status of these files is indicated by two question marks in yellow. These question marks indicate these files are not yet tracked by GitHub. Let's change this.



Click the commit button. A menu opens. You can choose to 'stage' the changes that have been made (this prepares the files for the commit, and is an intermediate step that provides some detailed possibilities that you typically do not need). You can do this in several ways, such as double clicking each file, or selecting all files and clicking 'Enter'. When staging all files, the yellow question marks change to a green 'A' symbol. Every commit should be accompanied by a **commit message** where you describe which changes you have made. You can type in an informative message about what you have changed in the code (the first time, 'initial commit' is often used). The menu should look like the screenshot below:

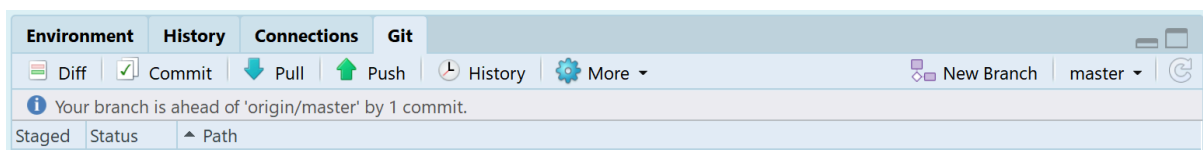



Now we are ready to **commit** these changes. Click the 'Commit' button. A new window opens that shows all changes that have been committed¹. We see that 5 files have changed. You can close this window and close the previous menu.



```
Git Commit Close  
  
>>> C:/Program Files/Git/bin/git.exe commit -F C:/Users/Daniel/AppData/Local/Programs/RStudio/bin/gitignore  
[master 4dbbace] initial commit  
5 files changed, 335 insertions(+)  
create mode 100644 .gitignore  
create mode 100644 main_analysis.Rmd  
create mode 100644 main_analysis.html  
create mode 100644 reproducibility_assignment.Rproj  
create mode 100644 stroop.csv
```

RStudio now reminds you that there is a difference between the local copy of your repository, and the remote version of the repository on GitHub. In the Git tab you see a reminder: "Your branch is ahead of 'origin/master' by 1 commit."



This means the last commit is not yet synchronized with the **remote repository**, something that can be solved by 'pushing' the changes to the remote repository. Simply click the **push** button:  Push². Another pop-up window appears:



```
Git Push Close  
  
>>> C:/Program Files/Git/bin/git.exe push origin refs/heads/master  
To https://github.com/Lakens/reproducibility_assignment  
9a2eaad..4dbbace master -> master
```

¹ One possible error at this point is GitHub asking you who you are. A solution (which requires using the command line: <http://www.thecreatedev.com/solution-github-please-tell-me-who-you-are-error/>)

² It is possible that you still need to enter your GitHub username and password at this point

This window informs us there were no errors, and we successfully pushed the changes to the remote version of the repository. You can close this window.

You can check that you successfully pushed all files to GitHub by visiting the GitHub page for your repository in the browser. You should see something like:

The screenshot shows the GitHub interface for a repository named 'Lakens / reproducibility_assignment'. At the top, there are navigation tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. Below the repository name, there is a description: 'This is an assignment to practice an open and reproducible data analysis workflow'. The repository statistics show 2 commits, 1 branch, 0 releases, 1 contributor, and the MIT license. A table lists the files in the repository, including .gitignore, LICENSE, README.md, main_analysis.Rmd, main_analysis.html, reproducibility_assignment.Rproj, and stroop.csv, all of which were committed initially.

| File | Commit | Time |
|----------------------------------|----------------|---------------|
| .gitignore | initial commit | 6 minutes ago |
| LICENSE | Initial commit | 5 hours ago |
| README.md | Initial commit | 5 hours ago |
| main_analysis.Rmd | initial commit | 6 minutes ago |
| main_analysis.html | initial commit | 6 minutes ago |
| reproducibility_assignment.Rproj | initial commit | 6 minutes ago |
| stroop.csv | initial commit | 6 minutes ago |

Congratulations on your first GitHub push! If you want to read a more extensive introduction to Git, you can read this [tutorial paper](#) by Matt Vuorre and James P. Curley.

Step 6: Reproducible Data Analysis

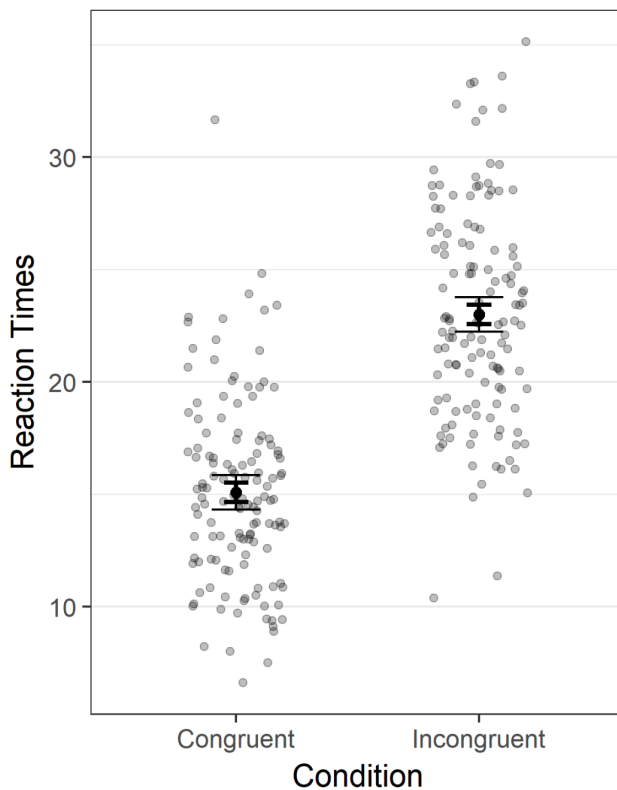
So far, we have only read in data. The goal of an R Markdown file is to create a manuscript that contains a fully **reproducible data analysis**. In this assignment, I cannot teach you how to analyze data in R (but I can highly recommend learning it – there are plenty of excellent online resources). Instead of programming from scratch, visit https://raw.githubusercontent.com/Lakens/Stroop/master/main_analysis.Rmd. This is a raw text version of an R Markdown file that will analyze the Stroop data. In the website, select all text (CTRL+A), copy it (CTRL+C). Then go to your main_analysis.Rmd file in RStudio. Select all text (CTRL+A) and hit delete. That's right – delete everything. You don't need to worry about losing anything – you have a **version controlled file** in your GitHub repository, which means you can always go back to a previous version! In the (now empty) main_analysis.Rmd file, press CTRL+V and paste all text. The file should look like the screenshot below.

This R Markdown file does a number of things, which we will explain in detail below. For example, it will automatically install libraries it needs, load the data, and create a report in HTML. You can hit the Knit button, and the HTML document should load. You should see output as in the second screenshot below.

```
main_analysis.Rmd x
1 ---
2 title: "Main Analysis"
3 author: "Lakens"
4 date: "17 juni 2018"
5 output: html_document
6 ---
7
8 {r global_options, echo=FALSE, warning=FALSE, message=FALSE,
include=FALSE}
9 knitr::opts_chunk$set(echo=FALSE, warning=FALSE, message=FALSE,
include=TRUE)
10
11
12 #Introduction
13
14 Here, we analyze a simple dataset of a Stroop experiment. Students in an
introduction to psychology course completed an online Stroop task
(http://faculty.washington.edu/chudler/java/ready.html) and named the
colors in congruent trials (e.g., the word 'red' written in a red font)
and in incongruent trials (e.g., the word 'red' written in a green font).
The time they took to name all words was self-reported in seconds (e.g.,
21.3 seconds) for both the congruent and incongruent blocks. In this
analysis, we are interested in examining whether there is a Stroop effect.
15
16 {r}
17 # #run only once to download the data
18 # stroop_data <- read.table("https://raw.githubusercontent.com/Lakens/Stroop/master/stroop.txt", sep = "\t", header = TRUE)
19 #
```

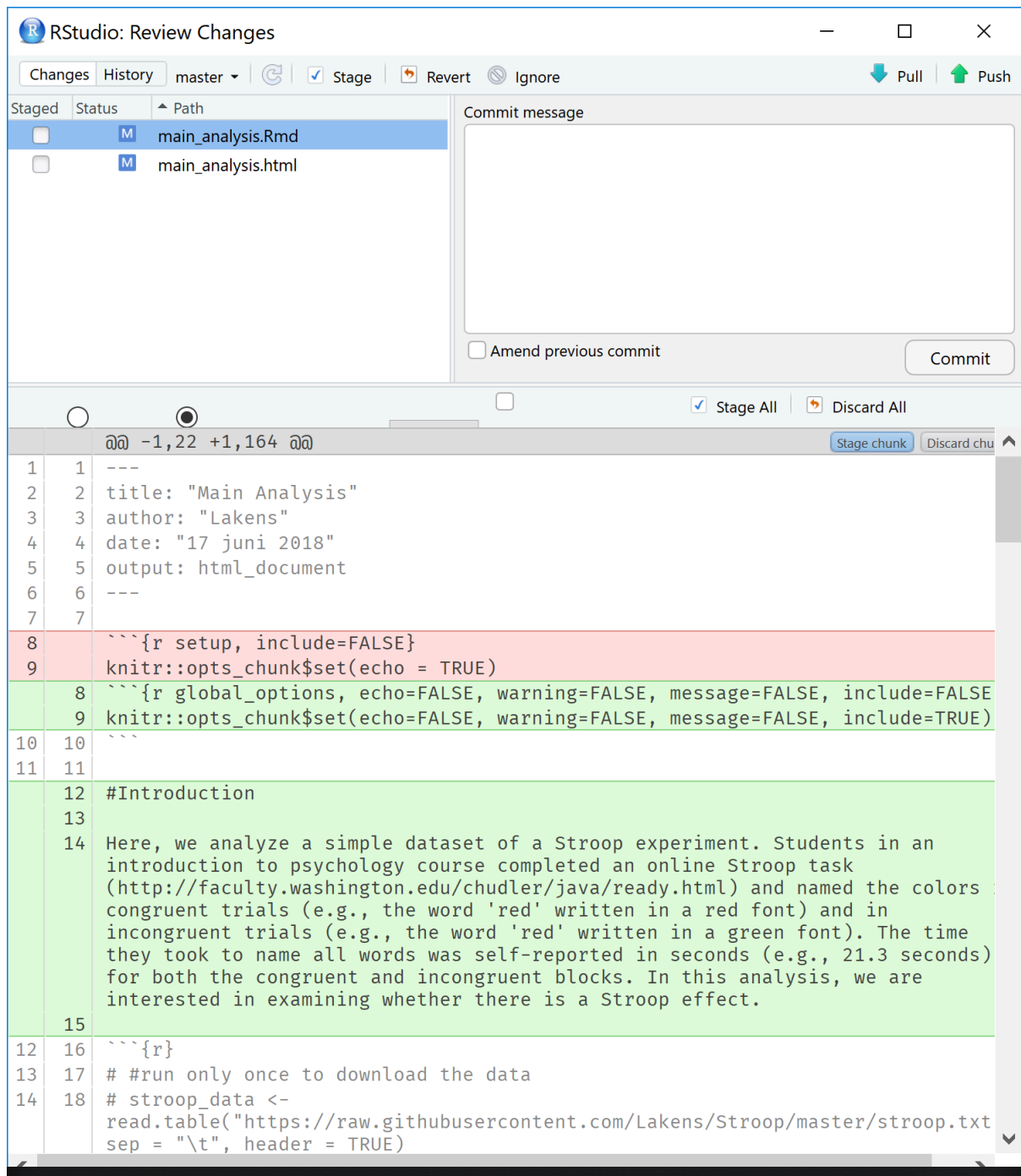
Results

The mean reaction time (in seconds) of participants in the Congruent condition ($M = 15.1$, $SD = 4.1$) was lower than the mean of participants in the Incongruent condition ($M = 23$, $SD = 4.78$, $r = 0.37$). An independent t -test indicated we could reject the null-hypothesis, based on an alpha of 0.05, $t(130) = 18.04$, $p < 0.001$. As we can expect from the Stroop effect, the standardized effect size is very large, Hedges' $g_{av} = 1.76$, 95% CI [1.48;2.06].



It is important to note that none of the numbers that are in this text are static, or copy-pasted. They are all calculated at the moment that the document is created, directly from the raw data. If you have access to the .Rmd (R Markdown) file, you can perfectly **reproduce** the reported data analysis.

Since we have made substantial changes, this is the perfect moment to **commit** and **push** the changes to GitHub! Go to the Git tab in the top right pane. Click 'Commit'. The window below will open. If the main_analysis.Rmd file is selected, you will see red and green chunks of text. These tell you what will be overwritten (red) and what is new (green).



Select all files that have changed, and 'stage' them (for example by pressing enter). The checkboxes in front of the files, under the 'Staged' column, should be checked.

| Staged | Status | Path |
|-------------------------------------|--------|--------------------|
| <input checked="" type="checkbox"/> | M | main_analysis.Rmd |
| <input checked="" type="checkbox"/> | M | main_analysis.html |

Type in a commit message, such as 'update mean analysis' in the 'commit message' field. Press the 'Commit' button. Close the window that pops up to inform you about the result of the commit. Then click 'push'. Close the window that informs you about the push command, and close the commit window. You can always visit the GitHub repository online and look at the full history of your document to see all changes that have been made.

Let's take a look at some sections of our new R Markdown document. First the header:

```
```{r global_options, echo=FALSE, warning=FALSE, message=FALSE,
include=FALSE}
knitr::opts_chunk$set(echo=FALSE, warning=FALSE, message=FALSE,
include=TRUE)
```
```

This sets general options for the code chunks in the R Markdown file. The echo, warning, and message = FALSE hide the code chunks, warning messages, and other messages, where the 'include=true' will make all figures appear in the text. You can set some of these variables to TRUE, and hit Knit to see what they change. Sometimes you might want to share the HTML file with all code visible, for example when sharing with collaborators.

If you scroll down, you will see the introduction text, the code that generates the first figure, and the code that performs the analyses. These variables are used in the Results section. Let's look at this code:

#Results

```
The mean reaction time (in seconds) of participants in the Congruent
condition (M = `r round(mean(stroop_data$Congruent), digits = 2)`, SD
= `r round(sd(stroop_data$Congruent), digits = 2)`) was lower than the
mean of participants in the Incongruent condition (M = `r
round(mean(stroop_data$Incongruent), digits = 2)`, SD = `r
round(sd(stroop_data$Incongruent), digits = 2)`, r = `r
round(cor(stroop_data$Congruent, stroop_data$Incongruent), digits = 2)`).
An independent t-test indicated we could reject the null-hypothesis,
based on an alpha of 0.05, t(`r round(ttest_result$parameter,
digits=2)`) = `r round(ttest_result$statistic, digits=2)`, p `r
ifelse(ttest_result$p.value > 0.001, " = ", " < ")` `r
ifelse(ttest_result$p.value > 0.001, formatC(round(ttest_result$p.value,
digits=3), digits=3, format="f"), "0.001")`. As we can expect from the
Stroop effect, the standardized effect size is very large, Hedges' g
= `r round(d_unb, digits=2)`, 95% CI [g `r round(ci_l_d_av, digits=2)`,
`r round(ci_u_d_av, digits=2)`].
```

This section of code shows how you can **mix text and R code**. The start of this code is normal text. The **M** is still normal text (the * * make the M italicized, just as further down the *~av~* indicates these letters should be a subscript), but then you see R code. In R Markdown you can embed R code within ``r``. Any R code within the two apostrophes will be executed. In this case, the mean of the Congruent reaction times is calculated, and rounded to 2 digits. You can see this number in the text.

Learning to program takes time. Some things are quite tricky to program. For example, the code:

```
`r ifelse(ttest_result$p.value > 0.001," = ", " < ")` `r  
ifelse(ttest_result$p.value > 0.001, formatC(round(ttest_result$p.value,  
digits=3), digits=3, format="f"), "0.001")`
```

is a lot of code to make sure the exact *p*-value is reported, unless this *p*-value is smaller than 0.001, in which case '*p* < 0.001' is printed. The first time you program something like this will take a lot of time. But remember, you will be able to re-use code in the future, and you can steal a lot of code from others! For a full introduction to Markdown, click [here](#).

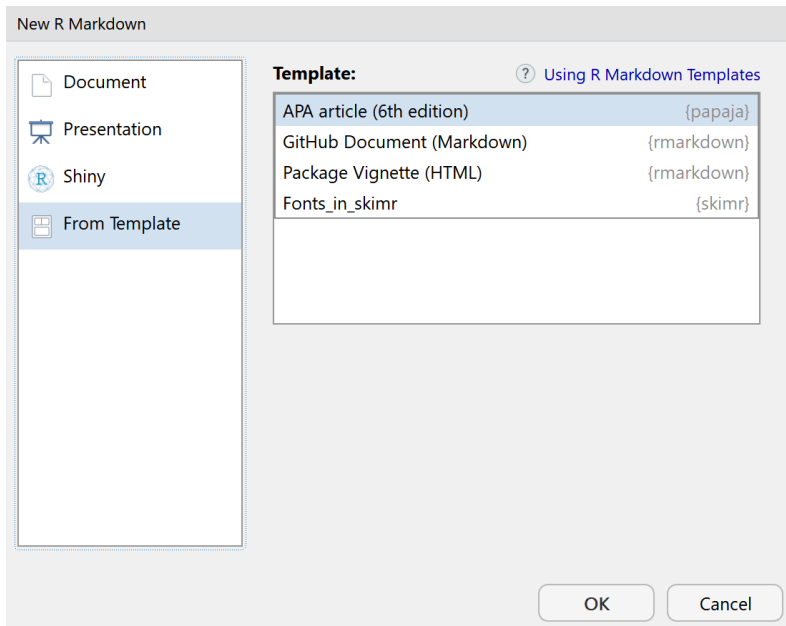
Extra: APA formatted manuscripts in papaja

If you want to write a reproducible manuscript in **APA style** (common in for example psychology) you might want to try out the R package [papaja](#) created by Frederik Aust. Install the papaja package by following the [instructions](#). Note that papaja is not yet on the Comprehensive R Archive Network ([CRAN](#)), but is installed directly from GitHub through:

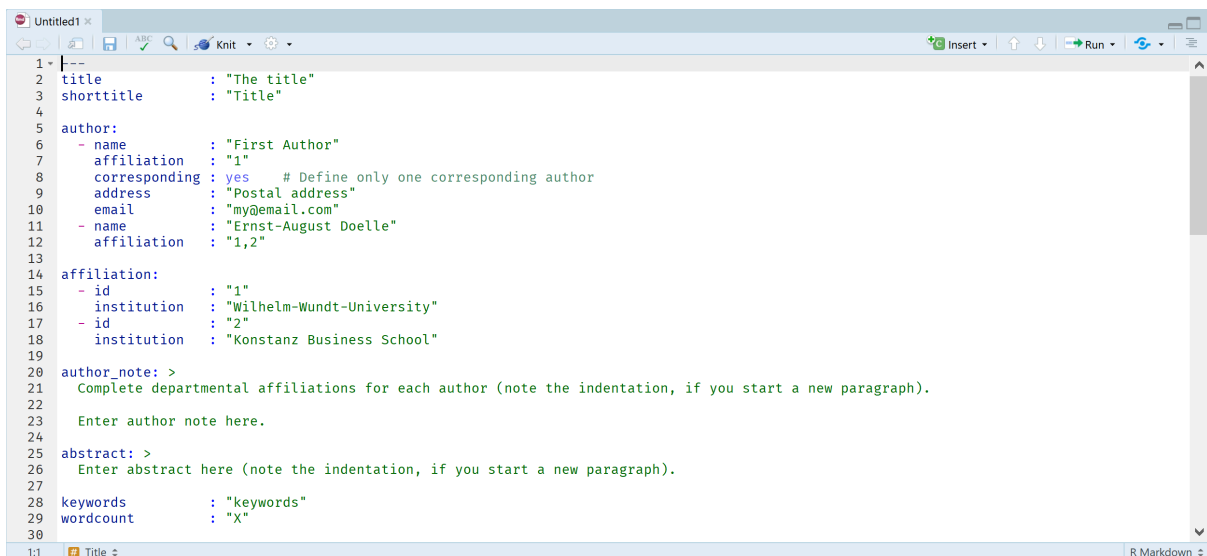
```
install.packages("devtools")  
devtools::install_github("crsh/papaja")
```

You need to install these packages only once (and update them whenever an update is available). To share packages on CRAN, creators need to meet a set of criteria (e.g., provide good documentation), but researchers sometimes prefer to share early versions of their packages through GitHub.

Then, create a new R Markdown document, but instead of selecting the document option, select the 'From Template' option, and select the template APA article (6th edition) provided by the papaja package.



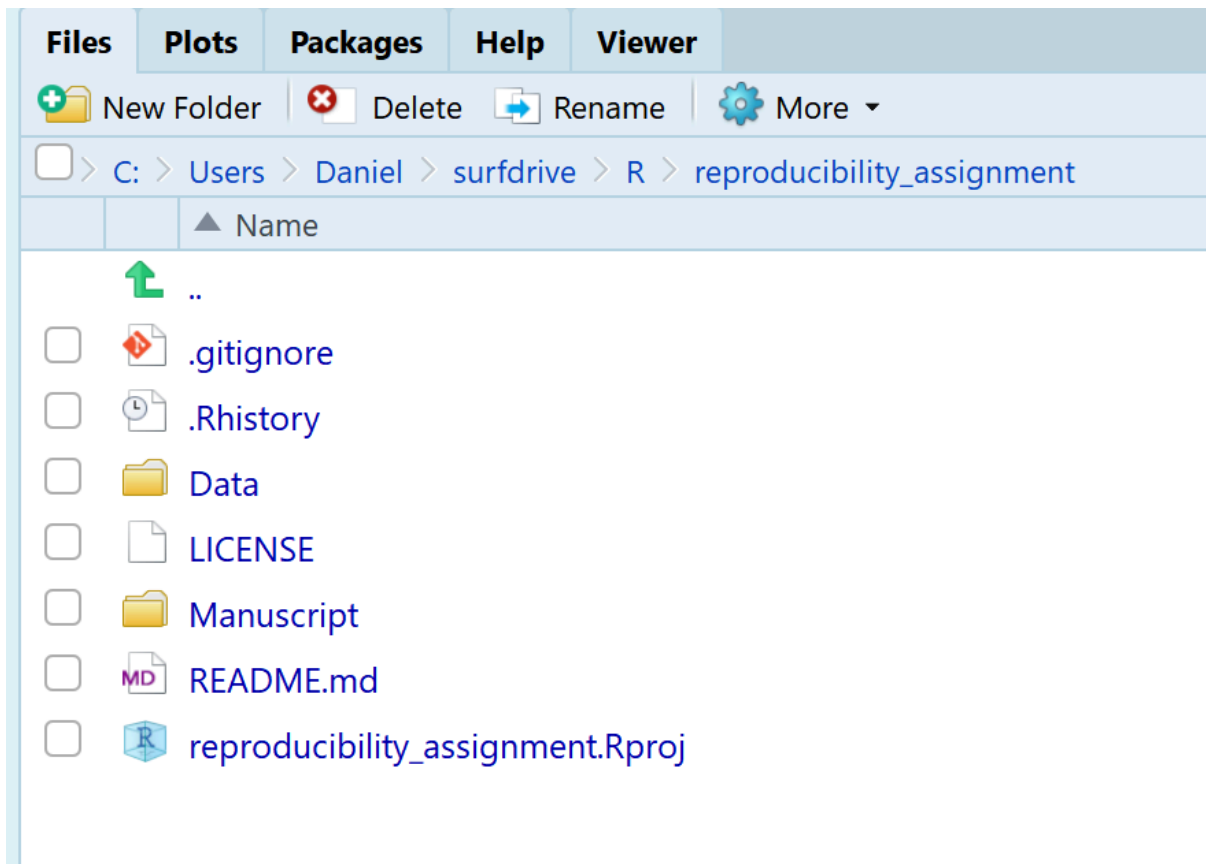
You will see a template with a lot of fields to fill in, such as the title, author names and affiliation, the author note, the abstract, etc. Papaja takes care that all this information ends up in a nice lay-out – exactly following the APA rules. This means that if you have installed MiKTeX, you can Knit the document to a pdf, and submit an APA formatted document that is completely reproducible. For a tutorial covering all options in papaja, including how to add citations, see https://crsh.github.io/papaja_man/index.html



Step 7: Organizing Your Data and Code

It is important to always **organize your data and analysis files**. This helps others to quickly find the files they are looking for. In general, I recommend the **TIER protocol**: <https://www.projecttier.org/tier-protocol/>. You can create folders and move files through the RStudio interface (click the 'New Folder' button and the 'More' button for a 'Move' option). But you can also just go to the folder and create new folders and move

files as you do normally. I've created a data folder that contains the stroop.csv file, and a manuscript folder that contains the .Rmd and .html files. Push the changes to GitHub!



If you try to knit your code now, you will get an error:

```
✖ Line 17 Error in file(file, "rt") : cannot open the connection Calls: <Anonymous> ...  
withVisible -> eval -> eval -> read.csv -> read.table -> file In addition: Warning  
message: In file(file, "rt") : cannot open file 'stroop.csv': No such file or  
directory
```

The data is no longer in the same folder as our manuscript file. If you want to knit your file, you need to tell R to move up one folder level, and then go to the Data folder:

```
stroop_data <- read.csv("../Data/stroop.csv", sep = " ", header = TRUE)
```

When you are organizing your code, **take great care to make sure that any personally identifying information in your data is stored safely**. Open science is great, but you should share data responsibly. This means that you need to **ask participants permission to share their data** in the informed consent form (a [useful resource](#) is the Research Data Management Support page of the University of Utrecht). Whenever you collect personal data, make sure you [handle this data responsibly](#). Information specialists at your university library should be able to help.

Step 8: Archiving Your Data and Code

Although we have shared our data and code on GitHub, when you publish your article and want to **share your data and code**, it is important to remember that GitHub is not a data repository that guarantees long-term data storage. This makes it less suitable to link to in scientific articles, which will be around decades from now. For scientific publications, you will want to link to a stable long-term data repository. For a **list of data repositories**, click [HERE](#). We will use the Open Science Framework in this assignment.

Log in to the OSF at <https://osf.io/> (create an account if you haven't already done so). Click 'Create new project'. Give your project a name (for example 'Stroop Reproducible Analysis Assignment').

It is again important to add a **license** to your work, also on your OSF project. Click 'Add a license':

License: Add a license

Choose a license: MIT License ▼

Year: 2018

Copyright Holders: Daniel Lakens

The MIT License (MIT)
Copyright (c) 2018 Daniel Lakens
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights








You can again choose a MIT license. You will have to fill in a year, and the copyright holders – in this case, that is you, so fill in your name (it will appear in the license text).

Then click .

Although we could upload all our files to the OSF, we can also simply link our GitHub project to the OSF. In the menu bar, click on 'Add-ons'. In the list, scroll to GitHub:

Select Add-ons


Sync your projects with external services to help stay connected and organized. Select a category and browse the options.

| Categories | Search... |
|------------|--|
| All |  Dropbox Enable |
| Citations |  figshare Enable |
| Storage |  GitHub Enable |
| |  GitLab Enable |
| |  Google Drive Enable |
| |  Mendeley Enable |
| |  OneDrive Enable |

Follow the step-by-step guide to connect to GitHub provided by the OSF (<http://help.osf.io/a/837075>)

Select your repository that contains the reproducibility assignment and click 'Save'.

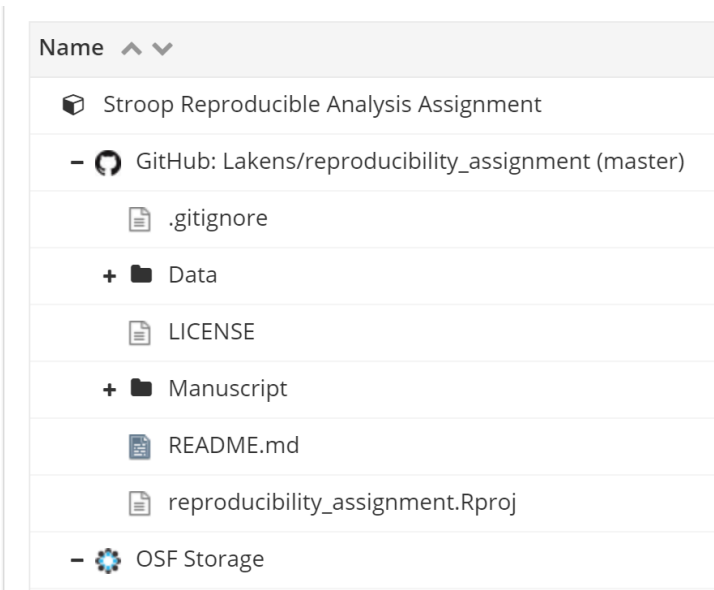
Configure Add-ons

 GitHub authorized by Daniel Lakens Disconnect Account

Current Repo:

Save Create Repo

Click the title of the OSF project page to go back to the main project page. You will now see in the 'Files' pane that the GitHub repository is linked (see the picture below). One nice thing of this setup is that any changes you make on GitHub is also visible in your OSF project. Your GitHub repository is linked, which means that every change on GitHub is directly visible.



This is a good moment to click the 'Make Public' button in the top right of your project. After making the project public, people will be able to find it on the OSF. If you don't want to make your project public just yet, but you do want to give others access to your files, you can create a **'View-only' link** on the OSF. Go to 'Contributors' and click the +Add button next to View-only links. For a step-by-step guide, see [this tutorial](#).

View-only Links + Add








Create a link to share this project so those who have the link can view—but not edit—the project.

You can use a view-only link to share access to your files only with reviewers. You can create an anonymized view-only link to hide your contributor names in the project - this is particularly useful in **double-blind peer review** (where both the reviewer and author identities are concealed from the reviewers, and vice versa, throughout the review process). Giving access to the files during peer review helps reviewers to evaluate your work in as much detail as they want to. But beware: this means that people you don't know will have access to your files. So far, I don't know of any negative experiences with this process, but it is important to be aware that others will have access to your files before they are published.

The OSF page now just links to the files on the GitHub page. It does not independently store them. This means we do not yet have a **long-term stable data storage solution**.

To create a snapshot of all files in the GitHub repository that will be stored for a long time, you have to create a **Registration** of your project. We will not create a Registration of your project in this assignment. Creating a registration starts with several formal procedures: data in linked repositories (such as GitHub) are stored by the OSF, and the project appears in the list of registrations. You should only register when you want to create a stable copy of your work. Below you can see an example of the files in an OSF project that has been registered. The GitHub repository that was linked to the project has been turned into an Archive of GitHub - this creates a stable version of the project,

as it was at the moment you registered. This archive will not change. There is still a live version of your project on the OSF.

| Name ^ v | Modified ^ v |
|---|--------------|
|  Equivalence Testing for Psychological Research:... | |
| -  OSF Storage | |
| +  Archive of GitHub: Lakens-EquivalenceTe... | |
| -  Equivalence Testing for Psychological Resea... | |
| -  OSF Storage | |
|  PREPRINT_Lakens_etal_EquivalenceTe... 2018-02-19 05:19 PM | |
| +  Equivalence Testing for Psychological Res... | |

A good moment to create a stable version of your project is when your manuscript is accepted for publication. You can create a Registration and use the Digital Object Identifier (DOI) to link to the code, data, and materials in the paper (you can add this link to the manuscript as you check the proofs of your article before it is published). Note that it is recommended to link to the materials using the **DOI**. The DOI is a persistent link (meaning it will keep working) where a website address might change. A registration does not automatically get a DOI. After creating the Registration, you need to click the 'Create DOI' link to create a persistent object identifier.

[Contributors:](#) [Daniel Lakens](#)

Date registered: 2016-11-10 04:52 PM

Date created: 2016-07-14 09:04 AM

[Create DOI](#)

Category:  Project

If you are ready to create a Registration, follow the instructions on the OSF: <http://help.osf.io/m/registrations//524205-register-your-project>. As an example of a Registration that was made to store all work related to one of my scientific publications, see <https://doi.org/10.17605/OSF.IO/9Z6WB> (this link is itself an example of how to link to an OSF project using a DOI).

EXTRA: Sharing Reproducible Code on Code Ocean

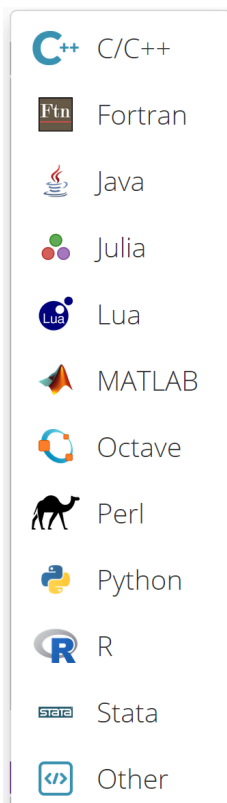
If you have used the workflow above to create a reproducible manuscript, you might want to make it easy for other people to explore your data and code. People can simply clone your GitHub repository – but this still requires them to install the software you used, and even if they have R installed, it requires them to install the relevant packages to analyze your data. This can potentially lead to reproducibility problems. Packages in R update and change over time, and code that works on one machine might not run well on another computer. Furthermore, even when shared perfectly, downloading all files

and getting the code up and running takes time. Several solutions exist, such as [Packrat](#), which is a dependency management system for R, or [Docker](#), which can create a container that works as a virtual machine that includes a computing environment including all the libraries, code and data that you need to reproduce an analysis.

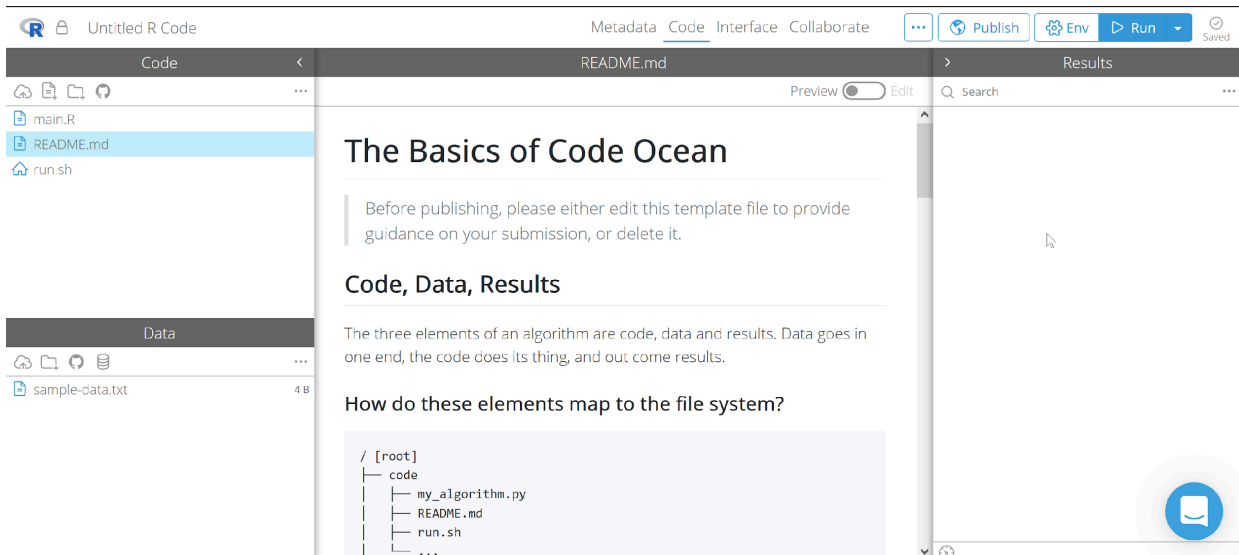
Here, I'll focus on a software solution that is designed to be easier to use than Docker, but provides many of the same benefits: [Code Ocean](#). Code Ocean is a cloud-based computational reproducibility platform. You can create a computing capsule that runs online and contains all packages your code needs to run. Although Code Ocean does not (yet) guarantee long term storage of data and code, it is an useful way to make your reproducible code available to fellow researchers, and makes it easy for researchers (or reviewers) to make small changes to your code and examine the results. Create a (free) account on CodeOcean. Then create a new computing capsule:



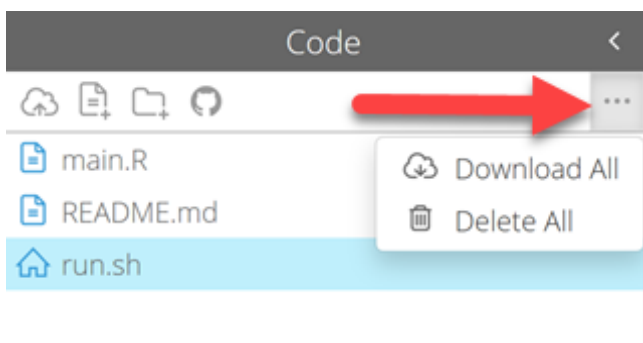
From the list of software package, choose R:



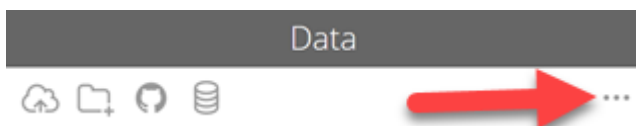
You will see the computing capsule. On the top left you see a CODE section, on the bottom left you see a DATA section, in the middle you can see the contents of the highlighted code file (in this case, the README.md file), and on the right you see a RESULTS pane (that is currently empty).



We don't need any of the default files, so in the CODE window click the ... menu button and select delete all:



Do the same for the files in the DATA window, so that we have a completely empty container. You can import your files by just uploading them, but if you already use GitHub, the easiest way to get your files into Code Ocean is to import them from GitHub. Click the GitHub button in the Data pane:



A menu opens. If you made your GitHub repository public, you can just copy-paste the address of your repository:

Import from GitHub ?













Enter any public GitHub repository:

`https://github.com/Lakens/reproducibility_assignment`


Cancel

Import Input Files

Click 'Import Input Files' and you will see all files from your repository:

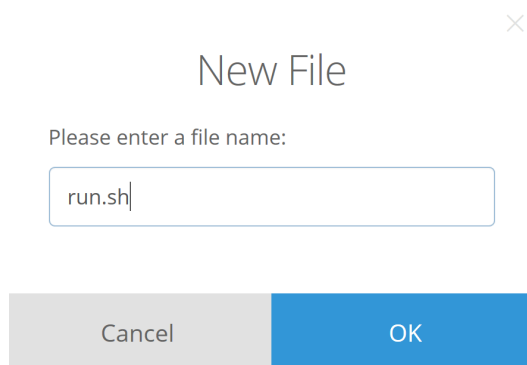
| Data | |
|--|-----------|
|     ... | |
|  Data | 2.78 KB |
|  Manuscript | 805.25 KB |
|  .gitignore | 40 B |
|  LICENSE | 1.04 KB |
|  README.md | 111 B |
|  reproducibility_assignment.Rproj | 205 B |

Delete all files except the Data folder. Repeat the import process for the Code window, but there delete all files except the Manuscript folder. The two panes now both contain a single folder: one containing the main_analysis.Rmd file, and one containing 'stroop.csv'.

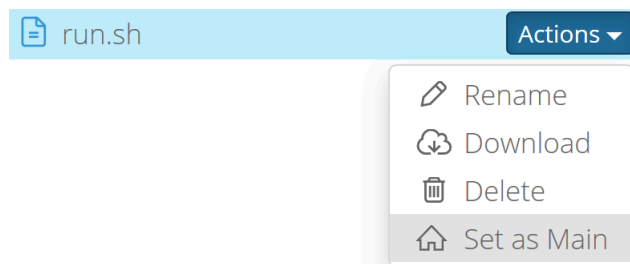
Now we need to make some changes to make sure Code Ocean can Knit our R Markdown file. First, click the  button to create a blank file:




Call the new file run.sh:



Set this file as the Main Code file by clicking the 'Actions' dropdown menu (it will appear when you hover over the file name), and selecting the 'Set as Main' option:




The symbol in front of the run.sh file will change into a house. When (after performing the steps below) you click the big run button ( Run), Code Ocean will execute the file you have specified as the main file. Let's add some code in our new file. Select the run.sh file, and it will open in the middle pane:



Copy the code below (again, be careful when copying code from a pdf file – check the public Code Ocean example linked on page 2 if you get stuck) and paste it in the new run.sh file:

```
#!/bin/bash
```

```
Rscript -e "rmarkdown::render(input = 'Manuscript/main_analysis.Rmd', \  
  output_dir = '../results', clean = TRUE)"
```

Now, we need to set up the environment. Click 

It is important to tell Code Ocean which packages and files it needs. This can be a bit tricky to figure out (in part, this is a lack of clarity in the way R informs users which packages it needs to run). Many R packages are already pre-installed in the environment. For example, we don't need to tell Code Ocean that we want to make

ggplot2 available – it is installed by default. However, we do need to specify that we want to use the MBESS package, and we need to specify we need pandoc to knit our .Rmd file to HTML. We can specify packages to run from CRAN (which contains many R packages), install packages directly from GitHub (for example the papaja package, if you had used this to create an APA formatted manuscript), and the apt-get installers are used to make other software (such as pandoc) available to the Linux system the Code Ocean container runs on.

✕

Run Environment

Packages
Setup Script

Base Environment: R 3.4 v 3.4.4 v

Pick an installer from the list to view installed packages and add new ones to it:

Installers

apt-get ⚙

Bioconductor

R (CRAN) ⚙

R (GitHub)

| Package | Version |
|---------|---------|
| | |

Package
Version
Add

e.g. gcc e.g. 1.4

Show all packages

Cancel
Done

When the R (CRAN) option is selected, in the package field type MBESS, and click 'Add'.

Package

MBESS

Version

e.g. 1.4

Add

The MBESS package now appears in the list:

| Package | Version |
|---------|---|
| ● MBESS | latest 🗑 |

We also need to specify some packages when selecting the apt-get option. Click on the apt-get item in the list of installers. In the package window type 'pandoc' and click 'Add', click gsl-bin and click 'Add', and type libgsl-dev and click 'Add'. This should look like:

| Package | Version |
|--------------|-----------------|
| ● gsl-bin | 2.1+dfsg-2 |
| ● libgsl-dev | 2.1+dfsg-2 |
| ● pandoc | 1.16.0.2~dfsg-1 |

Then click 'Done'.

We also need to make one change to the R Markdown file. You need to specify where the data should be read in from (which is no longer the local hard drive). Our 'Stroop.csv' file is in the Data pane. You need to change the original line of code:

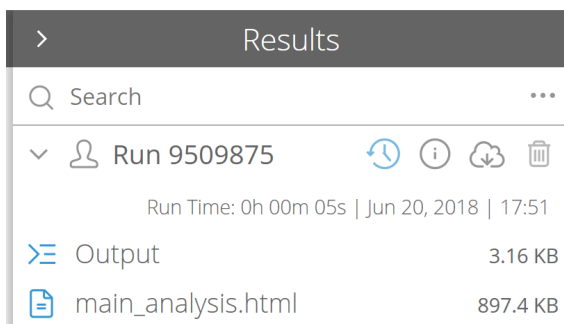
```
stroop_data <- read.csv("stroop.csv", sep = " ", header = TRUE)
```

to the line below:

```
stroop_data <- read.csv("/data/Data/stroop.csv", sep = " ", header = TRUE)
```

This tells Code Ocean to look in the data pane (/data/) and then in the data folder (Data/).

We can now click the 'Run' button. This is a good moment to go and have a coffee. Code Ocean will create the container and install all packages you need – not just packages you have specified, such as MBESS, but also all packages MBESS depends on – and this takes at least 10 minutes. Wait patiently. This only takes long once – after that, running code is fast. When the code is completely done, you should see files appear in the 'Results' pane:



Click on the main_analysis.html file, and you will see a pop-up with the HTML file:

Main Analysis

Lakens

17 juni 2018

Introduction

Here, we analyze a simple dataset of a Stroop experiment. Students in an introduction to psychology course completed an online Stroop task (<http://faculty.washington.edu/chudler/java/ready.html>) and named the colors in congruent trials (e.g., the word 'red' written in a red font) and in incongruent trials (e.g., the word 'red' written in a green font). The time they took to name all words was self-reported in seconds (e.g., 21.3 seconds) for both the congruent and incongruent blocks. In this analysis, we are interested in examining whether there is a Stroop effect.

There is now a completely reproducible analysis file online. Anyone can now reproduce your data analysis – they can go into the main_analysis.Rmd file, and change anything they want in your code, and run it again. For example, let's say you dislike the black straight line in the first scatterplot, and you want it to be red. It is easy to change 'black' to 'red' in line 40, and re-run the analysis, and you will get a figure with a red line. Although that might in itself not be very exciting, the ability to easily re-analyze data might be useful in more realistic scenarios. For example, imagine you are reviewing a paper where the researchers do not plot the data. Without having to install any software, you can just type in `hist(stroop_data$Congruent)` after the data has been read in (e.g., on line 24), run the code again, and you will see a histogram for the reaction times in the Congruent condition. Give it a try.

Conclusion

In this assignment, a number of platforms and software solutions were introduced, such as GitHub, the Open Science Framework, RStudio, R, and R Markdown. Following the example here is not the same as being able to use these tools in your research. Learning to use these tools will take time. You will often get frustrated when the code or software doesn't work as you want, or when you have gotten your local and remote GitHub repositories so much out of sync you just need to delete everything on your local computer and re-download all files from GitHub (`git reset --hard [HEAD]` is your friend). But there are many resources available online to find answers, or to ask for help. In my experience, it takes some work, but is doable, even if you have very limited knowledge of programming. You can get a basic reproducible workflow up and running by using all the steps described here, and then learn new skills as you need them. These skills are valued both within and outside of academia and will save you time (e.g., when re-creating figures for a revision, or when analyzing similar datasets in the future). A reproducible workflow also improves the quality of your scientific work, and makes it easier for other scientists to re-use your work in the future.



© Daniel Lakens, 2018. This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 License](https://creativecommons.org/licenses/by-nc-sa/4.0/).