

Lab 01 - Hello R!

Moriah Ruggerio

`R` is the name of the programming language itself and `RStudio` is a convenient interface.

The main goal of this lab is to introduce you to `R` and `RStudio`, which we will be using throughout the course both to learn the statistical concepts discussed in the course and to analyze real data and come to informed conclusions.

`git` is a version control system (like "Track Changes" features from Microsoft Word on steroids) and `Git`

An additional goal is to introduce you to `Git` and `GitHub`, which is the collaboration and version control system that we will be using throughout the course.

As the labs progress, you are encouraged to explore beyond what the labs dictate; a willingness to experiment will make you a much better programmer. Before we get to that stage, however, you need to build some basic fluency in `R`. Today we begin with the fundamental building blocks of `R` and `RStudio`: the interface, reading in data, and basic commands.

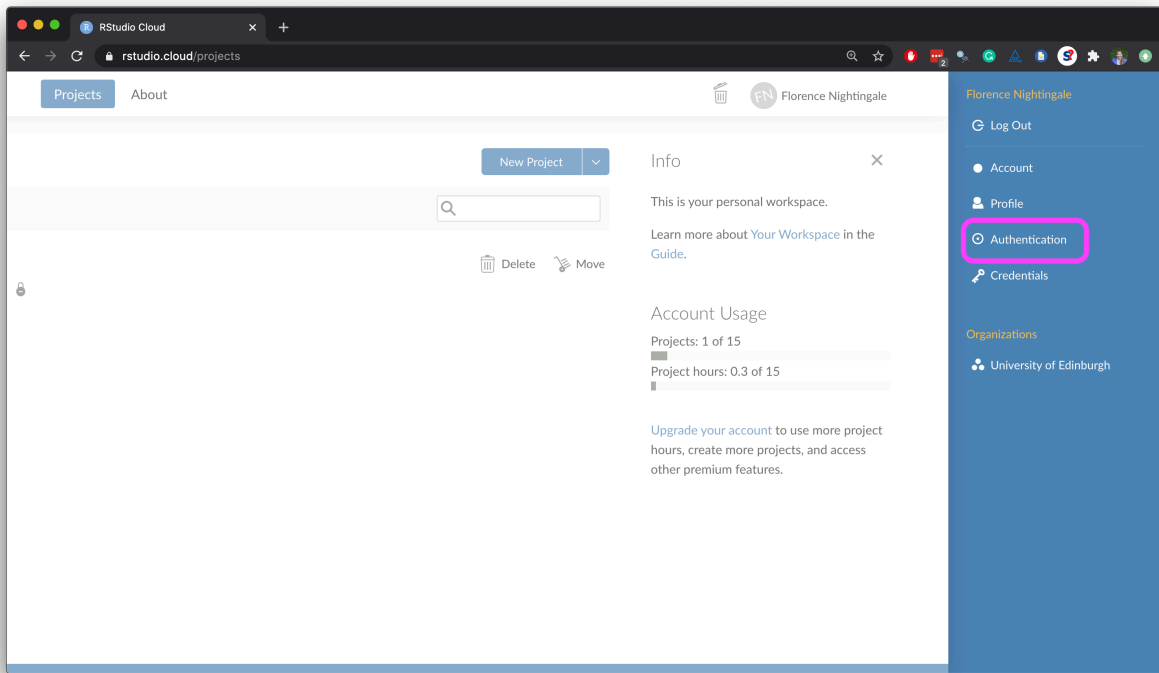
And to make versioning simpler, this is a solo lab. Additionally, we want to make sure everyone gets a significant amount of time at the steering wheel. In future labs you'll learn about collaborating on `GitHub` and produce a single lab report for your team.

Connecting `GitHub` and `RStudio Cloud`

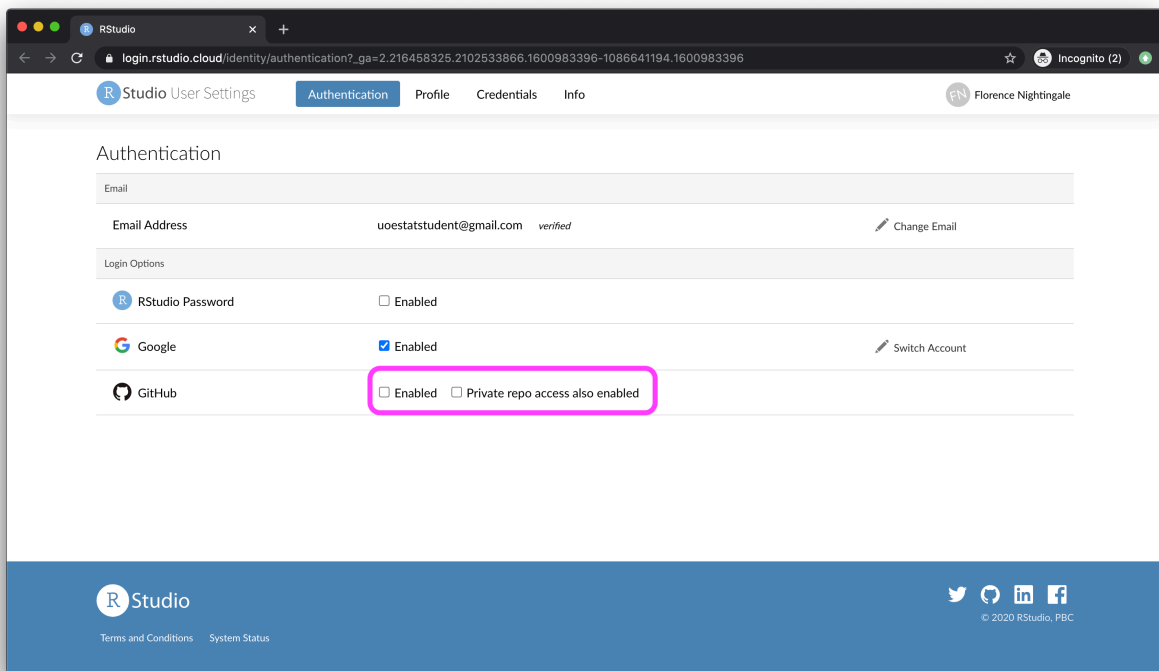
You should have already received an invitation to join the `GitHub` organization for this course. You need to accept the invitation before moving on to the next step.

To connect your `RStudio` and `GitHub` accounts by following the steps below:

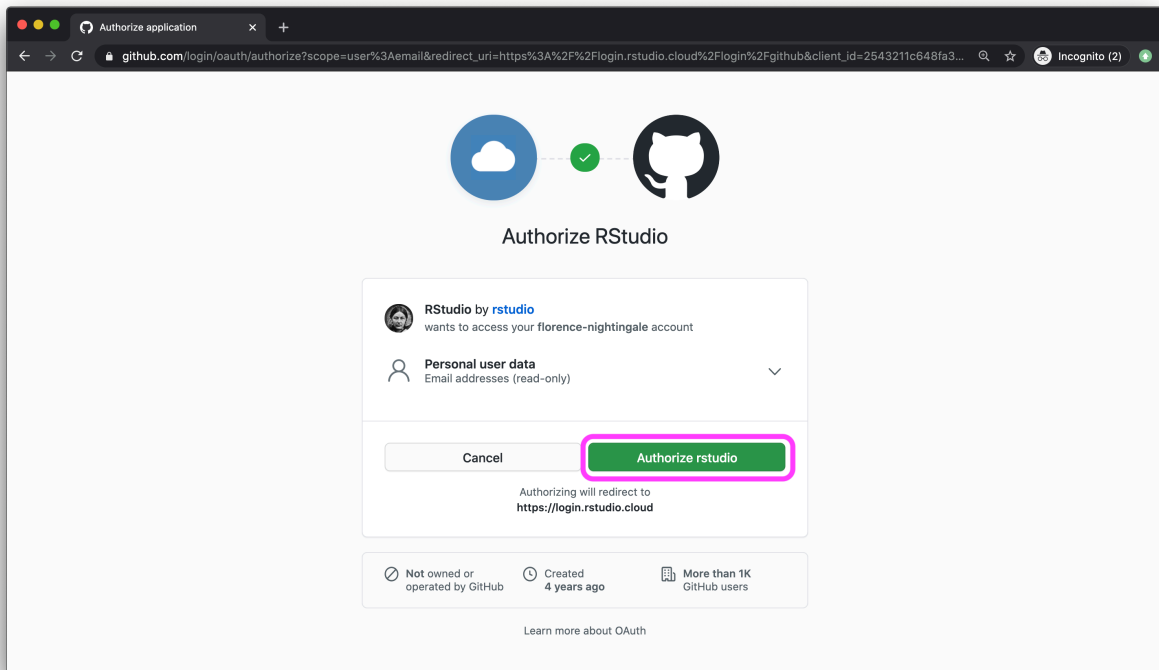
- Click on your name on the top right corner to open the right menu.
- Then, click on Authentication.



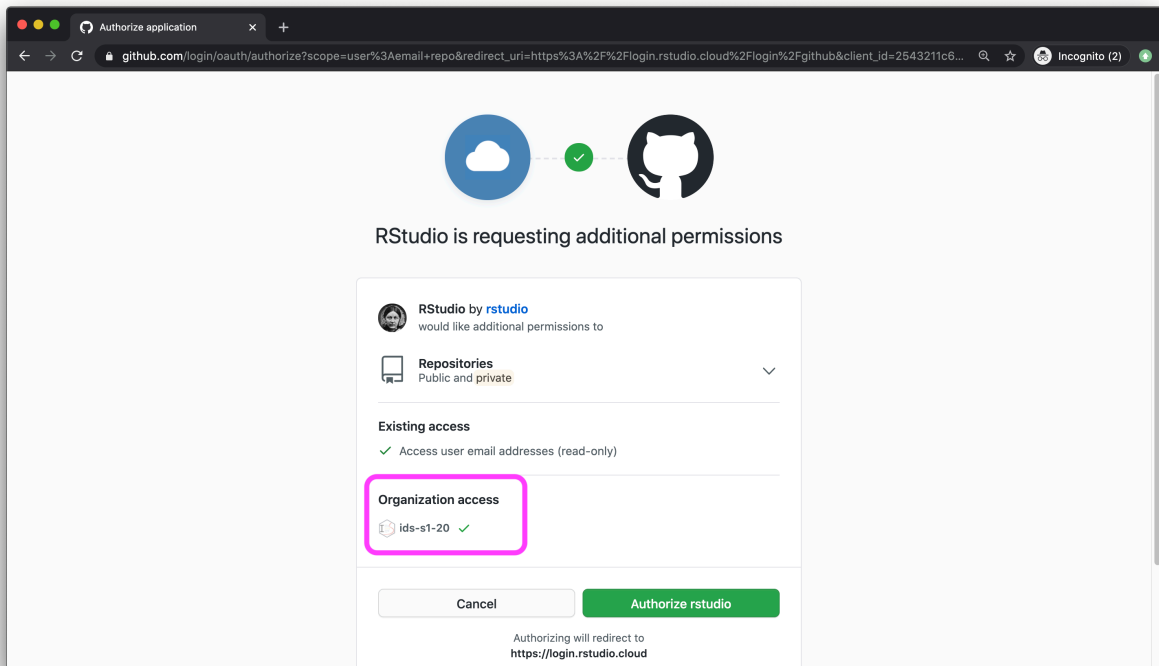
- In the Authentication window, check the box for *Enabled*.



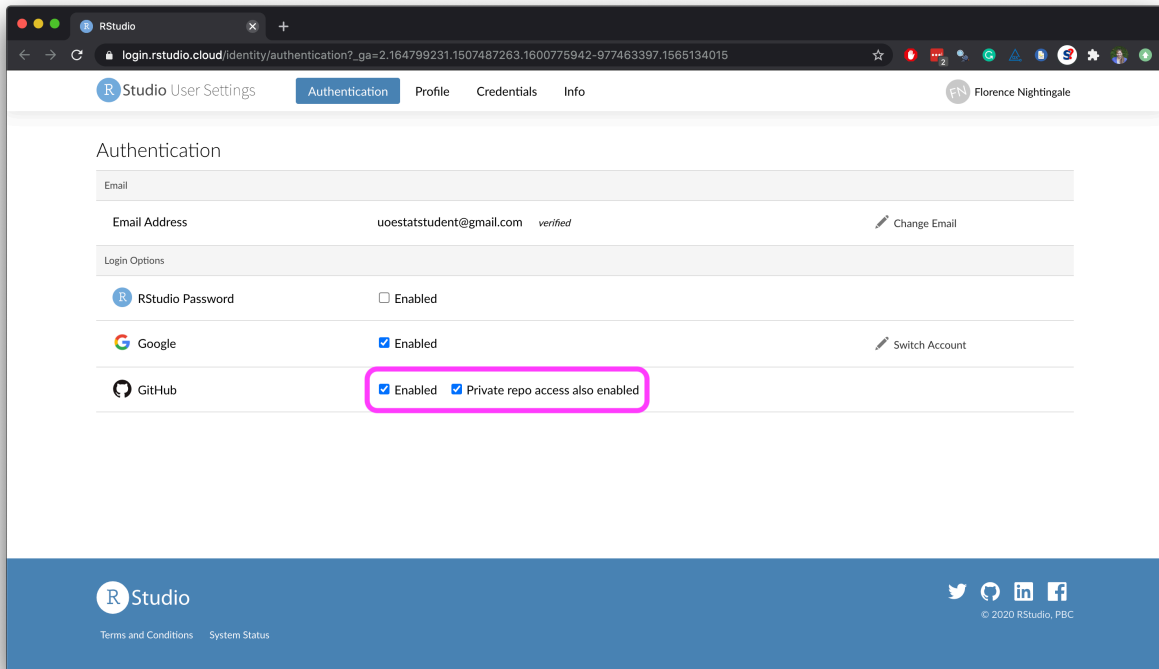
- In the next window, click on the green box that says “Authorize rstudio”.



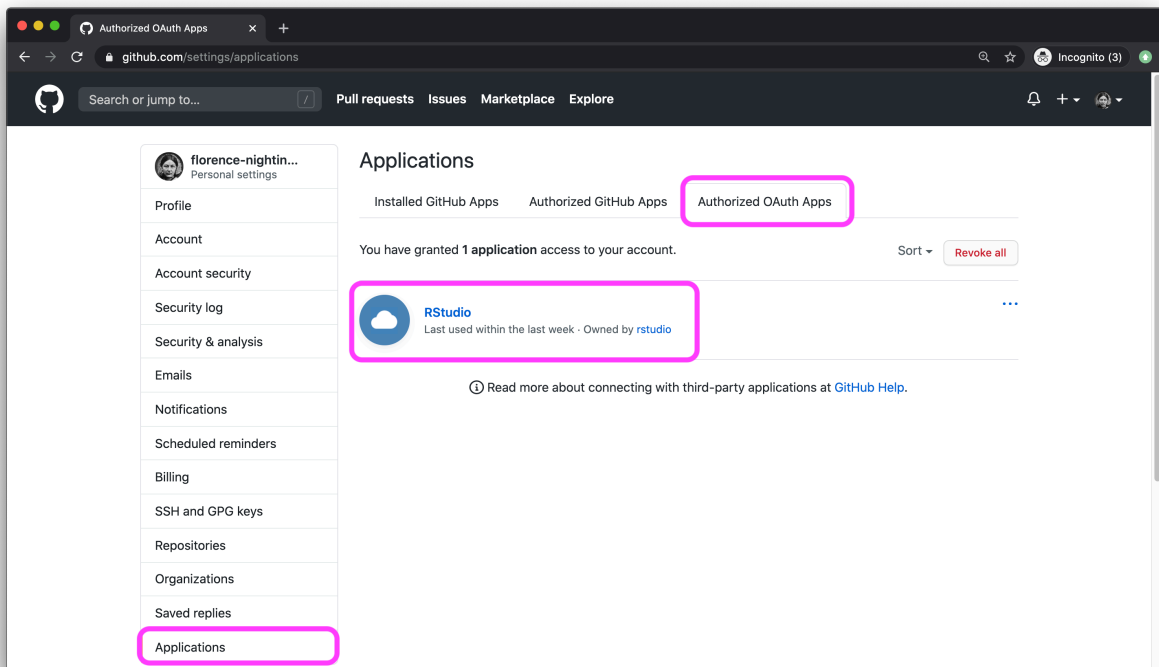
- Back in the Authentication window, check the box for *Private repo access also enabled*, and once again, on the green box that says “Authorize rstudio” in the next window. At this point you should also make sure that the course organization shows up for you under *Organization access*. If it does not, this means you have not yet accepted the GitHub invitation to join the course, and you should go back and do that.



- Once you're done, both of these boxes should be checked.



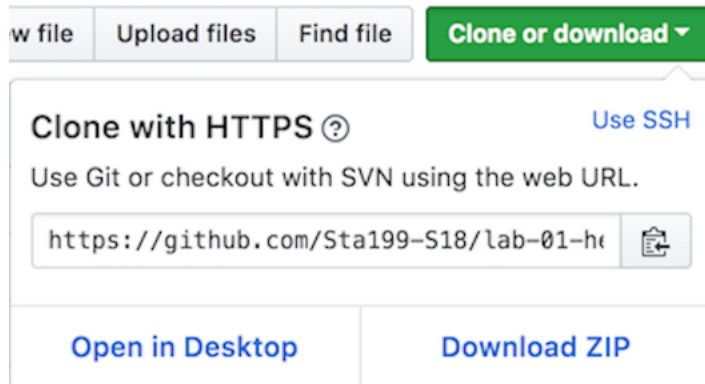
- To confirm that you've successfully linked up your GitHub and RStudio Cloud accounts, GitHub settings > Applications. You should see RStudio listed as an authorized app under *Authorized OAuth Apps*. If you don't this is a good time to ask a question.



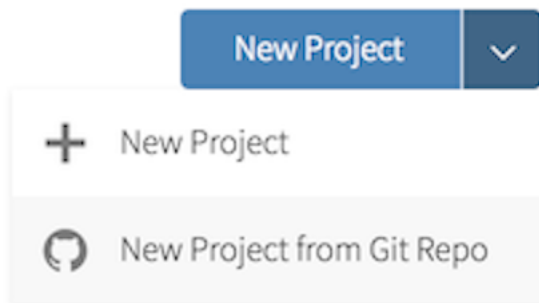
Getting started

Each of your assignments will begin with the following steps. You saw these once in class yesterday, they're outlined in detail here again. Going forward each lab will start with a "Getting started" section but details will be a bit more sparse than this. You can always refer back to this lab for a detailed list of the steps involved for getting started with an assignment.

- Click on the assignment link that you should have received in your email to create your GitHub repository (which we'll refer to as "repo" going forward) for the assignment. This repo contains a template you can build on to complete your assignment.



- On GitHub, click on the green **Clone or download** button, select **Use HTTPS** (this might already be selected by default, and if it is, you'll see the text **Clone with HTTPS** as in the image below). Click on the clipboard icon to copy the repo URL.



- Go to RStudio Cloud and into the course workspace. Create a **New Project from Git Repo**. You will need to click on the down arrow next to the **New Project** button to see this option.

New Project from Git Repo

URL of your Git repository

OK

- Copy and paste the URL of your assignment repo into the dialog box:
- Hit OK, and you're good to go!

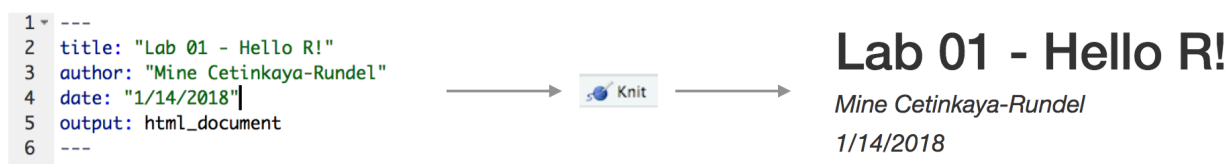
Warm up

Before we introduce the data, let's warm up with some simple exercises.

The top portion of your R Markdown file (between the three dashed lines) is called **YAML**. It stands for

YAML

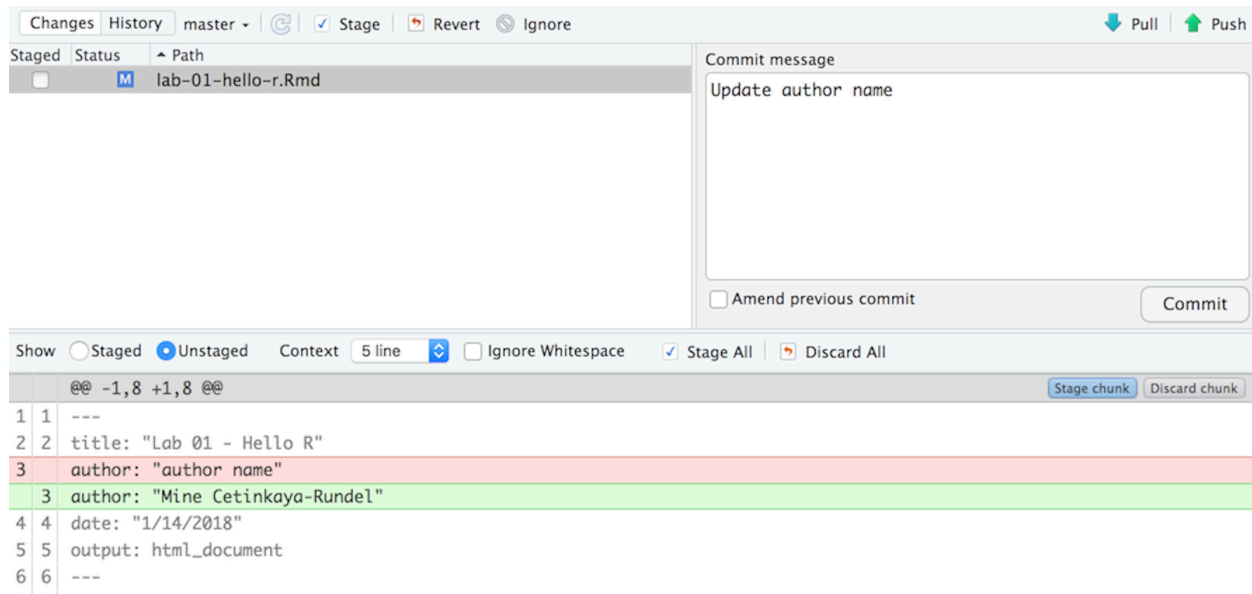
Open the R Markdown (Rmd) file in your project, change the author name to your name, and knit the document.



Committing changes

Then go to the Git pane in your RStudio.

If you have made changes to your Rmd file, you should see it listed here. Click on it to select it in this list and then click on **Diff**. This shows you the *difference* between the last committed state of the document and its current state that includes your changes. If you're happy with these changes, write "Update author name" in the **Commit message** box and hit **Commit**.



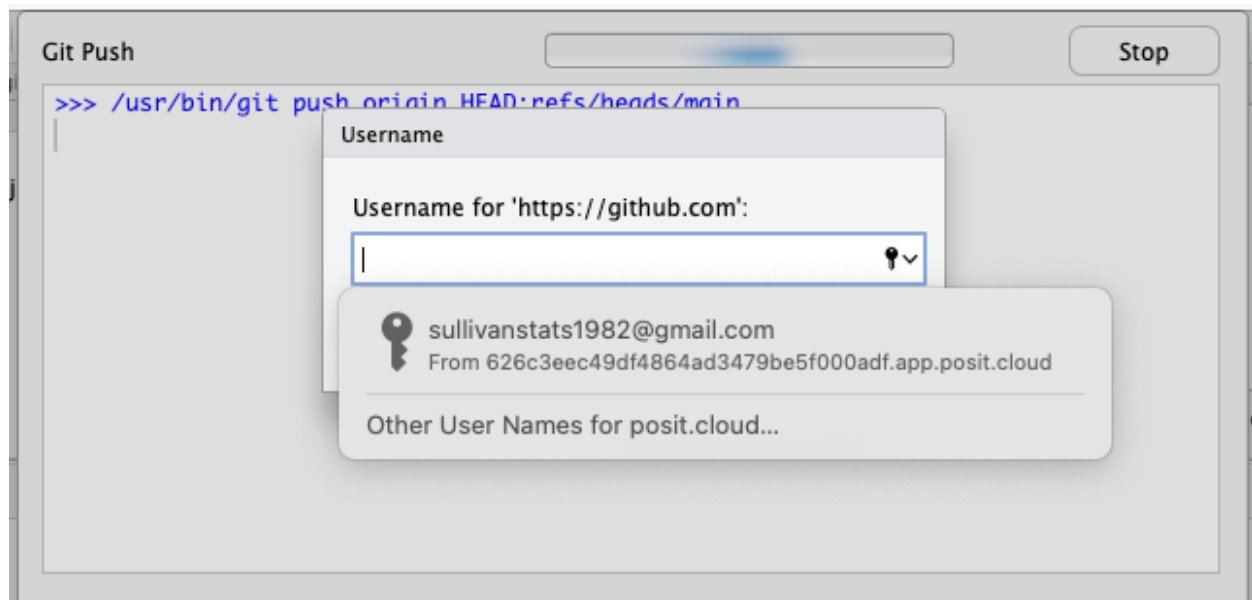
You don't have to commit after every change, this would get quite cumbersome. You should consider committing states that are *meaningful to you* for inspection, comparison, or restoration. In the first few assignments we will tell you exactly when to commit and in some cases, what commit message to use. As the semester progresses we will let you make these decisions.

Pushing changes

Now that you have made an update and committed this change, it's time to push these changes to the web! Or more specifically, to your repo on GitHub. Why? So that others can see your changes. And by others, we mean the course teaching team (your repos in this course are private to you and us, only).

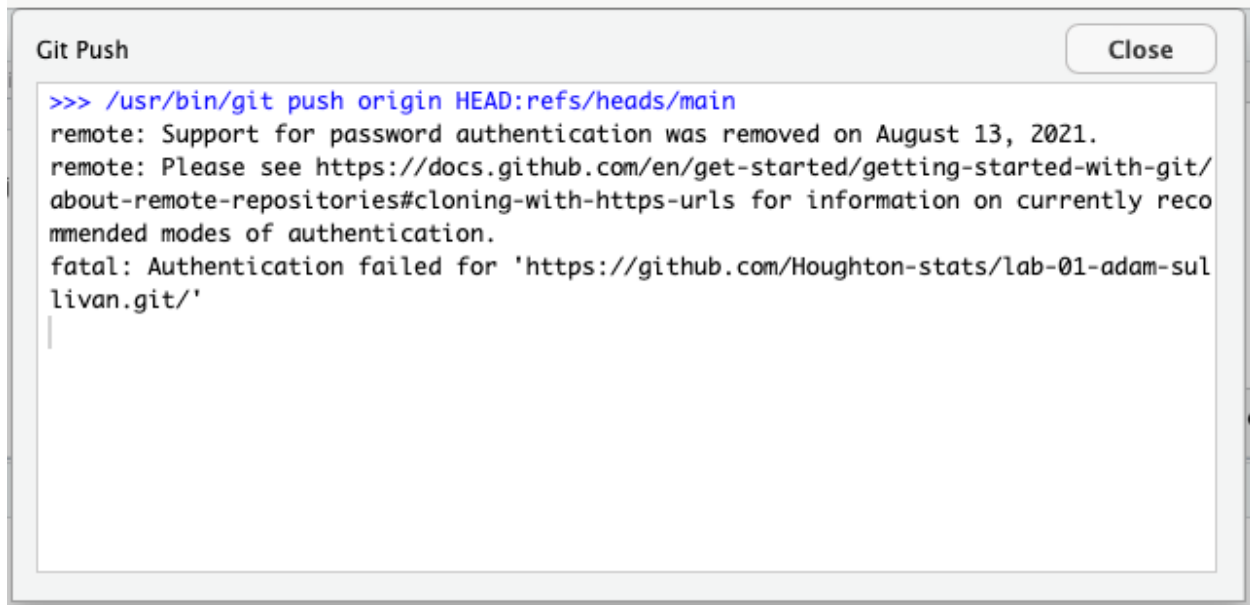
In order to push your changes to GitHub, click on **Push**. This will prompt a dialogue box where you first need to enter your user name, and then your Personal Access Token.

The box to enter your username looks like this:



When you enter this and hit enter it asks you to enter your password. However what this really wants is a

personal access token and if you just enter your password you see the message below:



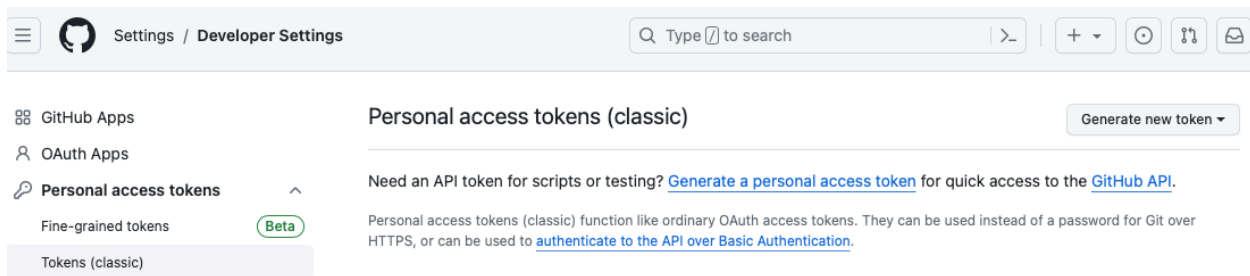
```
Git Push
Close

>>> /usr/bin/git push origin HEAD:refs/heads/main
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/Houghton-stats/lab-01-adam-sullivan.git/'
```

Here are the instructions to get a Personal Access Token:

Go to your github user account settings.

Scroll down to where it says developer settings on the left hand side and click this. Then you will see the following menu below where you will click ***Tokens (classic)***.



Once you have done this click ***Generate a personal access token***. It will take you to the page below:

New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

data-science-class

What's this token for?

Expiration *

Custom...

05/31/2024

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events

Then change the following:

- Note
 - data-science-class
- Expiration
 - custom
 - ends may 31, 2024
- Select Scopes
 - repo*

Scroll to the bottom of this page and click to finish. You then have generated a personal access token.

Personal access tokens (classic)

Generate new token ▼

Revoke all

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!

✓ ghp_UU54UNb7UGAA7NooCnoo6Vw6rBU0VV4V8Xph

Delete

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Copy this token and go back to the previous directions about using the *push* command. When prompted enter your github username and then when prompted for password paste the token we just generated in.

Then complete the lab as directed in the video.

Packages

In this lab we will work with two packages: **datasauRus** which contains the dataset we'll be using and **tidyverse** which is a collection of packages for doing data analysis in a “tidy” way. These packages are already installed for you. You can load the packages by running the following in the Console.

```
library(tidyverse)
library(datasauRus)
```

Note that the packages are also loaded with the same commands in your R Markdown document.

Data

If it's confusing that the data frame is called ``datasaurus_dozen`` when it contains 13 datasets, you're

The data frame we will be working with today is called **datasaurus_dozen** and it's in the **datasauRus** package. Actually, this single data frame contains 13 datasets, designed to show us why data visualisation is important and how summary statistics alone can be misleading. The different datasets are marked by the **dataset** variable.

To find out more about the dataset, type the following in your Console: `?datasaurus_dozen`. A question mark before the name of an object will always bring up its help file. This command must be ran in the Console.

Exercises

1. Based on the help file, how many rows and how many columns does the **datasaurus_dozen** file have? What are the variables included in the data frame? Add your responses to your lab report.

There are **1846** rows with **3** variables. These variables are **dataset**, **x**, and **y**.

Let's take a look at what these datasets are. To do so we can make a *frequency table* of the dataset variable:

```
datasaurus_dozen %>%
  count(dataset) %>%
  print(13)
```

```
## # A tibble:
## #   13 x 2
##   dataset
##   <chr>
## 1 away
## 2 bullseye
## 3 circle
## 4 dino
## 5 dots
## 6 h_lines
## 7 high_lines
## 8 slant_down
## 9 slant_up
## 10 star
## 11 v_lines
## 12 wide_lines
```

```
## 13 x_shape
## # i 1 more
## #   variable:
## #     n <int>
```

Matejka, Justin, and George Fitzmaurice. "Same stats, different graphs: Generating datasets with varied

The original Datasaurus (`dino`) was created by Alberto Cairo in this great blog post. The other Dozen were generated using simulated annealing and the process is described in the paper *Same Stats, Different Graphs: Generating Datasets with Varied Appearance and Identical Statistics* through Simulated Annealing by Justin Matejka and George Fitzmaurice. In the paper, the authors simulate a variety of datasets that have the same summary statistics as the Datasaurus but have very different distributions.

Knit, commit, and push your changes to GitHub with the commit message "Added answer for Ex 1". Make sure to commit and push all changed files so that your Git pane is cleared up afterwards.

2. Plot `y` vs. `x` for the `dino` dataset. Then, calculate the correlation coefficient between `x` and `y` for this dataset.

Below is the code you will need to complete this exercise. Basically, the answer is already given, but you need to include relevant bits in your Rmd document and successfully knit it and view the results.

Start with the `datasaurus_dozen` and pipe it into the `filter` function to filter for observations where `dataset == "dino"`. Store the resulting filtered data frame as a new data frame called `dino_data`.

```
dino_data <- datasaurus_dozen %>%
  filter(dataset == "dino")
```

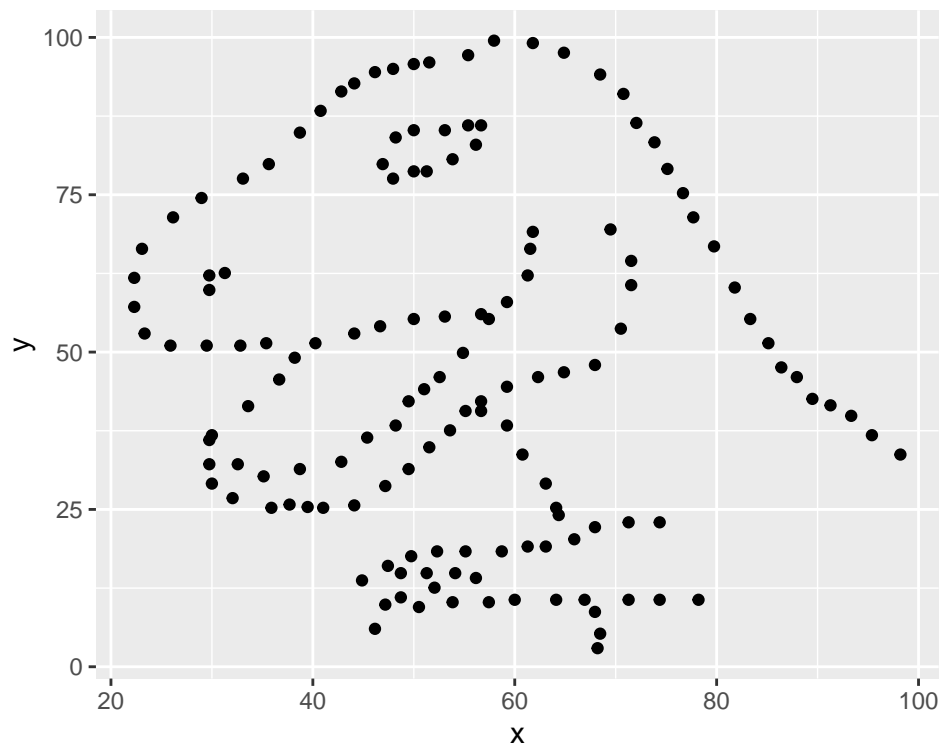
There is a lot going on here, so let's slow down and unpack it a bit.

First, the pipe operator: `%>%`, takes what comes before it and sends it as the first argument to what comes after it. So here, we're saying `filter` the `datasaurus_dozen` data frame for observations where `dataset == "dino"`.

Second, the assignment operator: `<-`, assigns the name `dino_data` to the filtered data frame.

Next, we need to visualize these data. We will use the `ggplot` function for this. Its first argument is the data you're visualizing. Next we define the `aesthetic` mappings. In other words, the columns of the data that get mapped to certain aesthetic features of the plot, e.g. the `x` axis will represent the variable called `x` and the `y` axis will represent the variable called `y`. Then, we add another layer to this plot where we define which `geometric` shapes we want to use to represent each observation in the data. In this case we want these to be points, hence `geom_point`.

```
ggplot(data = dino_data, mapping = aes(x = x, y = y)) +
  geom_point()
```



If this seems like a lot, it is. And you will learn about the philosophy of building data visualizations in layer in detail next week. For now, follow along with the code that is provided.

For the second part of these exercises, we need to calculate a summary statistic: the correlation coefficient. Correlation coefficient, often referred to as r in statistics, measures the linear association between two variables. You will see that some of the pairs of variables we plot do not have a linear relationship between them. This is exactly why we want to visualize first: visualize to assess the form of the relationship, and calculate r only if relevant. In this case, calculating a correlation coefficient really doesn't make sense since the relationship between x and y is definitely not linear – it's dinosaurial!

But, for illustrative purposes, let's calculate the correlation coefficient between x and y .

Start with `dino_data` and calculate a summary statistic that we will call `r` as the `cor` relation bet

```
dino_data %>%
  summarize(r = cor(x, y))
```

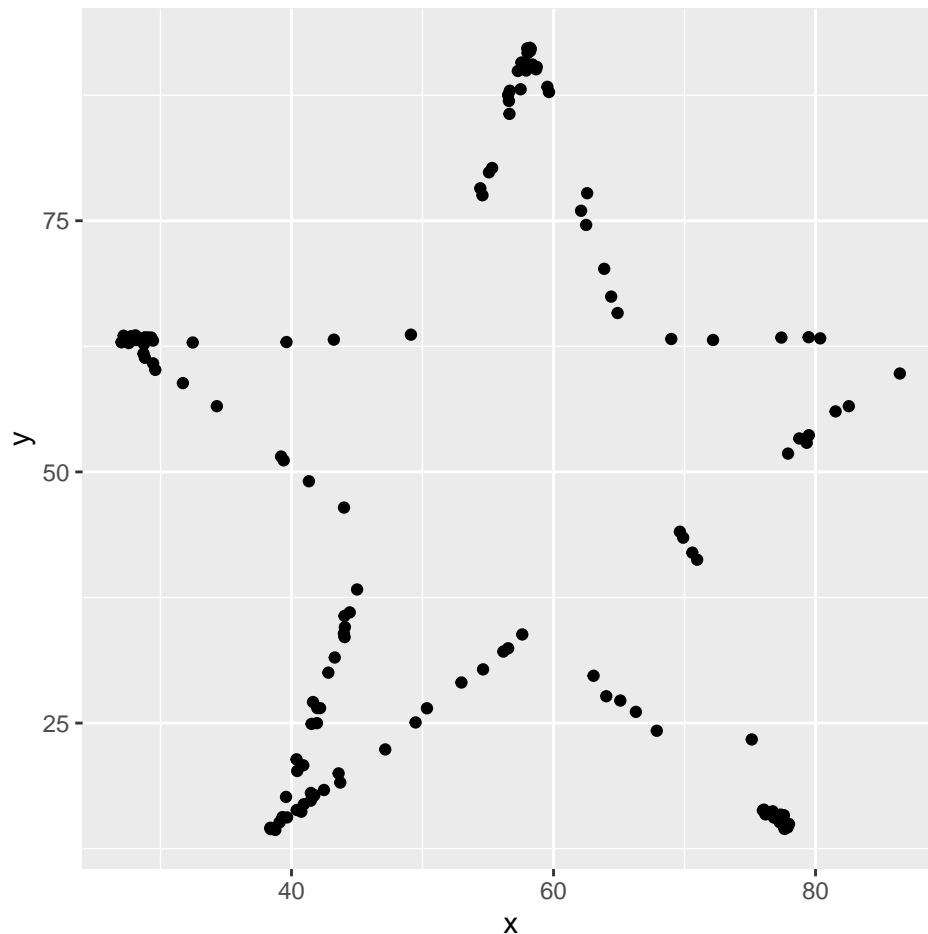
```
## # A tibble: 1 x 1
##       r
##   <dbl>
## 1 -0.0645
r = -0.06447185
```

Knit, commit, and push your changes to GitHub with the commit message “Added answer for Ex 2”. Make sure to commit and push all changed files so that your Git pane is cleared up afterwards.

3. Plot y vs. x for the `star` dataset. You can (and should) reuse code we introduced above, just replace the dataset name with the desired dataset. Then, calculate the correlation coefficient between x and y for this dataset. How does this value compare to the r of `dino`?

```
star_data <- datasaurus_dozen %>%
  filter(dataset == "star")
```

```
ggplot(data = star_data, mapping = aes(x = x, y = y)) +
  geom_point()
```



```
star_data %>%
  summarize(r = cor(x, y))
```

```
## # A tibble: 1 x 1
##       r
##   <dbl>
## 1 -0.0630
```

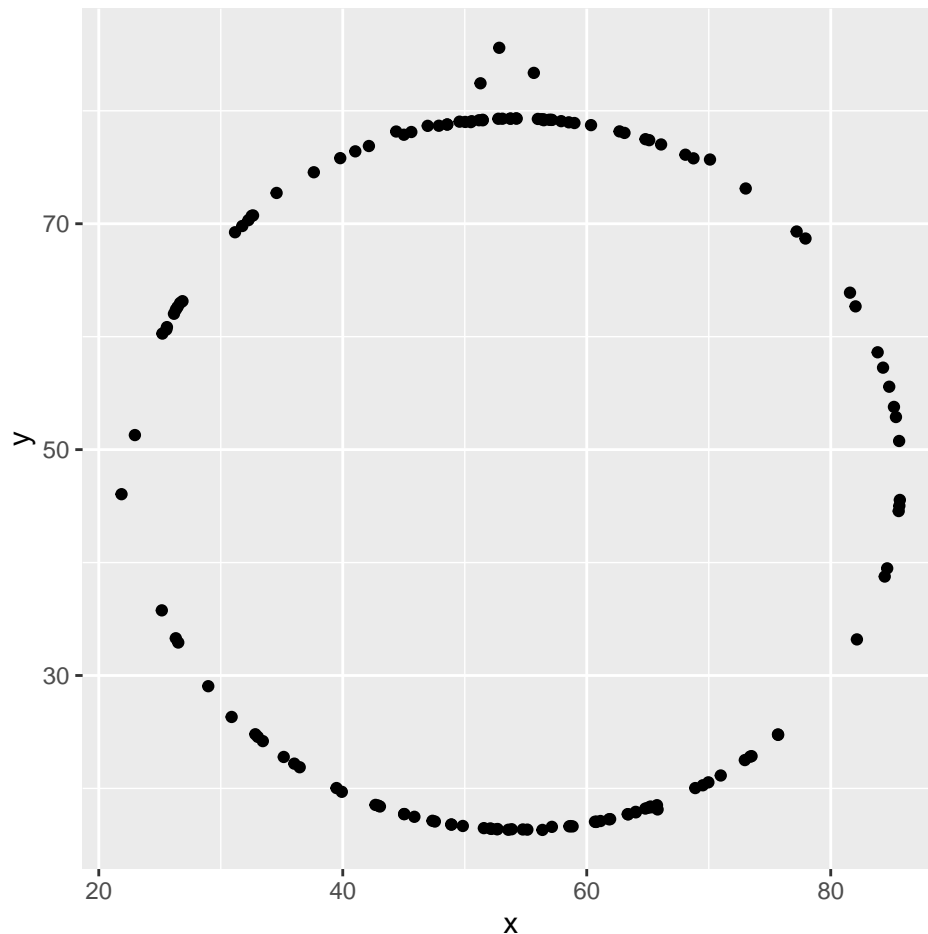
$r = -0.0629611$ The r for `star` is just slightly larger than the r for `dino`. However, they are both negative and within 0.0015 of each other.

This is another good place to pause, knit, commit changes with the commit message “Added answer for Ex 3”, and push. Make sure to commit and push all changed files so that your Git pane is cleared up afterwards.

4. Plot y vs. x for the `circle` dataset. You can (and should) reuse code we introduced above, just replace the dataset name with the desired dataset. Then, calculate the correlation coefficient between x and y for this dataset. How does this value compare to the r of `dino`?

```
circle_data <- datasaurus_dozen %>%
  filter(dataset == "circle")
```

```
ggplot(data = circle_data, mapping = aes(x = x, y = y)) +  
  geom_point()
```



```
circle_data %>%  
  summarize(r = cor(x, y))
```

```
## # A tibble: 1 x 1  
##       r  
##   <dbl>  
## 1 -0.0683
```

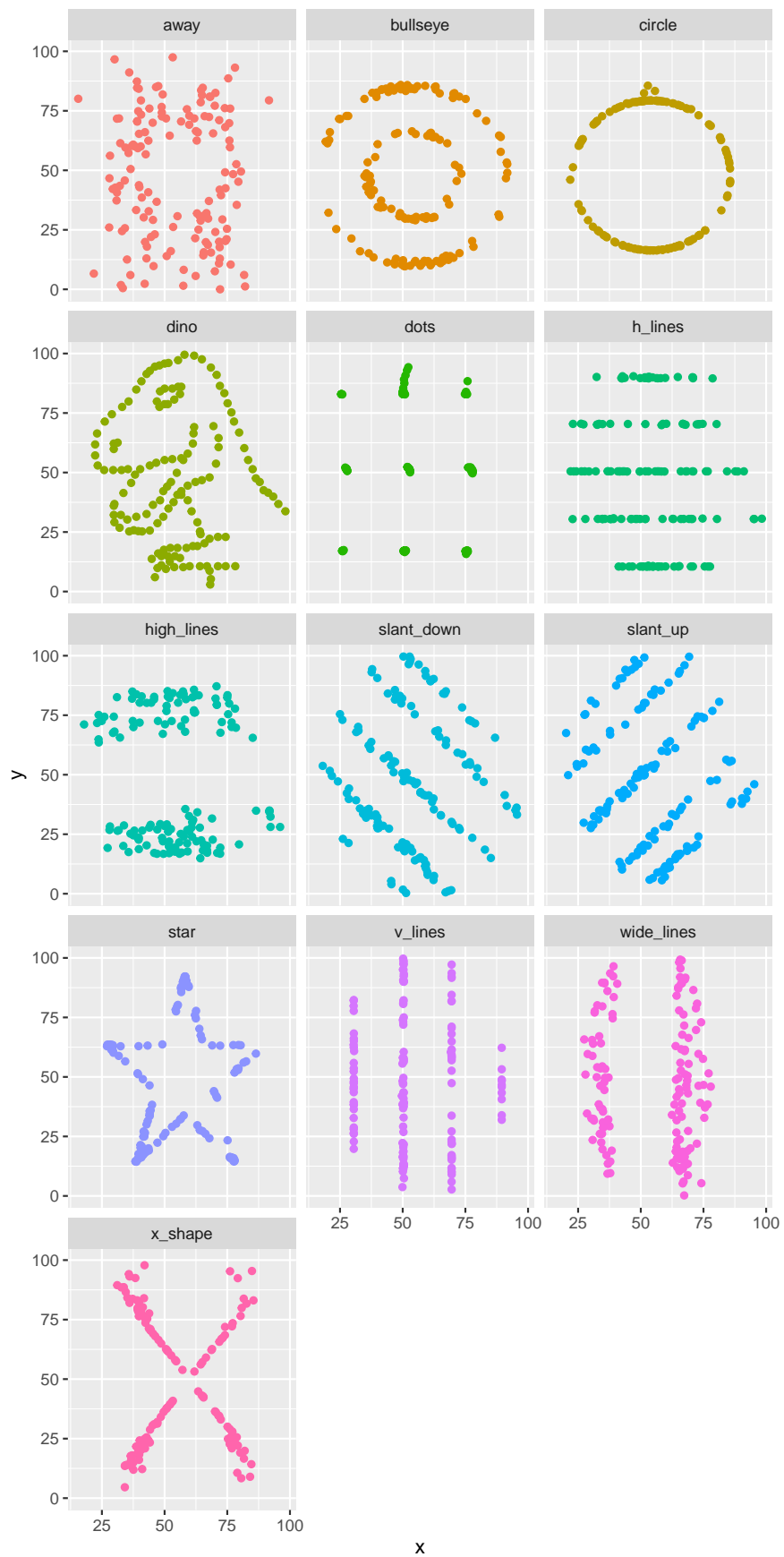
$r = -0.06834336$ The r of circle is slightly smaller than the r of dino. They are both negative.

You should pause again, commit changes with the commit message “Added answer for Ex 4”, and push. Make sure to commit and push all changed files so that your Git pane is cleared up afterwards.

Facet by the dataset variable, placing the plots in a 3 column grid, and don't add a legend.

5. Finally, let's plot all datasets at once. In order to do this we will make use of faceting.

```
ggplot(datasaurus_dozen, aes(x = x, y = y, color = dataset)) +  
  geom_point() +  
  facet_wrap(~ dataset, ncol = 3) +  
  theme(legend.position = "none")
```



And we can use the `group_by` function to generate all the summary correlation coefficients.

```
datasaurus_dozen %>%  
  group_by(dataset) %>%  
  summarize(r = cor(x, y)) %>%  
  print(13)
```

You're done with the data analysis exercises, but we'd like you to do two more things:

Edit R Markdown Document Options

Output Format: HTML
Recommended format for authoring (you can switch to PDF or Word output anytime).

GeneralFiguresAdvanced

Default figure width in inches:

Default figure height in inches:

☐ Render figures with captions

OK

Cancel

- **Resize your figures:**

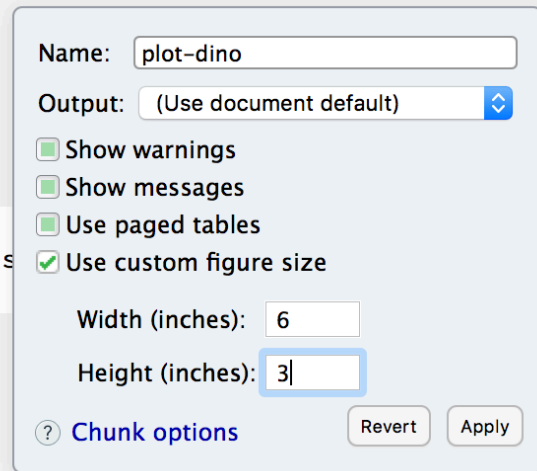
Click on the gear icon in on top of the R Markdown document, and select “Output Options...” in the dropdown menu. In the pop up dialogue box go to the Figures tab and change the height and width of the figures, and hit OK when done. Then, knit your document and see how you like the new sizes. Change and knit again and again until you're happy with the figure sizes. Note that these values get saved in the YAML.


```
fig.height=3, fig.width=6}
asaurus_dozen %>%
  == "dino")

no_data, mapping = aes(x = x, y = y)) +

te the correlation between `x` and `y` in this

%>%
== "dino") %>%
cor(x, y))
```



Name:

Output:

☐ Show warnings

☐ Show messages

☐ Use paged tables

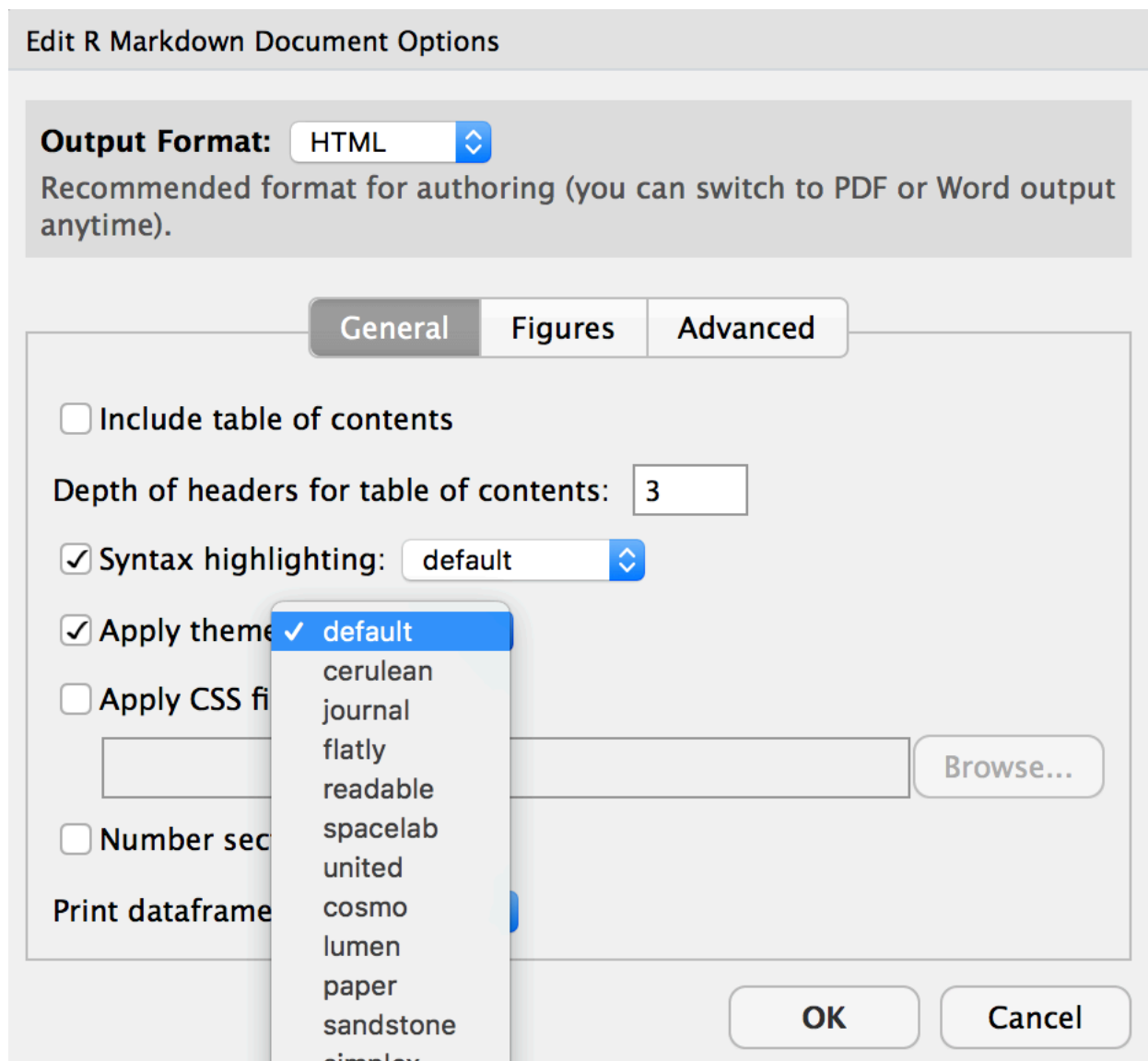
☒ Use custom figure size

Width (inches):

Height (inches):

[? Chunk options](#)

You can also use different figure sizes for different figures. To do so click on the gear icon within the chunk where you want to make a change. Changing the figure sizes added new options to these chunks: `fig.width` and `fig.height`. You can change them by defining different values directly in your R Markdown document as well.



- Change the look of your report:

Once again click on the gear icon in on top of the R Markdown document, and select “Output Options...” in the dropdown menu. In the General tab of the pop up dialogue box try out different Syntax highlighting and theme options. Hit OK and knit your document to see how it looks. Play around with these until you’re happy with the look.

Not sure how to use emojis on your computer? Maybe a teammate can help? Or you can ask your TA as well!

Yay, you’re done! Commit all remaining changes, use the commit message “Done with Lab 1! “, and push. Make sure to commit and push all changed files so that your Git pane is cleared up afterwards. Before you wrap up the assignment, make sure all documents are updated on your GitHub repo.