Lab 01 - Hello R!

Max Asmar

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

git is a version control system (like "Track Changes" features from Microsoft Word on steroids) and GitHub is the home for your Git-based projects on the internet (like DropBox but much, much better).

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.

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.

Learning Objectives

By the end of this lab, you will be able to:

- Navigate the RStudio interface
- Edit and knit R Markdown documents
- Use Git for version control (commit and push changes)
- Load R packages
- Create basic visualizations with ggplot2
- Calculate summary statistics
- Understand why visualization is important for data analysis

Before You Begin

Prerequisites: - You have a GitHub account - You have accepted the invitation to the course GitHub organization - You can access JupyterHub and RStudio - You have watched the "Meet the toolkit" lecture videos

If you haven't completed these prerequisites, stop here and complete them first!

Getting Started

Important: This lab should be completed in your **forked lab-instructions repository** that you set up at the beginning of the semester. If you haven't forked the repository yet, go back to the README in the lab-instructions repo and follow those instructions first.

Step 1: Navigate to the Lab

1. Open JupyterHub and launch RStudio
2. Navigate to your `lab-instructions` repository
3. Open the `Lab01` folder
4. Open the file `lab-01-hello-r.Rmd`

Step 2: Verify Setup

Before you begin, make sure: - The file opens in the RStudio editor (top left pane) - You can see the Git pane (top right) - The Git pane shows your repository name with YOUR username (not the course organization)

If the Git pane shows the course organization name instead of yours, you need to fork the repository first!

Step 3: Test Knit

Click the **Knit** button to make sure the document compiles without errors. You should see an HTML output.

If knitting fails:

- Check that you have the required packages installed
- Make sure you're in the correct working directory
- Ask for help if you can't resolve the error

Warm up

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

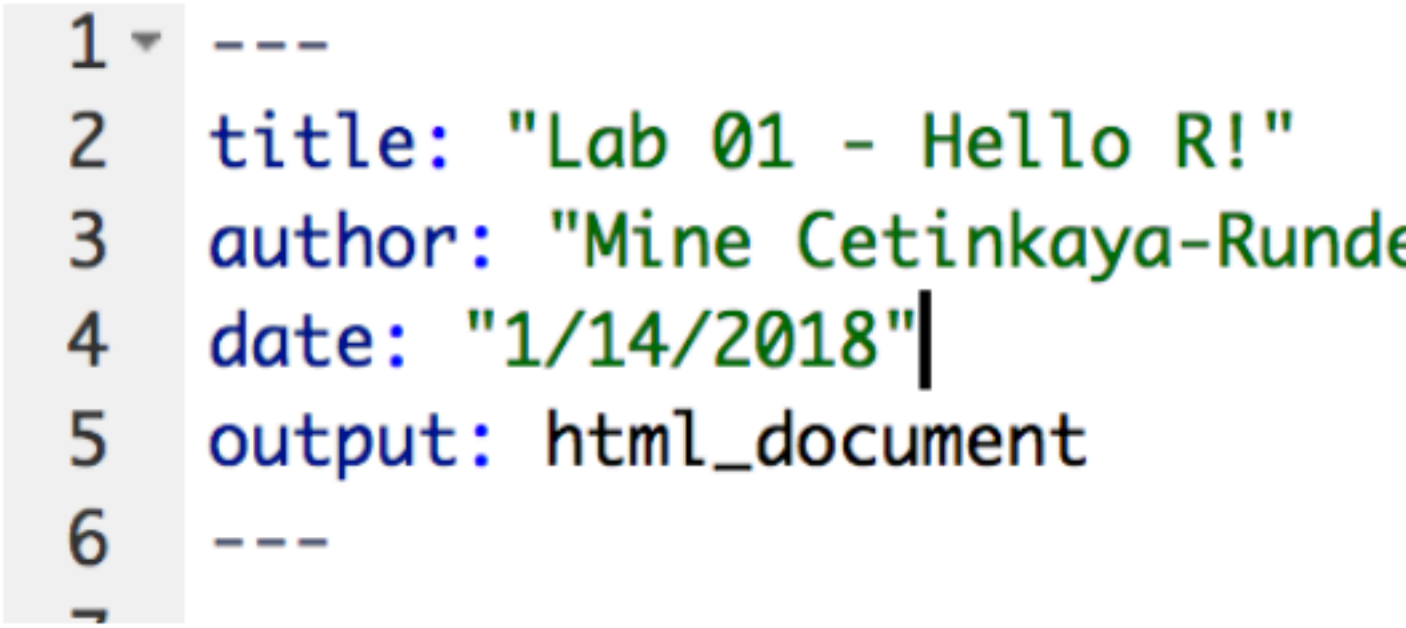
YAML

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

The top portion of your R Markdown file (between the three dashed lines) is called YAML. It stands for "YAML Ain't Markup Language". It is a human friendly data serialization standard for all programming languages. All you need to know is that this area is called the YAML (we will refer to it as such) and that it contains meta information about your document.

Committing changes

Then go to the Git pane in your RStudio.



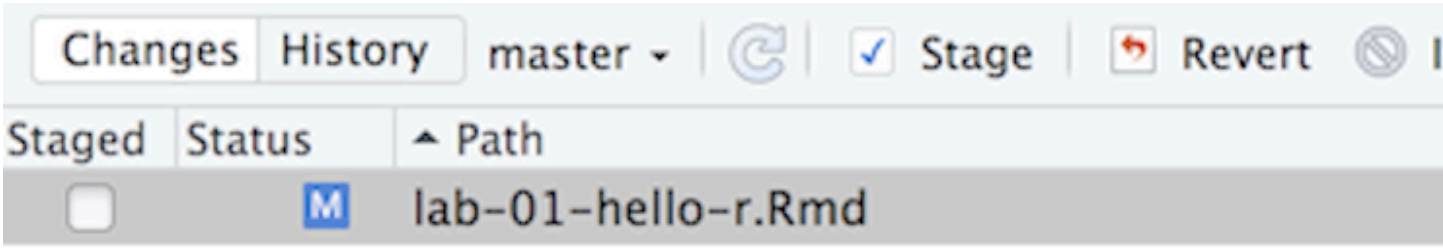
If you have made changes to your Rmd file, you should see it listed here. Click on it to select it in this list and then click on **Diff**. This shows you the *diff*erence between the last committed state of the document and its current state that includes your changes. If you're happy with these changes,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

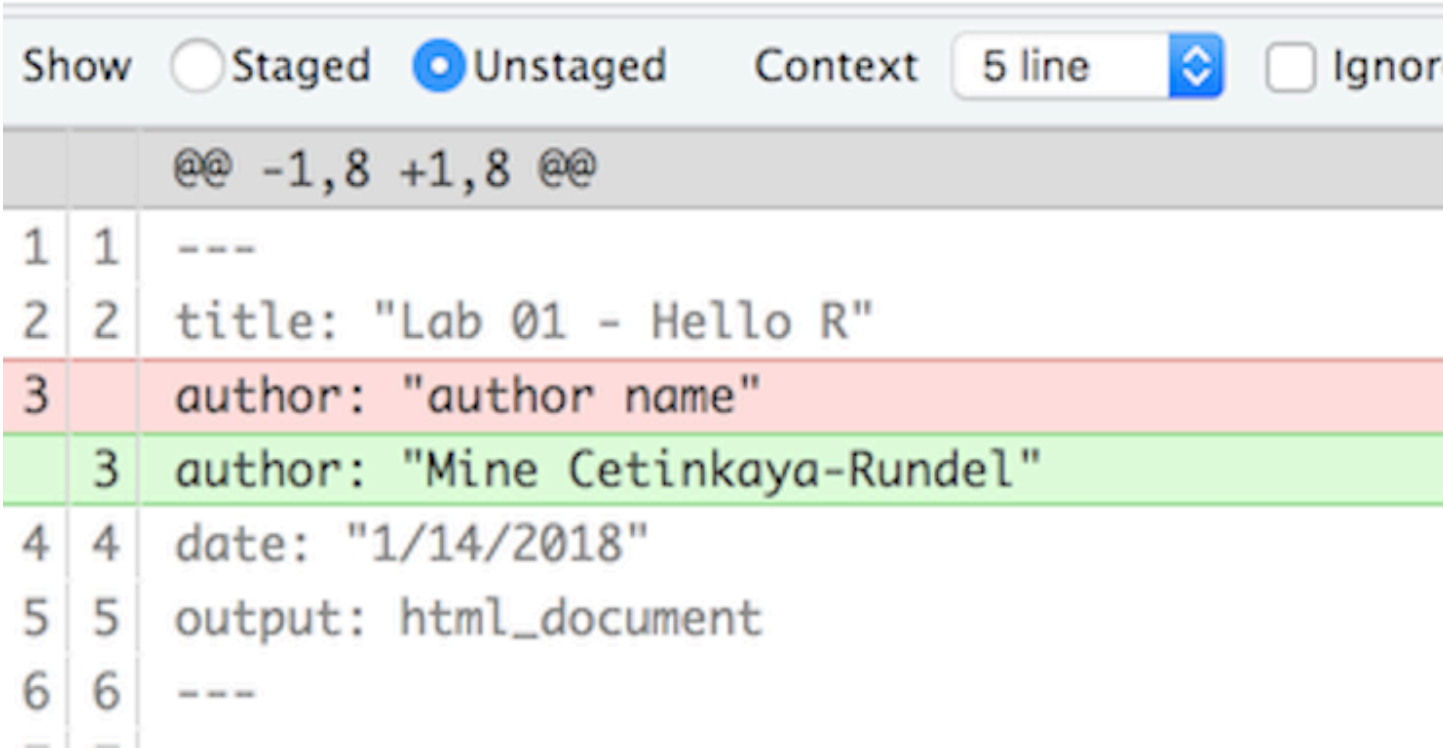
You don't have to commit after every change, this would get quite cumbersome. You should consider committing states that are *meaningful to you* for inspection, comparison, or restoration. In the first few assignments we will tell you exactly when to commit and in some cases, what commit message to use.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


Changes
History
master
Stage
Revert
Staged
Status
Path
M
lab-01-hello-r.Rmd


Show
Staged
Unstaged
Context
5 line
@@ -1,8 +1,8 @@
---
title: "Lab 01 - Hello R"
author: "author name"
author: "Mine Cetinkaya-Rundel"
date: "1/14/2018"
output: html_document
---

name, and then your password. This might feel cumbersome. Bear with me… We *will* teach you how to save your password so you don't have to enter it every time. But for this one assignment you'll have to manually enter each time you push in order to gain some experience with it.

Why push? Your local commits are only on your computer until you push them to GitHub. Pushing makes them visible in your online repository.

Checkpoint: Go to your forked repository on GitHub (in your web browser) and refresh the page. You should see your updated file with your name in it!

The Workflow Pattern:

Throughout this course, you'll see this pattern:

Knit → Commit → Push

Get comfortable with it - you'll use it in every lab and homework assignment!

Packages

Packages are like apps for R - they add new functions and capabilities. We load them at the beginning of our analysis.

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)
```

How to run this code: 1. Click in the Console (bottom left pane) 2. Type (or copy/paste) the code above 3. Press Enter

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 not alone! Have you heard of a baker's dozen?

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 visualization 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.

Pro tip: A question mark before the name of any object will bring up its help file. This only works in the Console, not in your R Markdown document.

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 13 rows and 2 columns in the datasaurus_dozen file.

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>
```

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*

Matejka, Justin, and George Fitzmaurice. "Same stats, different graphs: Generating datasets with varied appearance and identical statistics through simulated annealing." Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. ACM, 2017.

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"`.

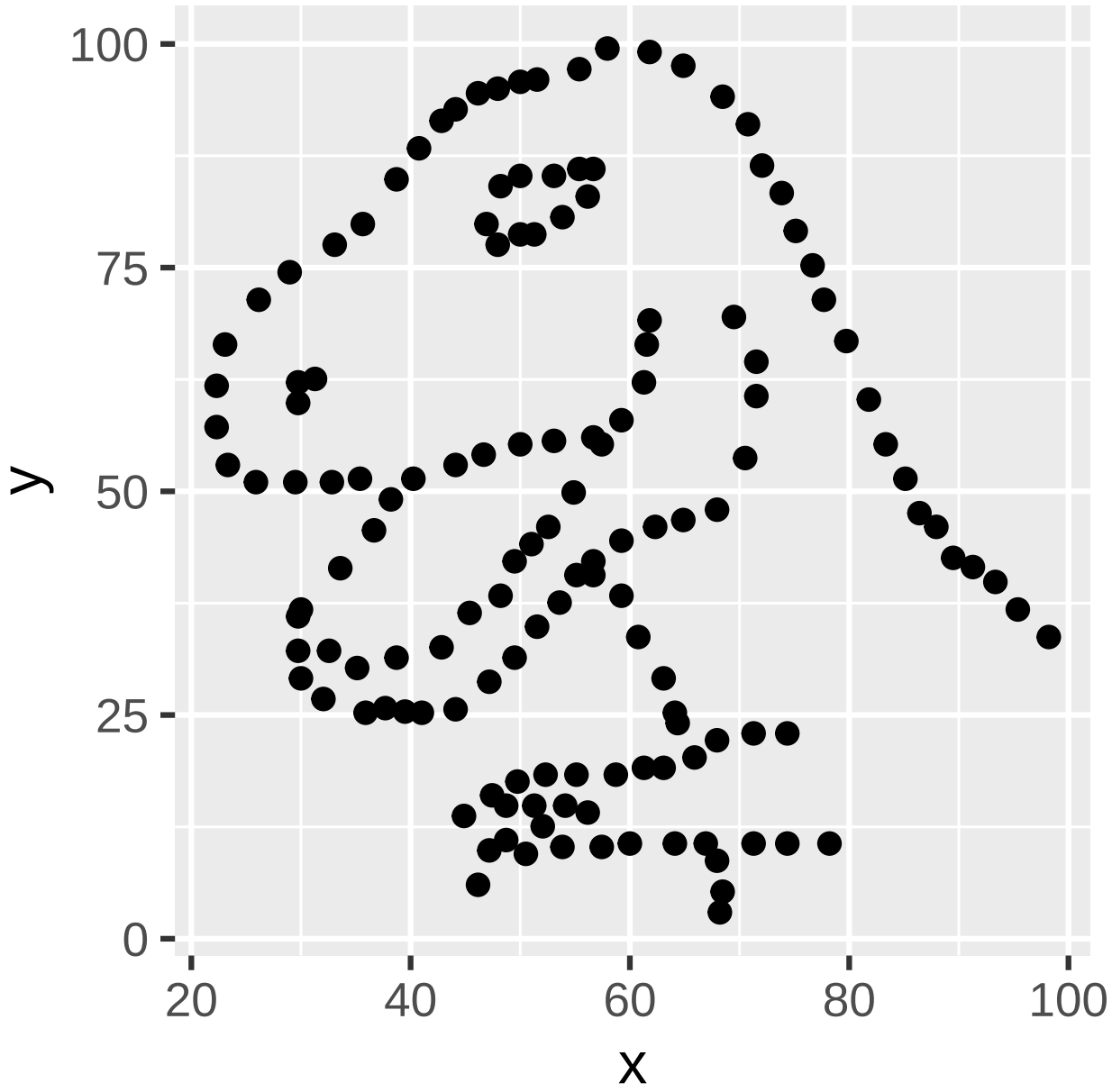
Second, the assignment operator: `<-`, assigns the name `dino_data` to the filtered data frame.

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 `aes`thetic 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 `geom`etric shapes we want to use to represent each observation in the data. In this case we want these to be points, hence `geom_point`.

Here is the graph of x and y for the "dino" dataset.

```
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.


100
75
50
25
0
y
20
40
60
80
100
x

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 between `x` and `y`.

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

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

The correlation coefficient between x and y is -0.0645.

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`?

Your task: Adapt the code from Exercise 2, changing `"dino"` to `"star"`.

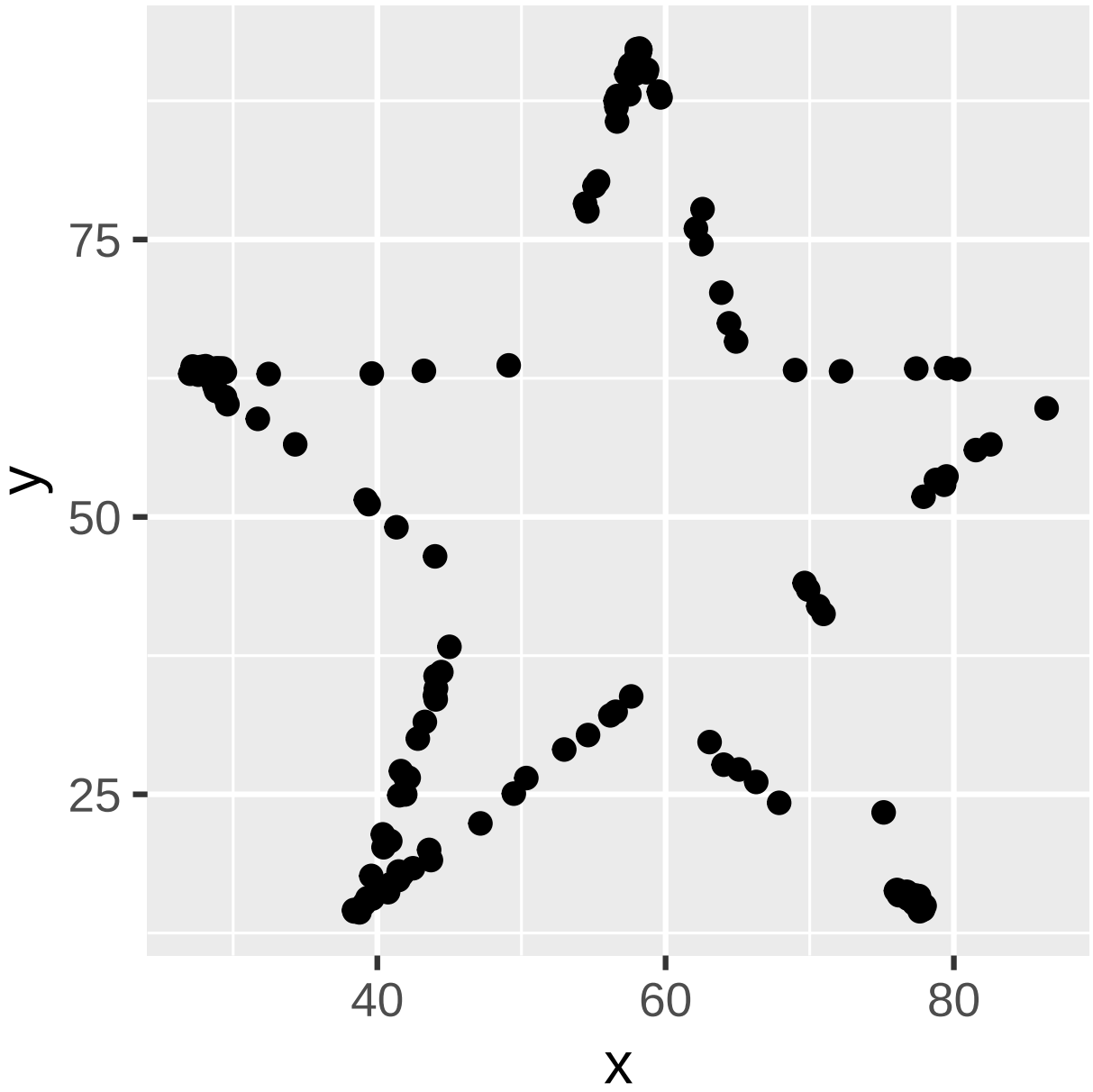
Step 1: Filter for star data

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

Step 2: Create the plot

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

Step 3: Calculate correlation


75
50
25
y
40
60
80
x

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

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

Write your answers:

- The correlation coefficient for the star dataset is: -0.0630
- Compared to the dino dataset (r = -0.06447185), the star dataset's correlation is: Greater than the correlation coefficient of the dino dataset.

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`?

Your task: Adapt the code from Exercises 2 and 3, changing the dataset to `"circle"`.

Filter, plot, and calculate correlation:

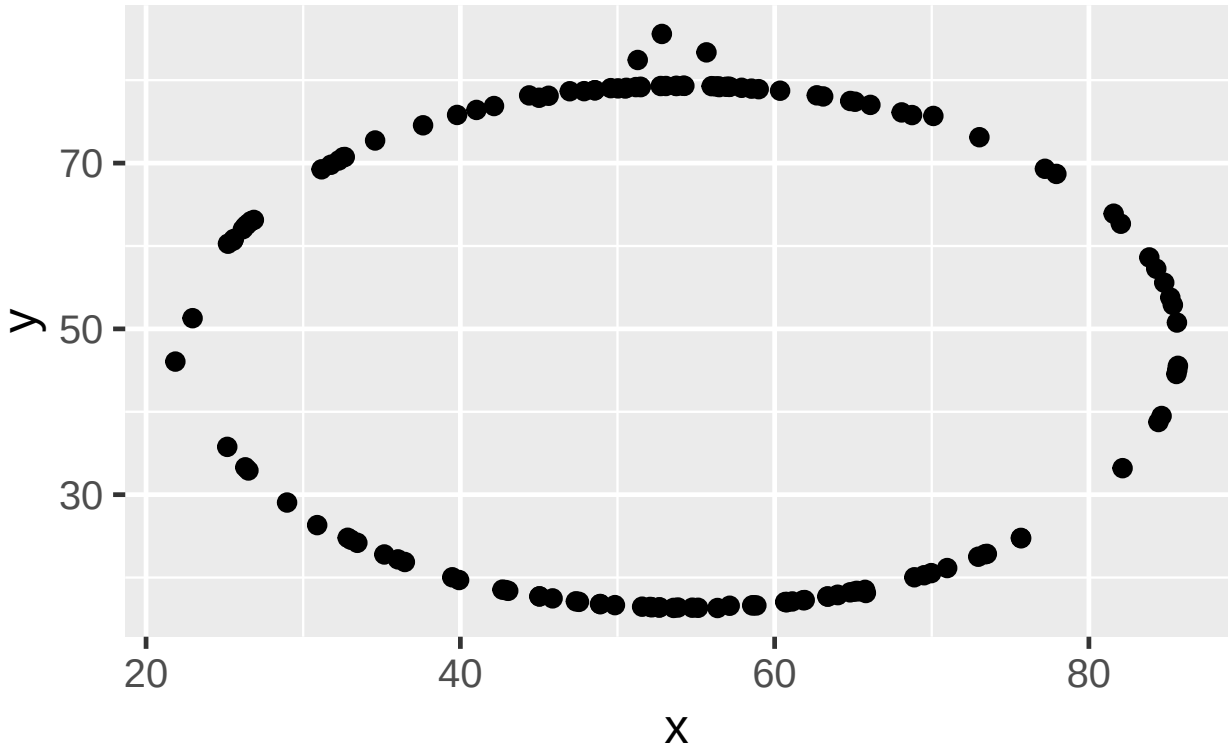
```
circle_data <- datasaurus_dozen %>%
  filter(dataset == "circle")
```

```
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
```

Write your answers:



- The correlation coefficient for the circle dataset is: -0.0683
- Compared to the dino dataset, the circle dataset's correlation is: Less than the correlation coefficient of the dino dataset.
- What do you notice about the correlation coefficients for dino, star, and circle? I notice that the coefficients for all 3 sets are close to -0.06. More specifically they are in between -0.06 and -0.07.

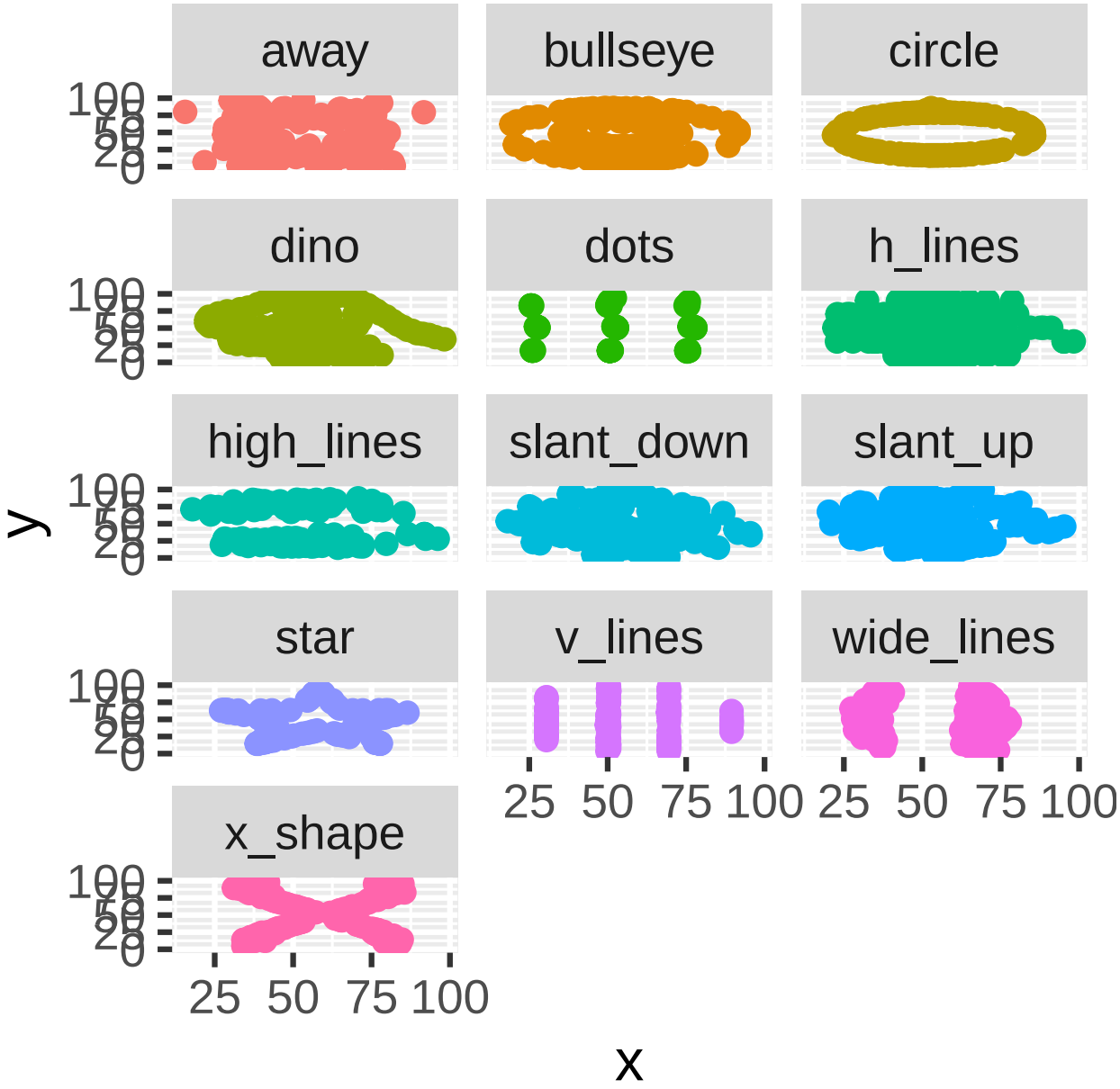
You should pause again, commit changes with the commit message "Added answer for Ex 4", and push. Make sure to commit and push all changed files so that your Git pane is cleared up afterwards.

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

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

```
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.


away
bullseye
circle
dino
dots
h_lines
high_lines
slant_down
slant_up
star
v_lines
wide_lines
x_shape
y
x
0
25
50
75
100

```
datasaurus_dozen %>%
  group_by(dataset) %>%
  summarize(r = cor(x, y)) %>%
  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:
## #   r <dbl>
```

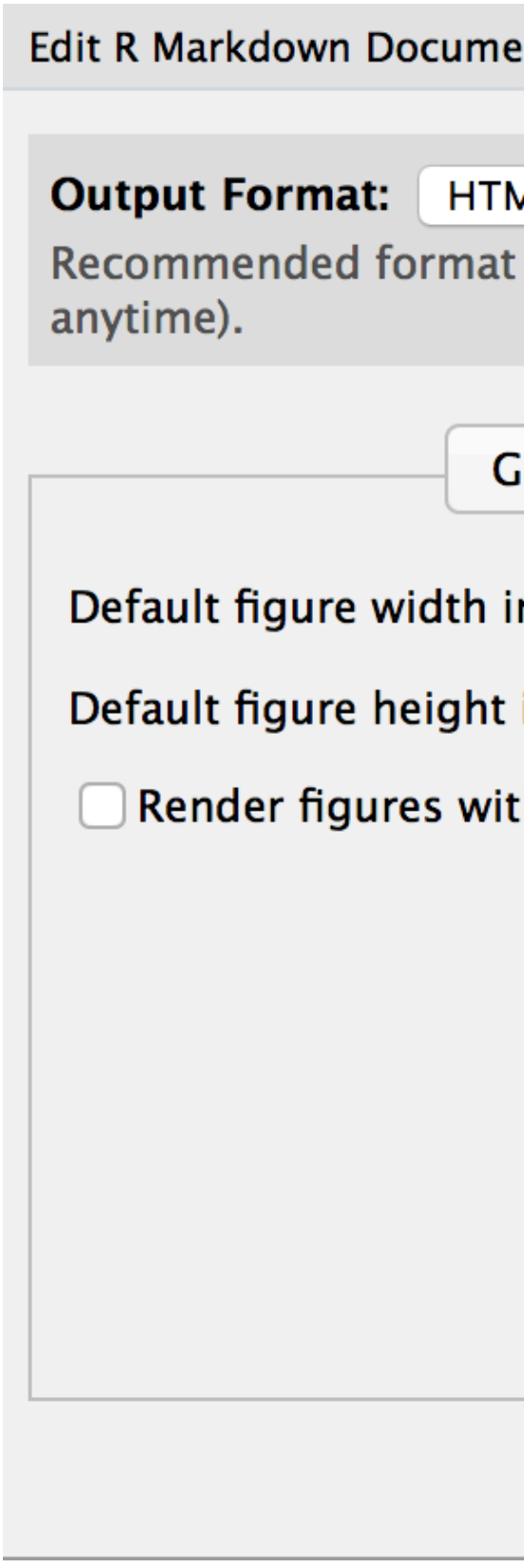
You're done with the data analysis exercises, but we'd like you to do two more things:

- **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.

You can also use different figure sizes for differen 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.