Hello, R!

Dr. Shoemaker

Read all instructions carefully, fill in the provided template as described in this document. Your repository will need to include your in your knit document (you'll learn what that means soon) and should have the commits and changes as described in this document.

Intro

The main goal of this lab is to get you working with R, RMarkdown 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 let you practice using git and GitHub, the collaboration and version control system.

As the semester progresses, 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.

Getting Started

Hopefully you've followed the instructions in the assignment description in Blackboard! If not, head back there and get this opened in a version-controlled RProject.

Packages

A quick peek at the future, we're going to be using some `packages`. R uses packages as a way for users to share code and data with each other in a *reproducible* way. All packages are freely available, and have documentation that tells you how to use the code and data inside.

In this lab, we'll use `datasauRus` (which contains the data we'll use) and `tidyverse` (a collection of packages for doing data analysis in a "tidy" way).

We'll install these packages by running the following commands in the console:

Install these packages by running the following in the console.

```
install.packages("tidyverse")
install.packages("datasauRus")
```

Now that the necessary packages are installed, you should be able to Knit your document and see the results!

(If you'd like to run your code in the console as well you'll also need to load the packages there. To do so, run the `library` commands that you see in the RMarkdown document in the console)

Some Housekeeping

(We did this in class already, but we'll look again now to make sure.) To try and make sure git plays nicely with pushing and pulling from RStudio, we'll configure our user information.

Go to the RStudio terminal (a tab next to the console), and run each of these commands

```
git config --global user.email
git config --global user.name
```

If these are your name and email (the email for your github account), move on. If not, run them again with your information as an additional argument. For example, mine would be:

```
git config --global user.email "katherine.shoemaker@gmail.com"
git config --global user.name "Katherine Shoemaker"
```

Warm up

Let's start with some simple exercises to get comfortable.

YAML

Open the `RProj1_Template.Rmd` file in your project. You can open a file by clicking on it in the "files" pane on the bottom right of RStudio.

The top portion of this file is called the YAML (YAML Ain't Markup Language). It contains meta information about your document. Change the author name to your name, and "knit" the file (this is the button with the blue yarn ball... get it? knit?). You'll see a new tab on your bottom left, and the html will pop up. Now you should see the document and you're the author!

Committing Changes

Now, go to the Git pane in your RStudio. (it's in the top right).

If you have made changes to your Rmd file (and you should have, as you were asked to above), 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 you make. You should consider committing states that are meaningful to you for inspection, comparison, or restoration.

Pushing

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 your changes are available to you if you're working on other computers.

In order to push your changes to GitHub, click on Push. It's a little green up arrow. This will prompt a dialogue box where you first need to enter your user 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.

The Data!

The data frame we will be working with today is called `datasaurus_dozen` and it's in the `datasauRus` package. This single data frame contains a baker's dozen 13 datasets, designed to show us why data visualisation is important and how summary statistics alone can be misleading. The different datasets are maked 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.

Exercise 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. When you're done, commit your changes with the commit message "Added answer for Ex 1", and push.

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
## # ... with 1
## #   more
## #   variable:
## #   n <int>
```

Note: 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 the same summary statistics to the Datasaurus but have very different distributions.

Data Visualization and Summary

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

Don't panic. The code you need for this is below. Basically, the answer is already given, but you need to include the relevant parts 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")
```

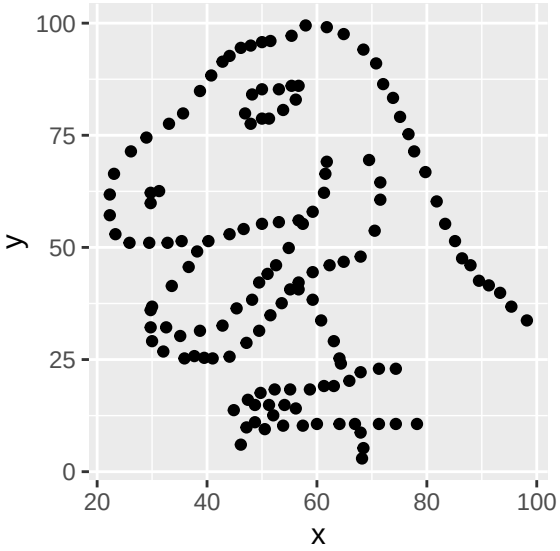
There is a lot going on here, so let's slow down and unpack it a bit.

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`.

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



If this seems like a lot, it is. I'm just using tools that we are learning later in order to show how these reports work.

For the second part of this 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 correlation coefficient between `x` and `y`.

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

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

This is a good place to pause, commit changes with the commit message "Added answer for Ex 2", and push.

Exercise 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?

Once this is done, this is a good place to pause, commit changes with the commit message "Added answer for Ex 3", and push.

Exercise 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?

You should pause again, commit changes with the commit message "Added answer for Ex 4", and push.

The Grand Finale

A little advanced plotting to finish it all off...

Exercise 5 Finally, let's plot all datasets at once and calculate their summary correlation coefficients. In order to do this we will make use of *facetting*. The code you'll need is below.

```
ggplot(datasaurus_dozen, aes(x = x, y = y, color = dataset))+
  geom_point()+
  facet_wrap(~ dataset, ncol = 3) + # facet the dataset variable, in 3 cols
  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))
```

```
## # A tibble: 13 x 2
##    dataset          r
##    <chr>        <dbl>
##  1 away       -0.0641
##  2 bullseye   -0.0686
##  3 circle     -0.0683
##  4 dino       -0.0645
##  5 dots       -0.0603
##  6 h_lines    -0.0617
##  7 high_lines -0.0685
##  8 slant_down -0.0690
##  9 slant_up   -0.0686
## 10 star       -0.0630
## 11 v_lines    -0.0694
## 12 wide_lines -0.0666
## 13 x_shape    -0.0656
```

And you're done! BUT I'd like for 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 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. (Unfortunately, I won't give you bonus points for using `kate`)

Yay, you're done! Commit all remaining changes, use the commit message "Done with Homework 1!", and push. Before you wrap up the assignment, make sure all documents are updated on your GitHub repo. Turn in your final version of your html to Blackboard, along with a link to your repo, which should have the commits as described in this document.