Lab 2: Datasaurus

Ziji Zhou

Due by 10pm ET Friday

Instructions

This problem set is a modified version of a lab developed by Mine Cetinkaya-Rundel.

The main goal of this lab is to introduce you to working with R and RStudio in conjunction with git and GitHub, all of which we will be using throughout the course.

We will work with two packages: `datasauRus` which contains the dataset for the lab, and `tidyverse` which is a collection of packages for doing data analysis in a "tidy" way. Both packages are ready to be loaded in the `setup` code chunk.

Step 1: Verify you are working on the lab saved in *your* repo and that you are working in *your* project repo.

Look in the top right of RStudio to verify the project name matches *your* repo.

Knit the Rmd file (`Cmd + Shift + K` on Mac or `Ctrl + Shift + K` on PC) and verify the pdf (or other relevant files) is created in *your* repo and not some other location.

Step 2: Change your name in the YAML header.

The More You Know...: "YAML" rhymes with "camel" and originally stood for "Yet Another Markup Language" but now stands for "YAML Ain't Markup Language"

Step 3: Knit, view diff, and commit!

Knit the Rmd file (`Cmd + Shift + K` on Mac or `Ctrl + Shift + K` on PC), then go to the **Git** pane.

Select this Rmd file (just the Rmd) in the Git Pane, and then click **Diff**—this shows you the *diff*erence between the last committed state of the document and its current state.

If you're happy with the changes, write "Update author name" in the **Commit message** box, and hit **Commit**.

How often should you commit?? You do not 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, I will tell you exactly when to commit and, in some cases, what commit message to use. As the semester progresses, I will let you make these decisions.

Step 4: Ahh, push it!

After committing, go ahead and **Push** your commit(s) back onto GitHub.

You can think of commits as snapshots of your work over time, and pushing will sort of sync your work with GitHub so you (or a collaborator) can pick up where you left off but on another device.

Edit, Commit, and Push until done!

When you think you are done with the assignment, save the pdf as "*Name_thisfilename_date*.pdf" before committing and pushing (this is generally good practice but also helps me in those times where I need to download all student homework files).

Gradescope Upload

This assignment is designed to ensure you've successfully submitted an assignment to Gradescope before the first real assignment is due. No feedback will be provided on this assignment. The typical Gradescope instructions that will be found on your assignments is provided below.

For each question (e.g., 3.1), allocate all pages associated with the specific question. If your work for a question runs onto a page that you did not select, you may not get credit for the work. If you do not allocate *any* pages when you upload your pdf, you may get a zero for the assignment.

You can resubmit your work as many times as you want before the deadline, so you should not wait until the last minute to submit some version of your work. Unexpected delays/crises that occur on the day the assignment is due do not warrant extensions (please submit whatever you have done to receive partial credit).

Data

The dataset we will be working with is called *datasaurus_dozen* and it's in the **datasauRus** package. Actually, this single dataset contains 13 datasets (have you heard of a baker's dozen?), designed to show us why data visualisation is important and how summary statistics alone can be misleading. The different datasets are identified 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 run in the Console.

Problem 1 **Understanding the data** It is tempting to jump into visualization and analysis (much like we did on the first day), but it is critical as a statistician that we first understand the context and structure of the data.

1.1 Based on the help file, how many rows and how many columns are in the data frame?

There are 3 columns with 1846 rows.

1.2 Based on the help file, what variables are included in the data frame? Use Markdown formatting to provide a bulleted list, with each variable typeset in monotype font.

- 'dataset': describes the origin of the dataset
- 'x': x-value of dataset
- 'y': y-value of dataset

1.3 How many observations are in each dataset within this larger data frame? Let's make a *frequency table* of the of the `dataset` variable to find out.

> *Note 1*: The `kable()` function makes nicer looking tables when you knit. For quick "professional quality" pdf tables, add the argument `booktabs = TRUE`. Try knitting the document without piping the frequency table to `kable()`, with the pipe to `kable()`, and finally with the pipe to `kable(booktabs = TRUE)` to see the differences.

```
datasaurus_dozen %>%
  count(dataset) %>%
  kable(booktabs = TRUE)
```

dataset	n
away	142
bullseye	142
circle	142
dino	142
dots	142
h_lines	142
high_lines	142
slant_down	142
slant_up	142
star	142
v_lines	142
wide_lines	142
x_shape	142

> *Note 2*: Annoyed by the left-justified table above? You can pipe to an additional function called `kable_styling()` which allows further customization of pdf tables. By default, this function will center the table. It may also do a weird LaTeX thing where the table ends up somewhere else in the document other than where you want it. To prevent that from happening, we can add the argument `latex_options = "hold_position"` (or `"HOLD_position"` if you're *really* serious).

Problem 2 **Data visualization and summary**

2.1 Calculate the correlation coefficient, r, between x and y for the dino 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. *What does this correlation coefficient tell us about the relationship between x and y in the dino dataset?*

```
# Start with `datasaurus_dozen`
# 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")

# Compute correlation between `x` and `y` for `dino` dataset with label `r`
dino_data %>%
  summarize(r = cor(x, y))
```

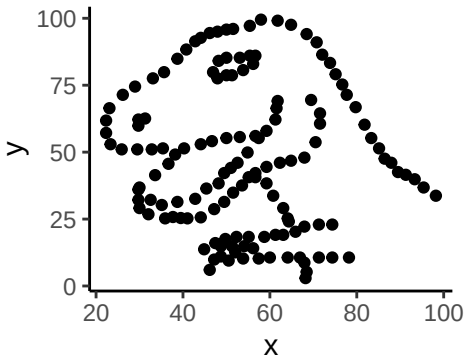
```
# A tibble: 1 x 1
        r
    <dbl>
1 -0.0645
```

The r value shows a very slight negative correlation between x and y that is almost negligible.

2.2 Now, plot y vs. x for the dino dataset using the ggplot() function. 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. *What do you notice now about the relationship between x and y?*

> *Note*: Overwhelmed by this information?! Don't worry!! You will learn about the philosophy of building data visualizations in detail next week. For now, follow along with the code that is provided.

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



There seems to be an overall downward trend of the points, as reflected in the slightly negative r value.

2.3 Now calculate the correlation coefficient between x and y for the star dataset. You can (and should) reuse code we introduced above, just replace the dataset name with the desired dataset. How does this value compare to the correlation coefficient from the dino dataset?

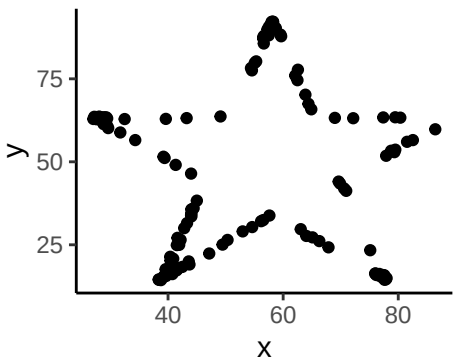
```
# Start with `datasaurus_dozen`
# Filter for observations where `dataset == "dino"`
# Store the resulting filtered data frame as a new data frame called `dino_data`
star_data <- datasaurus_dozen %>%
  filter(dataset == "star")

# Compute correlation between `x` and `y` for `dino` dataset with label `r`
star_data %>%
  summarize(r = cor(x, y))
```

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

The star dataset produces a very similar r value to the dino dataset.

2.4 Plot y versus x for the star dataset. Does the plot look the same as the plot of the dino data?

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



Looks different than the dino plot, draws the shape of a star.

2.5 Now calculate the correlation coefficient between x and y for the circle dataset. You can (and should) reuse code we introduced above, just replace the dataset name with the desired dataset. How does this value compare to the correlation coefficients from the other two datasets?

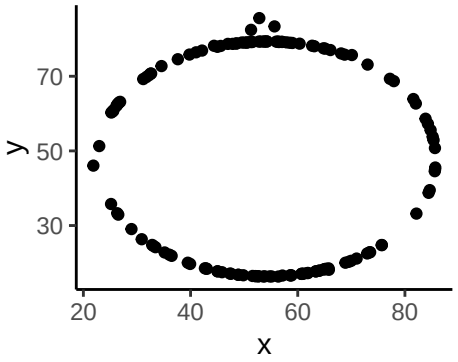
```r
# Start with `datasaurus_dozen`
# Filter for observations where `dataset == "dino"`
# Store the resulting filtered data frame as a new data frame called `dino_data`
circle_data <- datasaurus_dozen %>%
  filter(dataset == "circle")

# Compute correlation between `x` and `y` for `dino` dataset with label `r`
circle_data %>%
  summarize(r = cor(x, y))
```

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

Again a very similar r value of slight negative correlation. The most negative of the three datasets so far.

2.6 Plot y versus x for the `circle` dataset. Does the plot look the same as either plot from the other two datasets?

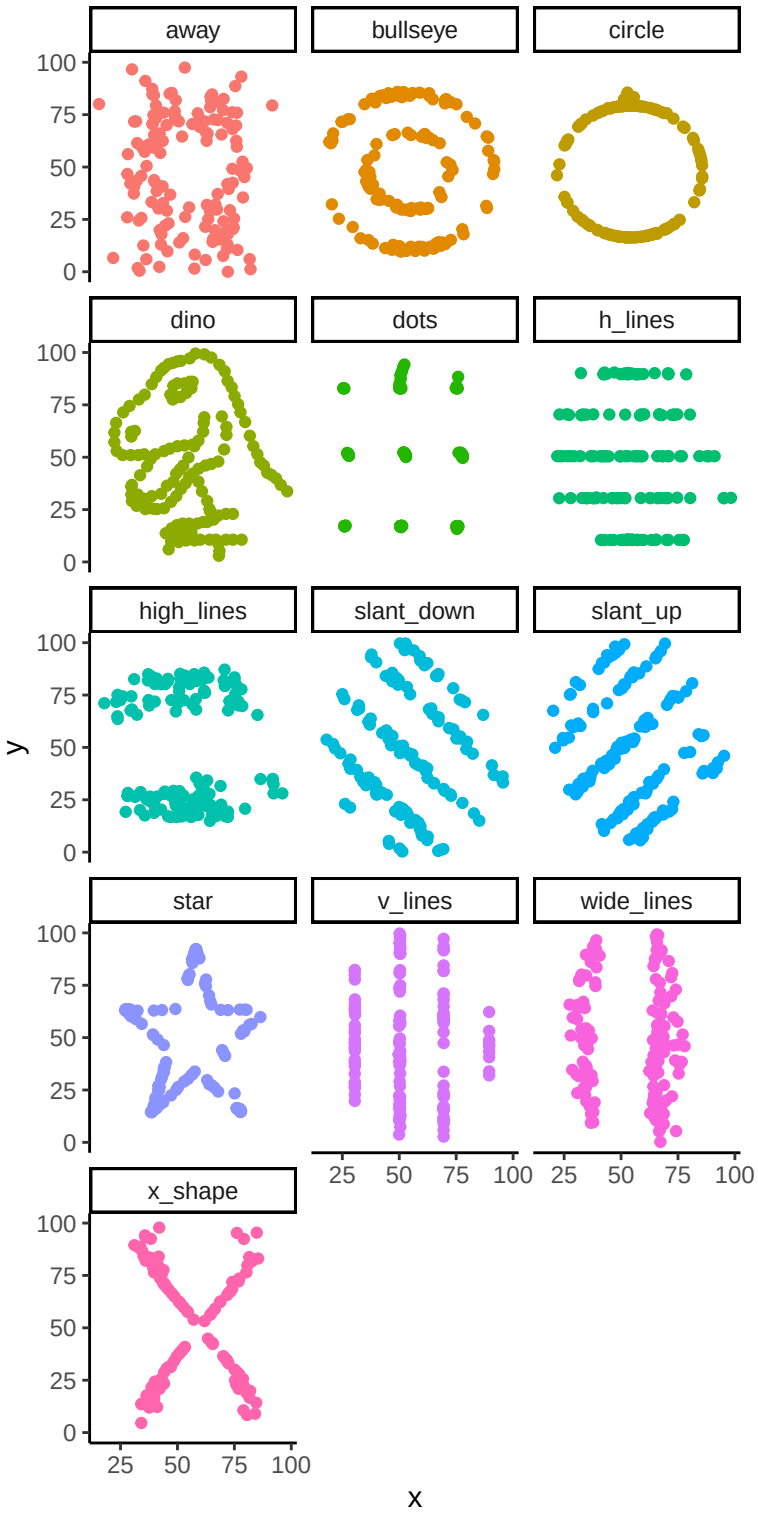
```r
ggplot(data = circle_data, mapping = aes(x = x, y = y)) +
  geom_point()
```



The plot looks different, I wonder why the r value is more negative than the previous two.

Problem 3 **Making things more efficient** The previous problem had a lot of repetition. To make things more efficient, we can plot all the datasets at one using *facets*, and we can compute all the correlations at once using the `group_by()` function. The code is provided below.

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




```
# Correlations by dataset
datasaurus_dozen %>%
  group_by(dataset) %>%
  summarize(r = cor(x, y)) %>%
  kable(booktabs = TRUE, digits = 3)
```

dataset	r
away	-0.064
bullseye	-0.069
circle	-0.068
dino	-0.064
dots	-0.060
h_lines	-0.062
high_lines	-0.069
slant_down	-0.069
slant_up	-0.069
star	-0.063
v_lines	-0.069
wide_lines	-0.067
x_shape	-0.066

3.1 What do you notice? Is the correlation coefficient an appropriate summary for any of these datasets? Why or why not?

All of the r values are very similar, telling almost nothing about the shape of the dataset. None of the correlation datasets are an appropriate summary because of of the lack of context for the data.

3.2 What do you think `ncol = 3` does in the code above? What about `digits = 3`?

"ncol = 3" likely dictates 3 columns for the graphs while "digits = 3" dictates the truncuation of the decimals.

Problem 4 **Additional Practice**

4.1 **Code chunk options** You might have noticed that this .Rmd lacks the usual global chunk options that most/all of my .Rmd files have. Let's explore how these options work when compared to the default settings. Add the following options (ONE AT A TIME) to the code chunk below (this is a modification of the code in Problem 3) and re-knit the PDF each time. Try to identify what each option is doing to the PDF output.

- `echo = FALSE`:
- `eval = FALSE`:
- `include = FALSE`:
- `comment = "##"`:
- `comment = ":)"`:
- `comment = NULL`:
- `prompt = TRUE`:
- `collapse = TRUE`:

```
# Correlations by dataset (plain R output)
datasaurus_dozen %>%
  group_by(dataset) %>%
  summarize(r = cor(x, y) %>% round(3))
```

```
# A tibble: 13 x 2
   dataset         r
   <chr>       <dbl>
 1 away       -0.064
 2 bullseye   -0.069
 3 circle     -0.068
 4 dino       -0.064
 5 dots       -0.06
 6 h_lines    -0.062
 7 high_lines -0.069
 8 slant_down -0.069
 9 slant_up   -0.069
10 star       -0.063
11 v_lines    -0.069
12 wide_lines -0.067
13 x_shape    -0.066
```

Note: There are a lot more R code chunk options, most of which we'll not use in this course. But if you're interested in the full list, check out the RMarkdown Reference Guide.

4.2 **Inline R code** Here's another fun functionality of RMarkdown: You can use inline R code chunks to place R numerical outout in your text. For instance:

The dino dataset contains 142 observations and has 3 variables. The mean x value is 54.2632732 units and the mean y value is 47.8322528 units.

You can use the `round` command to get a sensible number of digits: The mean x value is 54.3 units and the mean y value is 47.8 units.

Your turn: Use inline R coding to write a sentence about the standard deviation of x in the dino dataset reported to 1 decimal place.

References

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.