Practical 0

E Poarch

2024-05-10

Practical 0: Reproducible research in R

Objectives

- Begin to familiarise yourself with RStudio (and by extension R's tidyverse)
- Learn how to create basic data visualisations using ggplot2
- Use Rmarkdown to create literate/documented analyses
- Use git to version control your code in a github repository

Terminology

We've already thrown around a few new terms, so let's define them before we proceed.

- **R**: The open-source statistical programming language we will use throughout this course.
- **RStudio**: R's most popular integrated development environment (IDE). Rstudio offers a way to conveniently write R, run R, install R libraries, and perform version control.
- **Library**: R's equivalent of modules/packages/toolchains/toolboxes in other languages. They provide useful analysis methods, plotting functions, and development options.
- **TidyVerse**: A very popular collection of R libraries built around composability of functions, including dplyr (data manipulation) and ggplot (visualisation of data).
- **Rmarkdown**: R's literate programming library allowing creation of runnable notebooks that contain documentation and code.
- **Git**: The leading decentralised version control system we will use throughout this course.
- **GitHub**: The main website you can use to version control your project

Getting started

Unfortunately, this course will not be able to give a comprehensive introduction in R as a whole. However, there are lots of available R resources online to help your training (e.g., Harvard's Chan Schools Introductory Course or Roger Peng's Coursera courses).

R also has some of the best library documentation with a standardised requirement of full manuals and vignettes required to list libraries on the main repositories (e.g., CRAN or Bioconducter).

All of you will have learned at least 1 programming language or stastical package so the usual advice applies: experiment, look up the documentation, feel free to google wildly and often!

Submitting assignment

For each practical you will have to create a new rmarkdown notebook that contains the answers and any code required to reproducibly generate them (in **bold**)

You will then submit an email containing a link to this notebook in a git repository of your choice (this could be in a public or private git repo or Dal's institutional account, just enable access from my username `github`:fmaguire or `gitlab.cs.dal.ca`:finlaym if it is a private repository).

These are due before the next week's assignment.

Getting started

1. Install Rstudio and Git

Go to the RStudio website to download and install the software.

You may well have git already installed but if not please install it following the official documentation.

2. Open Rstudio

Rstudio is split up into 4 components by default (although this can be tweaked in the settings):

- On the bottom left is the Console Terminal (where you can run R code and any GUI-based interactions will be documented with appropriate R code)
- On the bottom right is the Files pane (lists files in your project)
- On the top left is the Editor pane (standard code editing/highlighting as appropriate with vim/emacs keybindings configurable)
- On the top right is the Environment pane (lists all currently defined variables)

Try typing `x <- 2` in the Console and hit enter, what do you get in the Environment pane?

3. Create a new project

Create a new Rproject and configure git to work with this project (connected to the appropriate repository on gitlab/github).

4. Install Required Packages

R is an open-source language, and developers contribute functionality to R via packages. In this practical we will work with three packages: `datasauRus` which contains the dataset, and `tidyverse` which is a collection of packages for doing data analysis in a "tidy" way (e.g., `dplyr`, `readr`, `tibble`, and `ggplot2`).

Load these packages by running the following in the Console.

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

If you haven't installed these packages yet and R complains, then you can install these packages by running the following command. (Note that R package names are case-sensitive)

```r
install.packages(c("tidyverse", "datasauRus"))
```

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

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 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.

YAML

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

Data

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 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.

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? (this can be hardcoded)

1846 rows, 3 columns including the 3 variables: dataset, x, y

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

```r
datasaurus_dozen %>%
  count(dataset)
```

```
## # A tibble: 13 x 2
##    dataset        n
##    <chr>      <int>
##  1 away         142
##  2 bullseye     142
##  3 circle       142
##  4 dino         142
##  5 dots         142
##  6 h_lines      142
##  7 high_lines   142
##  8 slant_down   142
##  9 slant_up     142
## 10 star         142
## 11 v_lines      142
## 12 wide_lines   142
## 13 x_shape      142
```

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

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`.

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

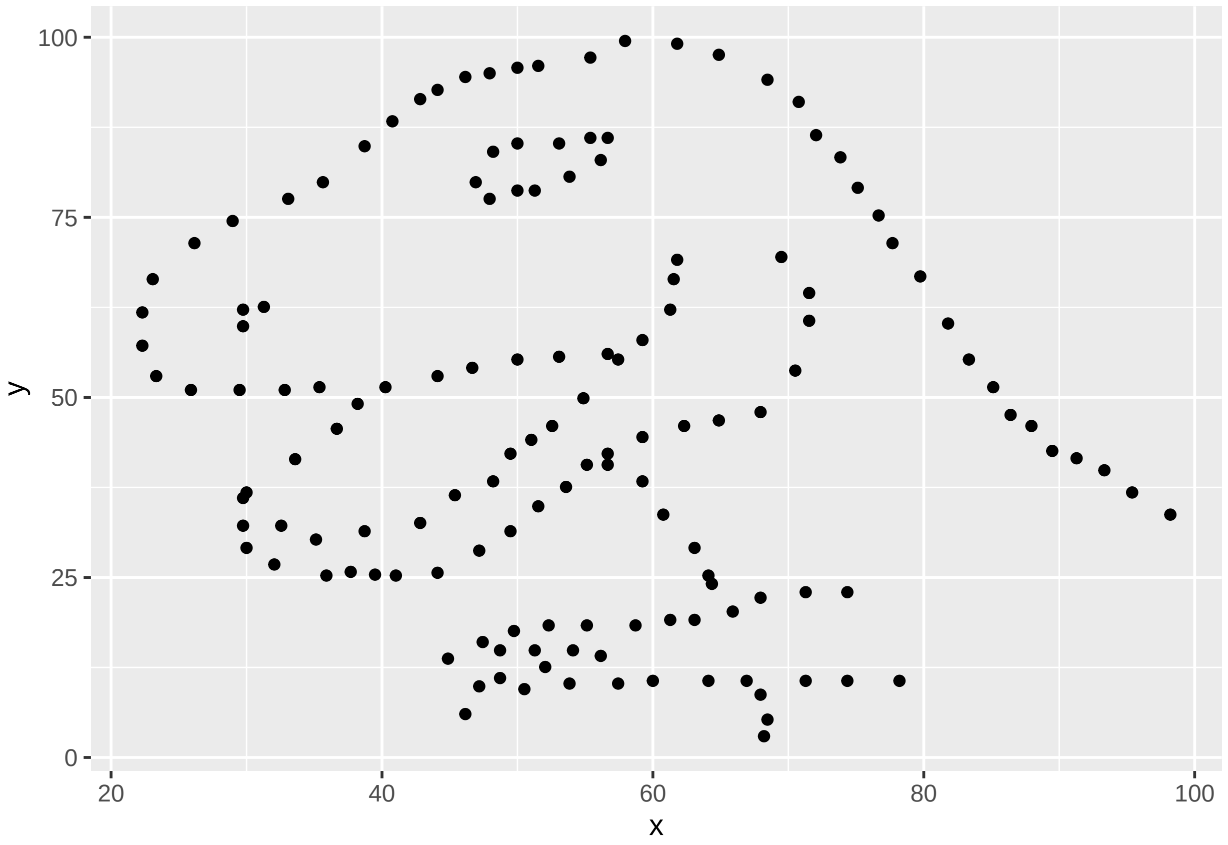
There is a lot going on here, so let's slow down and unpack it a bit.

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,m hence `geom_point`.

```r
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 as we go along. For now, follow along with the code that is provided.

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`.

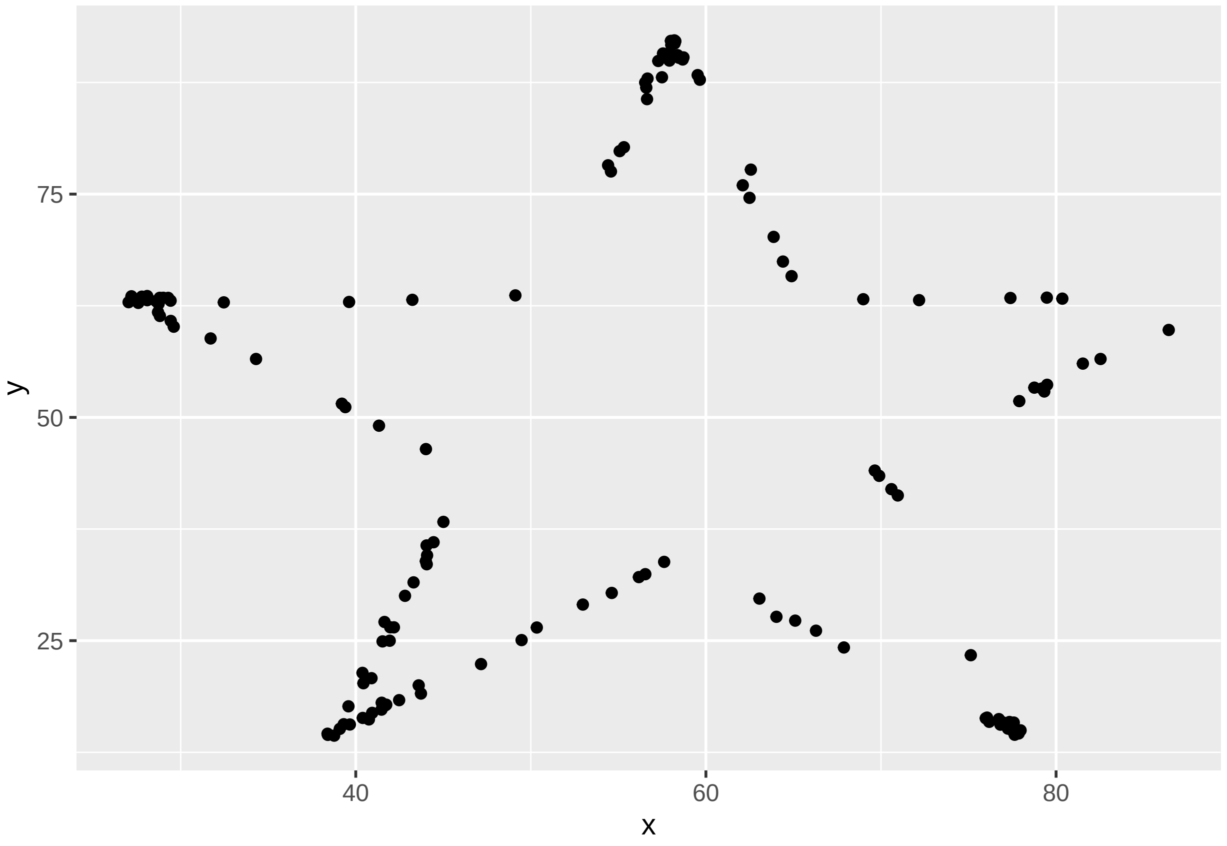
Start with `dino_data` and calculate a summary statistic that we will call `r` as the `cor`relation between `x` and `y`.

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

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`?

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

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



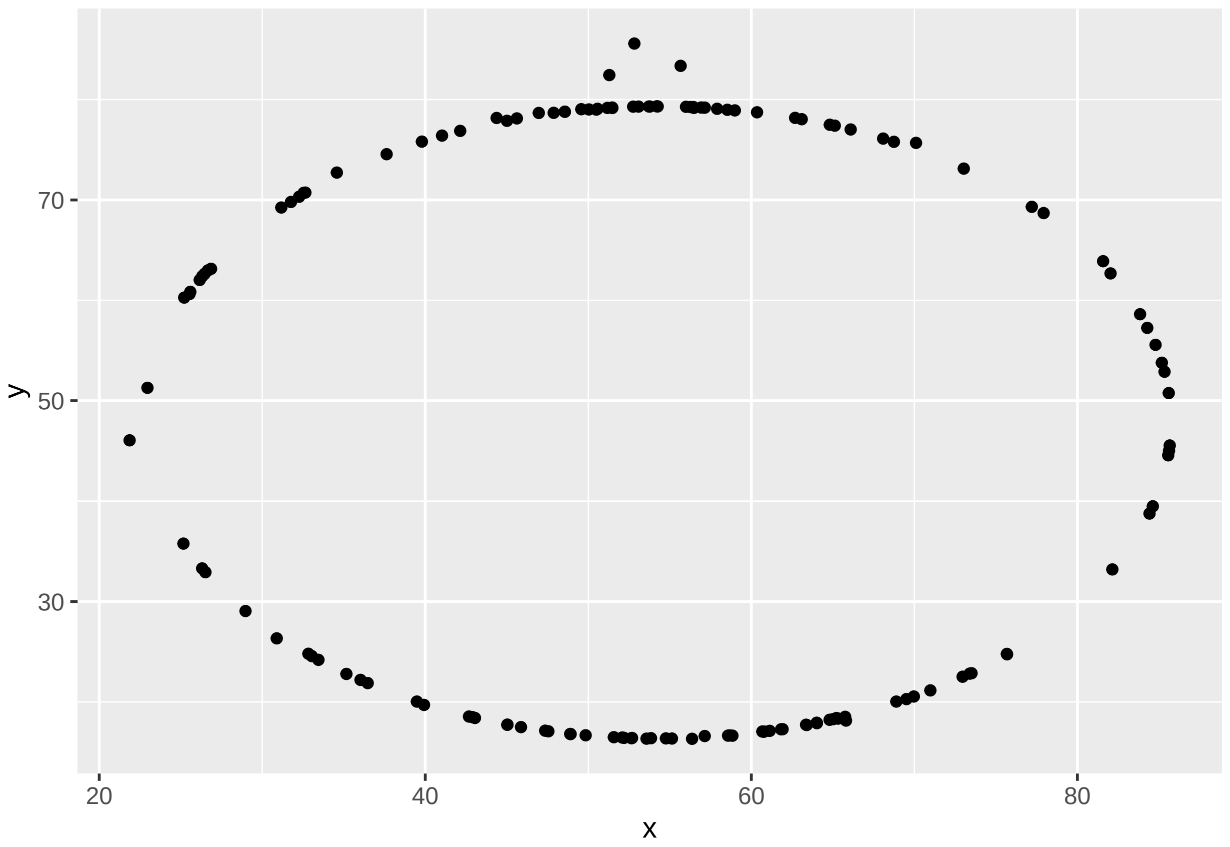
```r
star_data %>%
  summarize(r = cor(x, y))
```

Correlation coefficient is smaller in magnitude, less negative than for dino

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`?

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

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

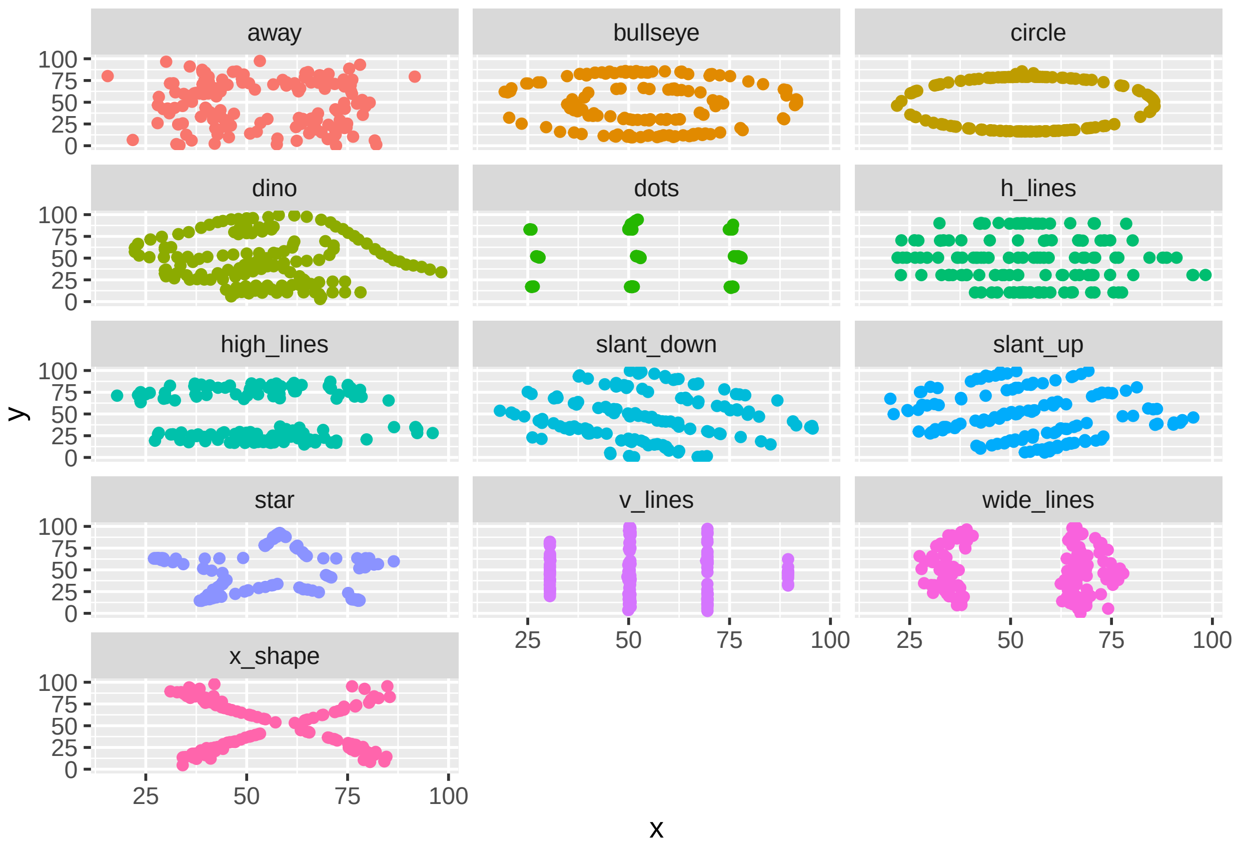


```r
circle_data %>%
  summarize(r = cor(x, y))
```

Correlation coefficient is greater in magnitude, more negative than for dino

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

```r
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 correlation coefficients.

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

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 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.

Optional

If you have time you can explore the different ways you can add styling to your rmarkdown document.

Here is a cheatsheet

and a markdown cheatsheet

This set of lab exercises have been adapted from Mine Çetinkaya-Rundel's class Introduction to Data Science and PM566