

# Stat 220 Introduction to Data Science

Deepak Bastola

2024-01-10



# Contents

<b>Course overview</b>	<b>5</b>
0.1 Learning Objectives . . . . .	5
<b>Set-up Instructions</b>	<b>9</b>
<b>1 What is R, RStudio, and RMarkdown?</b>	<b>9</b>
1.1 What is RStudio? . . . . .	9
1.2 R Studio Server . . . . .	9
1.3 R/RStudio . . . . .	10
1.4 <b>Installing R/RStudio (not needed if you are using the           maize server)</b> . . . . .	10
1.5 What is RMarkdown? . . . . .	11
1.6 Install LaTeX (for knitting R Markdown documents to PDF): . .	11
1.7 Updating R/RStudio (not needed if you are using the maize2 server) . . . . .	12
1.8 Opening a new file . . . . .	12
1.9 Running codes and knitting .Rmd files: . . . . .	12
1.10 Few More Instructions . . . . .	13
1.11 VPN . . . . .	13
<b>2 Assignments in Stat 220</b>	<b>15</b>
2.1 Do's and Don't of collaboration for individual assignments . . . .	15
2.2 Format and Content . . . . .	16
<b>3 Software in Stat 220</b>	<b>17</b>
3.1 File organization: Using maize . . . . .	17
3.2 File organization: Using your own Rstudio . . . . .	20
3.3 RStudio projects . . . . .	20
3.4 Best practices (or what not to do) . . . . .	20
3.5 Git and GitHub . . . . .	21
3.6 Slack . . . . .	21
3.7 Acknowledgements . . . . .	22

<b>4</b>	<b>GitHub Guide for Students in Stat 220</b>	<b>23</b>
4.1	Overview . . . . .	23
4.2	Getting setup with Git and GitHub . . . . .	23
4.3	Individual assignments . . . . .	24
4.4	Group work . . . . .	27
4.5	Additional resources . . . . .	28
4.6	Acknowledgements . . . . .	28
4.7	Reuse . . . . .	28
<b>5</b>	<b>R Markdown Syntax</b>	<b>29</b>
<b>6</b>	<b>Github Tutorial</b>	<b>33</b>
6.1	Tutorial 1: Creating and cloning a Repository starting from Github to RStudio . . . . .	34
6.2	Tutorial 2: Creating a new GitHub repository using <code>usethis</code> R package (RStudio to Github) (Works ONLY on local RStudio) . . . . .	35
	<b>Class Activities</b>	<b>39</b>
<b>7</b>	<b>Class Activity 1</b>	<b>39</b>
7.1	Extras (optional) . . . . .	43
7.2	Questions . . . . .	45
<b>8</b>	<b>Class Activity 2</b>	<b>47</b>
<b>9</b>	<b>Class Activity 3</b>	<b>49</b>
9.1	Question 1: data types . . . . .	49
9.2	Question 2: Subsetting and coercion . . . . .	50
9.3	Question 3: Lists . . . . .	52
<b>10</b>	<b>Class Activity 4</b>	<b>55</b>
10.1	Your turn 1 . . . . .	55

# Course overview

Greetings and welcome to Introduction to Data Science! In this course, we will delve into the computational aspects of data analysis, covering topics such as data acquisition, management, and visualization tools. Throughout this course, we will emphasize the principles of data-scientific, reproducible research and dynamic programming, utilizing the R/RStudio ecosystem.

If you have taken Stat 120, 230, or 250 at Carleton, you will find yourself well-equipped to handle the material. However, it is important to refresh your R and R-markdown skills before the start of the class. Specifically, I expect all students to be able to load a data set into R, calculate basic summary statistics, and perform basic exploratory data analysis. In the first week of class, we will delve into Git and GitHub version control, though prior exposure to these topics is not necessary.

## 0.1 Learning Objectives

- Develop research questions that can be answered by data. Import/scrape data into R and reshape it to the form necessary for analysis.
- Manipulate common types of data, including numeric, categorical (factors), text, date-times, geo-location variables in order to provide insight into your data and facilitate analysis.
- Explore data using both graphical and numeric methods to provide insight and uncover relationships/patterns.
- Utilize fundamental programming concepts such as iteration, conditional execution, and functions to streamline your code.
- Build, tune, use, and evaluate basic statistical learning models to uncover clusters and classify observations.
- Draw informed conclusions from your data and communicate your findings using both written and interactive platforms.



# Set-up Instructions





# Chapter 1

## What is R, RStudio, and RMarkdown?

R is a free and open source statistical programming language that facilitates statistical computation. There are a myriad of application that can be done in R, thanks to a huge online support community and dedicated packages. However, R has no graphical user interface and it has to be run by typing commands into a text interface.

### 1.1 What is RStudio?

RStudio provides graphical interface to R! You can think of RStudio as a graphical front-end to R that that provides extra functionality. The use of the R programming language with the RStudio interface is an essential component of this course.

### 1.2 R Studio Server

The quickest way to get started is to go to <https://maize2.mathcs.carleton.edu>, which opens an R Studio window in your web browser. Once logged in, I recommend that you do the following:

- Step 1: Create a folder for this course where you can save all of your work. In the Files window, click on New Folder.
- Step 2: Click on Tools -> Global Options -> R Markdown. Then uncheck the box that says “Show output inline...”

(It is also possible to download RStudio on your own laptop. Instructions may be found at the end of this document.)

### 1.3 R/RStudio

The use of the R programming language with the RStudio interface is an essential component of this course. You have two options for using RStudio:

- The **server version** of RStudio on the web at (<https://maize2.mathcs.carleton.edu>). The advantage of using the server version is that all of your work will be stored in the cloud, where it is automatically saved and backed up. This means that you can access your work from any computer on campus using a web browser. This server may run slow during peak days/hours. I also recommend you to download a local version of R server in your computer in case of rare outages.
- A **local version** of RStudio installed on your machine. This option is highly recommended due to the computational resources this course demands. Using this version you can only store your files in your local machine. Additionally, we can save our work on GitHub. We will learn how to use GitHub in the beginning of the course. Both R and RStudio are free and open-source. Please make sure that you have recently updated both R and RStudio.

### 1.4 Installing R/RStudio (not needed if you are using the maize server)

Download the latest version of R: <https://cran.r-project.org/> Download the free Rstudio desktop version: <https://www.rstudio.com/products/rstudio/download/>

Use the default download and install options for each. For R, download the “precompiled binary” distribution rather than the source code

#### Updating R/RStudio (not needed if you are using the maize server)

If you have used a local version of R/RStudio before and it is still installed on your machine, then you should make sure that you have the most recent versions of each program.

- To check your version of R, run the command `getRversion()` and compare your version to the newest version posted on <https://cran.r-project.org/>. If you need an update, then install the newer version using the installation directions above.
- In RStudio, check for updates with the menu option **Help > Check for updates**. Follow directions if an update is needed.

**\*\* Did it work? (A sanity check after your install/update) \*\***

Do whatever is appropriate for your operating system to launch RStudio. You should get a window similar to the screenshot you see here, but yours will be

more boring because you haven't written any code or made any figures yet!

Put your cursor in the pane labeled *Console*, which is where you interact with the live R process. Create a simple object with code like `x <- 2 * 4` (followed by enter or return). Then inspect the `x` object by typing `x` followed by enter or return. You should see the value 8 printed. If this happened, you've succeeded in installing R and RStudio!

## 1.5 What is RMarkdown?

An R Markdown file (.Rmd file) combines R commands and written analyses, which are 'knit' together into an HTML, PDF, or Microsoft Word document.

An R Markdown file contains three essential elements:

- Header: The header (top) of the file contains information like the document title, author, date and your preferred output format (pdf\_document, word\_document, or html\_document).
- Written analysis: You write up your analysis after the header and embed R code where needed. The online help below shows ways to add formatting details like bold words, lists, section labels, etc to your final pdf/word/html document. For example, adding `**` before and after a word will bold that word in your compiled document.
- R chunks: R chunks contain the R commands that you want evaluated. You embed these chunks within your written analysis and they are evaluated when you compile the document.

## 1.6 Install LaTeX (for knitting R Markdown documents to PDF):

You need a Latex compiler to create a pdf document from a R Markdown file. If you use the maize server, you don't need to install anything. If you are using a local RStudio, you should install a Latex compiler. Below are the recommended installers for Windows and Mac:

- MacTeX for Mac (3.2GB)
- MiKTeX for Windows (190MB)
- Alternatively, you can install the `tinytex` R package by running `install.packages("tinytex")` in the console.

## 1.7 Updating R/RStudio (not needed if you are using the maize2 server)

If you have used a local version of R/RStudio before and it is still installed on your machine, then you should make sure that you have the most recent versions of each program.

- To check your version of R, run the command `getRversion()` and compare your version to the newest version posted on <https://cran.r-project.org/>. If you need an update, then install the newer version using the installation directions above.
- In RStudio, check for updates with the menu option **Help > Check for updates**. Follow directions if an update is needed.

## 1.8 Opening a new file

If using Rstudio on your computer, using the **File>Open File** menu to find and open this .Rmd file.

If using Maize Rstudio from your browser:

- In the Files tab, select **Upload** and **Choose File** to find the .Rmd that you downloaded. Click *OK* to upload to your course folder/location in the maize server account.
- Click on the .Rmd file in the appropriate folder to open the file.

## 1.9 Running codes and knitting .Rmd files:

- You can run a line of code by placing your cursor in the line of code and clicking **Run Selected Line(s)**
- You can run an entire chunk by clicking the green triangle on the right side of the code chunk.
- After each small edit or code addition, **Knit** your Markdown. If you wait until the end to Knit, it will be harder to find errors in your work.
- Format output type: You can use any of `pdf_document`, `html_document` type, or `word_document` type.
- **Maize users:** You may also need to allow for “pop-up” in your web browser when knitting documents.

## 1.10 Few More Instructions

The default setting in Rstudio when you are running chunks is that the “output” (numbers, graphs) are shown **inline** within the Markdown Rmd. If you prefer to have your plots appear on the right of the console and not below the chunk, then change the settings as follows:

1. Select Tools > Global Options.
2. Click the R Markdown section and uncheck (if needed) the option Show output inline for all R Markdown documents.
3. Click OK.

Now try running R chunks in the .Rmd file to see the difference. You can recheck this box if you prefer the default setting.

## 1.11 VPN

If you plan to do any work off campus this term, you need to install Carleton’s VPN. This will allow you to access the **maize** server (if needed).

### Installing the GlobalProtect VPN

Follow the directions here to install VPN.



## Chapter 2

# Assignments in Stat 220

### 2.1 Do's and Don't of collaboration for individual assignments

- You *can* discuss homework problems with classmates but you must write up **your own** homework solutions and **do your own work in R (no sharing commands or output)**.
  - **Do not share R commands/code in any way**, including, but not limited to, sending commands via email, slack, text, or showing commands in a shared screen with the intention of showing a classmate your solution to a problem.
  - You **can** share a screen to help troubleshoot a coding problem in R.
- You *can* use the following resources to complete your homework:
  - Carleton faculty (myself, other math or statistics faculty, etc)
  - discussions with classmates (see above) or knowledgeable friends
  - Carleton resources like stats lab assistants
  - student solutions provided in the back of your student textbook or in the student solution manual.
- You *cannot* use any resources other than the ones listed above to complete assignments (homework, reports, etc) for this class. (e.g. you cannot use a friend's old assignments or reports, answers found on the internet, textbook (instructor) solutions manual, etc.)

#### 2.1.1 Examples that violate the academic integrity policy

- sending your .Rmd homework file to another person in the class
- receiving an .Rmd homework file from another person
- sharing a screen and copying code, verbatim, from another person
- sending/receiving R commands

- neglecting to acknowledge classmates with whom you worked with on an assignment

## 2.2 Format and Content

Submit via GitHub (for most assignments) an organized and correctly ordered assignment.

- Content: Good data scientists need to do more than just write code; they should be able to interpret and explain their analyzes.
  - Provide a **written answer** first, followed by any required R code and output.
  - Use **complete sentences** when answering any problem that requires an explanation or overall problem summary.
- When including code:
  - Be sure to show the natural sequence of work needed to answer the problem.
  - Include brief comments explain your code steps.
  - Do not include typos or unnecessary commands/output.
  - Always include code output.
- At the top of each individual assignment **include the names of classmates that you worked with** on all or part of the assignment (but each person must write up their assignment on their own)

---

**Disability Accommodations:** Carleton College is committed to providing equitable access to learning opportunities for all students. The Disability Services office (Henry House, 107 Union Street) is the campus office that collaborates with students who have disabilities to provide and/or arrange reasonable accommodations. If you have, or think you may have, a disability (e.g., mental health, attentional, learning, autism spectrum disorders, chronic health, traumatic brain injury and concussions, vision, hearing, mobility, or speech impairments), please contact [disability@carleton.edu](mailto:disability@carleton.edu) or call Sam Thayer ('10), Accessibility Specialist (x4464) or Chris Dallager, Director of Disability Services (x5250) to arrange a confidential discussion regarding equitable access and reasonable accommodations.

**Academic Honesty:** All work that you turn in under your name must follow Carleton's academic integrity policy. The use of textbook solution manuals (physical or online solutions), homework, reports or exams done by past students are not allowed. Look at the College's Writing Across the Curriculum website for additional guidance on plagiarism and how to avoid plagiarism in their writing.



## Chapter 3

# Software in Stat 220

You will work with many .Rmd Markdown files in this course. These include class activities, homework template, project helper files etc. To stay organized, I *strongly* suggest you create a **stat220** folder that contains the following sub-folders:

- **stat220** folder
  - **Assignments:** This folder will contain subfolders for each assignment. Each assignment subfolder (e.g. homework1, homework2, ...) will be a Github connected RStudio project that you will create **once an assignment is posted**.
  - **Content:** This folder should be used to save any non-assignment files (e.g. slides, examples) for this class. You will create this subfolder by creating an RStudio project (see step 5 below).

To get started with this organization, follow the steps below.

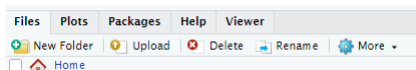
---

### 3.1 File organization: Using maize

The server (online) version of Rstudio is run from a unix server. You can navigate this file system using unix commands, but I assume that most or all of you will just use Rstudio to access your files on this server.

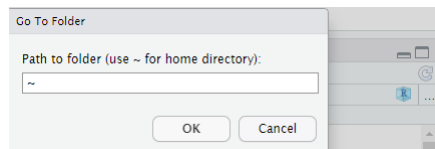
1. In Rstudio, click the **Files** *tab* in the lower right-hand window.

Note: this is **not** the same as the **File** *menu* option.

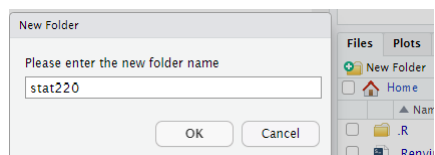


2. Verify that you are in your **HOME** folder (should simply say

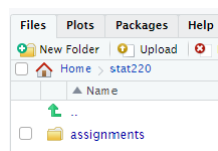
Home right under the New Folder button). To navigate to your Home folder (if somehow you are not in it), click the ... button (far right side of the **Files** tab) and enter a ~ (tilde) symbol



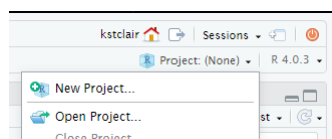
- **3.** Click the **New Folder** button and name the folder **stat220**.



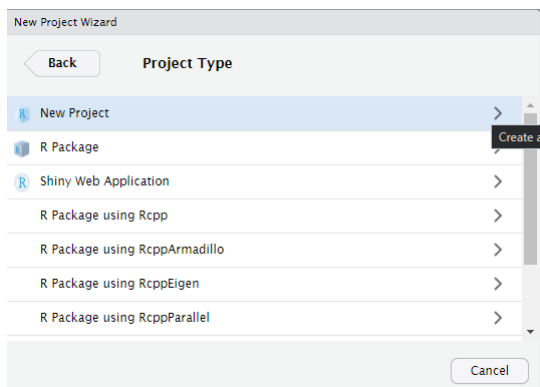
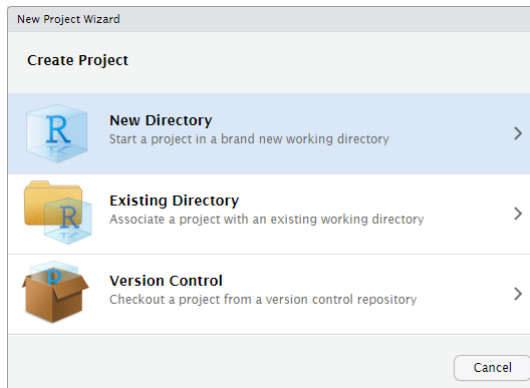
- **4.** Click on this newly created (empty) **stat220** folder. Within the folder create another **New Folder** and name it **assignments**.



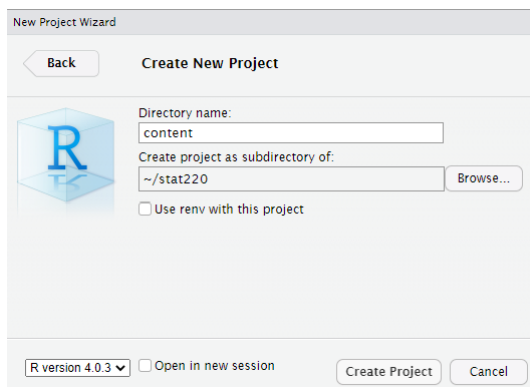
- **5.** Within the **stat220** folder, create an **RStudio project** called **content** with the following steps:
  - **a.** Click the **Project** button in the upper righthand corner of your RStudio window and select **New Project....**



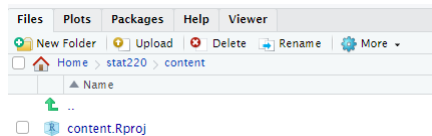
- **b.** Select **New Directory** and then **New Project**



- c. Enter **content** as the **Directory name** and use the **Browse** button to find your **stat220** folder. Then click **Create Project**.



- d. You should now have a new folder called **content** in your **stat220** folder and this folder will contain an RStudio project **.Rproj**. Feel free to add subfolders to this **content** folder (e.g. slides, examples, etc).



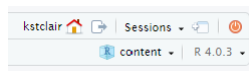
**Warning:** Do not create an RStudio project in the main `stat220` folder because it is not good practice to have RStudio projects in subfolders of another project (e.g. a project within a project is not recommended).

## 3.2 File organization: Using your own Rstudio

Create a folder called **stat220** somewhere on your computer. Within this folder create an **assignments** subfolder. Then complete **step 5** from above to create a **content** RStudio project folder.

## 3.3 RStudio projects

Once you've created a project, your R session should be running within that project folder. You can check which project you are in by checking the project name in the upper righthand part of your RStudio window. Here we see the **content** project is open:



Running R from an RStudio project sets your **working directory** to the project folder:

```
> getwd()
[1] "/Accounts/kstclair/stat220/content"
```

This allows for easy file path access to all files related to this project.

To **start** a project, click on the `.Rproj` file or use the **Open Project...** option shown in step 5 above.

## 3.4 Best practices (or what not to do)

- Never save files to a lab computer hard drive (e.g. desktop, downloads, etc). They will be erased when you log off.

- Do not use gmail as a file storage system! Avoid emailing yourself files that you created (and saved) on a lab computer. Eventually you will lose work this way.
- Avoid using online versions of google drive and dropbox. Similar to gmail, downloading, editing a doc, then uploading it back to drive/dropbox is another great way to lose work.
- Avoid this and this.

## 3.5 Git and GitHub

Git is version control software that you install locally on your computer. Git is already installed on the maize RStudio server.

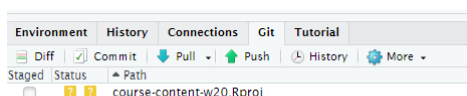
Github is a cloud-based service for hosting git projects. It allows multiple users to share and contribute to projects and it is how you will be submitting homework assignments and projects for this class. More information about Git and Github can also be found in Getting setup with Git and GitHub and Git and Github.

If you are using a local install of R/RStudio, then you will need to install Git.

### Installing Git

Directions for both Windows & Mac here: <http://happygitwithr.com/install-git.html>.

1. If you are using **maize**, then there is nothing you need to install.
2. Windows users should follow Option 1 in 6.2.
3. Mac users can follow Option 1 in 6.3 if comfortable, otherwise follow Option 2
4. Linux users can follow 6.4.



## 3.6 Slack

We will use Slack for all course communication. Sign up for our course Slack team here! You will need to create an account with a username, and log in to read and post. You can download a standalone Slack application to your Mac, Windows, Linux and/or Android/iOS device. You can control whether you receive notifications on new posts by going to Preferences, as well as decide which ‘channels’ to subscribe to. A ‘channel’ is a discussion thread, which is used to organize communications into topics. You can learn more about Slack features here.

Several channels have been set up for specific parts of the course. Feel free to ask questions anytime. You can browse the available channels in our team by clicking on “Channels” on the left-hand panel.

## 3.7 Acknowledgements

This installation guide is based on the guide from Adam Loy and Katie St. Clair.

## Chapter 4

# GitHub Guide for Students in Stat 220

### 4.1 Overview

If you are using the maize RStudio server, then you can connect to GitHub without any extra software downloads. If you are using RStudio on your computer, then you will need to download Git software (as directed in Software in Stat 220) to use GitHub connected projects. You will use GitHub to submit homework and collaborate on projects.

### 4.2 Getting setup with Git and GitHub

If you are **not** working on the maize RStudio server, then make sure that you have installed all of the software mentioned in Software in Stat 220. In addition, you should install the `usethis` and `gitcreds` R packages.

Everyone needs to connect Git and GitHub by doing the following:

1. Register for account on GitHub (<https://github.com/>). I recommend using a username that incorporates your name (e.g., dbastola) and Carleton email address for your Github account.
2. If you haven't done so already, accept the invite to the class organization DataScienceWinter24. This organization is where all course homework files and project repositories will live.
3. Setup options in Git by running the following code chunk in your console:

```
#install.packages("usethis") # uncomment to install  
usethis::use_git_config(user.name = "Jane Doe", user.email = "jane@example.org")
```

changing the first two arguments to your own name and email (this should be the email associated with your GitHub account).

4. In order to push changes to github (i.e. to track changes and submit homework), you will need to prove that you have permission to change a Github repo. This is done with a personal access token (PAT). Note that you will need to install the packages `usethis` and `gitcreds` to do this.

```
usethis::create_github_token()
```

Call ``gitcreds::gitcreds_set()`` to register this token in the local Git credentials. It is also a great idea to store this token in any password-management software. Opening URL `'https://github.com/settings/tokens/new?scopes=repo,user,gist,workf'`

“Generate token” and store your tokens somewhere safe in your local computer as you will need this again in the future. You can additionally add PAT to your `.Renv` file as well. Copy it and paste it into your `.Renv` file as system variable `GITHUB_PAT` using

```
usethis::edit_r_environ()
```

Add to the file and save. You can also set the PAT token in R using the following.

```
#install.packages("gitcreds") # uncomment to install
gitcreds::gitcreds_set()
```

You can check that you’ve stored a credential with `gitcreds_get()`:

```
gitcreds::gitcreds_get()
```

You should get something like this:

```
...
#> <gitcreds>
#> protocol: https
#> host      : github.com
#> username: PersonalAccessToken
#> password: <-- hidden -->
...
```

**Treat your PAT token like a password!** For details, follow the step in Section 9.1 on this page to do this: <https://happygitwithr.com/https-pat.html>.

### 4.3 Individual assignments

If you followed the suggestions in the File organization in RStudio page, then you should already have an assignments folder on your computer or maize account.

Each new assignment/project will be posted as a repository on GitHub and



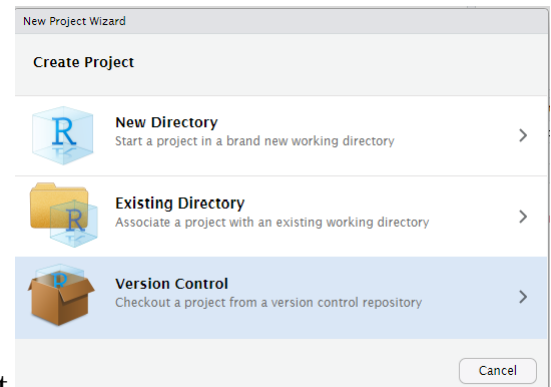
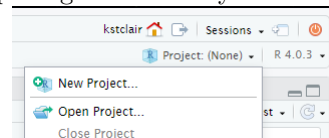
added directly to your account (within the Stat220 organization). This repository will contain assignment details (README, .Rmd).

### 4.3.1 Creating an individual assignment repo and project

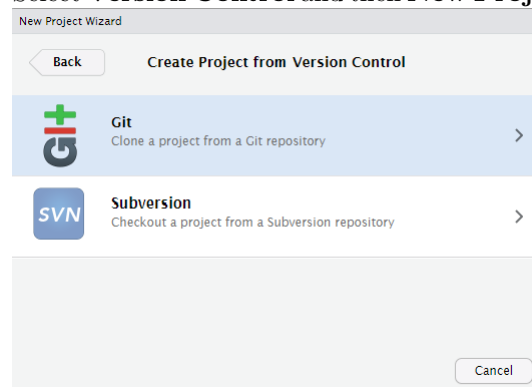
1. Go to our course GitHub organization page (DataScienceWinter24) and find your homework repo, such as `hw1-username` (where your username is attached).
2. Enter the online assignment repository on GitHub. Click the green “Code” button. Most of you should just use the default setting which is to “clone” (copy) using HTTPS. Click the clipboard to the right of the URL to copy the repo location.
3. Now open up RStudio and create a project as follows:

- Click the **Project** button in the upper right corner of your RStudio

window and select **New Project...**

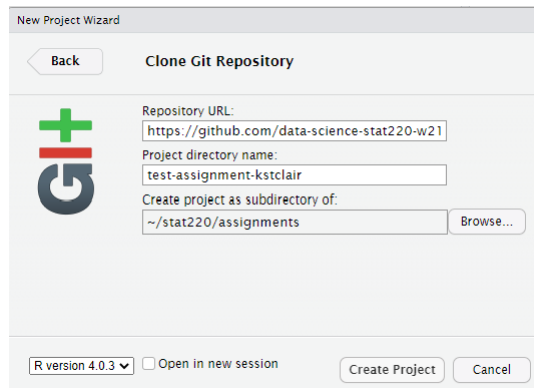


- Select **Version Control** and then **New Project**



- Paste the link you just copied into the Repository URL box. Leave

the Project directory name blank (or keep the auto-filled name). Use the **Browse** button to find your **assignments** folder, then click **Create Project**



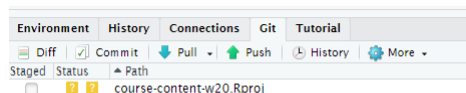
### 4.3.2 Working on your assignment

An RStudio project should now open, which will allow you to start working on your homework assignment. You should see the project assignment name in the top right side of RStudio. You will probably see a blank console screen when you open a new project. Look in the **Files** tab for your homework .Rmd file. Click on whatever file you want to edit (probably the .Rmd file) and edit away. Make sure that your current assignment's project is the one open and showing in the upper right project name. To **open** a project, click on the .Rproj file or use the **Open Project...** option available in the upper right project link.

#### 4.3.2.1 Commits

After you make changes to the homework assignment, commit them. What are commits you ask? Commits are essentially taking a snapshot of your projects. Commits save this snapshot to your local version of Git (located on your hard drive or the maize server). For example, if I make changes to a code so that it prints "Hello world", and then commit them with an informative message, I can look at the history of my commits and view the code that I wrote at that time. If I made some more changes to the function that resulted in an error, I could go back to the commit where the code was originally working. This prevents you from creating several versions of your homework (homework-v1, homework-v2, ...) or from trying to remember what your code originally looked like.

You can make commits in the Git tab in RStudio.



Click the **Commit** button in the Git tab. Check the boxes of the files that you want to commit, enter your commit message (briefly state what changes have been made), then hit **Commit**. You can read how to do this in RStudio in more detail here: <http://r-pkgs.had.co.nz/git.html#git-commit>.

Two things about committing.

- You should **commit somewhat frequently**. At minimum, if you're doing a homework assignment, you should make a commit each time that you've finished a question.
- Leave **informative commit messages**. "Added stuff" will not help you if you're looking at your commit history in a year. A message like "Added initial version of hello-world function" will be more useful.

#### 4.3.2.2 Pushing changes to Github

At some point you'll want to get the updated version of the assignment back onto GitHub, either so that we can help you with your code or so that it can be graded. You will also want to push work frequently when you have a shared GitHub repo for project collaborations (i.e. more than one person is working on a project and code). If you are ready to push, you can again click on the "Up" **Push** arrow in the Git tab or in the Commit pop-up window or in the Git tab (shown above).

To "turn in" an assignment, all you need to do is push all your relevant files to Github by the deadline.

## 4.4 Group work

Collaborative Github assignments are pretty similar to individual assignments.

### 4.4.1 Creating a group/partner assignment repo and project

Go to our course GitHub organization page(DataScienceWinter24) and find the repo for your group, for example if your group name is "team01" the you might find the `mp1-team01` repo. Clone this repo to your computer/maize account using the same steps done for an individual assignment (see steps 2-3).

#### 4.4.1.1 Working with collaborative repos

For group homework, I suggest that only the *recorder* edit the group-homework-x.Rmd file to avoid merge conflicts! Other group members can create a new Markdown doc to run and save commands. Only the recorder needs to **push** changes (answers) to the Github repo and all others can then **pull** these changes (i.e. the final answers) after the HW is submitted.

When you are working together on a Github project, you should commit and push your modifications frequently. You will also need to frequently **pull** updates from Github down to your local version of RStudio. These updates are changes that your teammates have made since your last pull. To pull in changes, click the “Down” **Pull** arrow in the Git tab (shown above).

If you get an error about conflict after pulling or pushing, don’t freak out! This can happen if you edit a file (usually an .Rmd or .R file) in a location that was also changed by a teammate. When this happens you should attempt to fix the **merge conflict**. Take a look at this resource site and try to fix the merge conflict in Rstudio.

## 4.5 Additional resources

- Happy Git and GitHub for the useR
- Rstudio, Git and GitHub
- Interactive learning guide for Git
- GitHub Guides
- Git setup for Windows (video)
- Git setup for Mac (video)
- How to clone, edit, and push homework assignments with GitHub Classroom (video)

## 4.6 Acknowledgements

Most of this content in this guide was taken from <https://github.com/jfiksels/github-classroom-for-students>, edited for our classroom use by Katie St. Clair.

## 4.7 Reuse

This guide is licensed under the CC BY-NC 3.0 Creative Commons License.

## Chapter 5

# R Markdown Syntax

Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

### 5.0.1 Lists in R Markdown:

You can use asterisk mark to provide emphasis, such as *\*italics\** or **\*\*bold\*\***. You can create lists with a dash:

```
- Item 1
- Item 2
- Item 3
  + Subitem 1
* Item 4
```

to produce

- Item 1
- Item 2
- Item 3
  - Subitem 1
- Item 4

You can embed Latex equations in-line,  $\frac{1}{n} \sum_{i=1}^n x_i$  to produce  $\frac{1}{n} \sum_{i=1}^n x_i$  or in a new line as  $\text{Var}(X) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$  to produce

$$\text{Var}(X) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

### 5.0.2 Embed an R code chunk:

Use the following

```
```r
Use back ticks to
create a block of code
```
```

to produce:

```
Use back ticks to
create a block of code
```

You can also evaluate and display the results of R code. Each task can be accomplished in a suitably labeled chunk like the following:

```
summary(cars)
```

| speed         | dist           |
|---------------|----------------|
| Min. : 4.0    | Min. : 2.00    |
| 1st Qu.: 12.0 | 1st Qu.: 26.00 |
| Median : 15.0 | Median : 36.00 |
| Mean : 15.4   | Mean : 42.98   |
| 3rd Qu.: 19.0 | 3rd Qu.: 56.00 |
| Max. : 25.0   | Max. : 120.00  |

```
fit <- lm(dist ~ speed, data = cars)
fit
```

Call:

```
lm(formula = dist ~ speed, data = cars)
```

Coefficients:

|             |       |
|-------------|-------|
| (Intercept) | speed |
| -17.579     | 3.932 |

### 5.0.3 Including Plots:

You can also embed plots. See Figure 5.1 for example:

```
par(mar = c(0, 1, 0, 1))
pie(
  c(280, 60, 20),
  c('Sky', 'Sunny side of pyramid', 'Shady side of pyramid'),
  col = c('#0292D8', '#F7EA39', '#C4B632'),
  init.angle = -50, border = NA
)
```

(Credit: Yihui Xie)

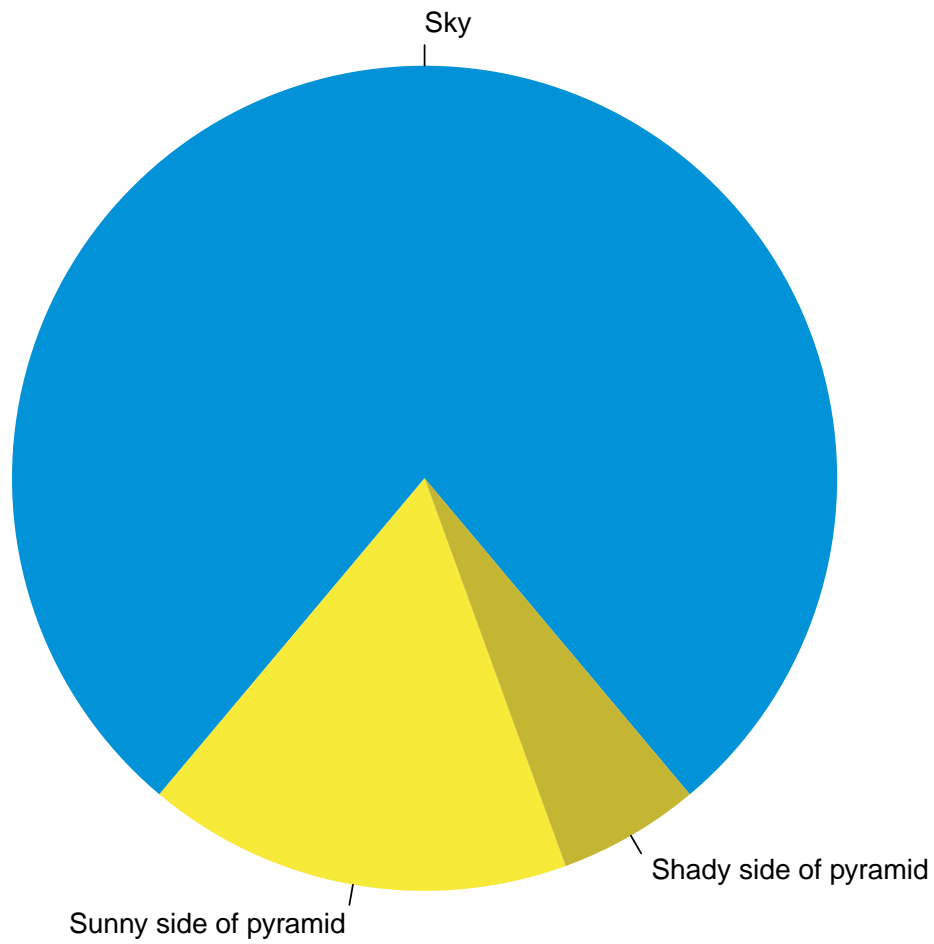


Figure 5.1: A fancy pie chart.

### 5.0.4 Read in data files:

```
simple_data <- read.csv("https://deepbas.io/data/simple-1.dat", )
summary(simple_data)
```

```

      initials      state      age
Length:3      Length:3      Min.   :45.0
Class :character Class :character 1st Qu.:47.5
Mode  :character Mode  :character Median :50.0
                                   Mean  :52.0
                                   3rd Qu.:55.5
                                   Max.   :61.0

      time
Length:3
Class :character
Mode  :character
```

```
knitr::kable(simple_data)
```

| initials | state | age | time |
|----------|-------|-----|------|
| vib      | MA    | 61  | 6:01 |
| adc      | TX    | 45  | 5:45 |
| kme      | CT    | 50  | 4:19 |

### 5.0.5 Hide the code:

If we enter the `echo = FALSE` option in the R chunk (see the .Rmd file). This prevents the R code from being printed to your document; you just see the results.

| initials | state | age | time |
|----------|-------|-----|------|
| vib      | MA    | 61  | 6:01 |
| adc      | TX    | 45  | 5:45 |
| kme      | CT    | 50  | 4:19 |



## Chapter 6

# Github Tutorial

```
# load the required libraries
library(credentials) # to help with PAT access
library(gitcreds)
library(usethis)

# STEPS INVOLVED TO ESTABLISH GIT CREDENTIALS / PAT

# Step 1

# usethis::use_git_config(user.name = "deepbas", user.email = "deepbas99@gmail.com")

# Step 2

# usethis::create_github_token()

# Step 3

# if this is the second/subsequent iteration start from here

# gitcreds::gitcreds_set()

# Verify

# gitcreds::gitcreds_get()
```

In this worksheet, you will practice creating a GitHub repository using the `usethis::use_github()` function and cloning it back to your local machine using RStudio's menu options.

## 6.1 Tutorial 1: Creating and cloning a Repository starting from Github to RStudio

1. Visit the GitHub website at <https://github.com> and sign in using your GitHub account. If you don't have an account yet, you can create one for free.
2. Once logged in, click on the “+” icon in the top right corner of the web-page, then click on “New repository”.
3. Enter a name for your new repository in the “Repository name” field. You may also provide an optional description.
4. Choose the visibility of your repository by selecting either “Public” or “Private”. Public repositories are visible to anyone, while private repositories are only visible to you and any collaborators you invite.
5. (Optional) Check the box to initialize the repository with a README file.
6. Click on the “Create repository” button to create your new repository.

This will create a new GitHub repository on your Github account. Follow further to clone the repository to your local folder using RStudio.

1. Go to your GitHub repository webpage and click on the green “Code” button. This will display a dropdown menu with a URL for your repository. Click on the clipboard icon to copy the URL to your clipboard.
2. Open RStudio, and from the “File” menu, select “New Project”.
3. In the “New Project” dialog, choose “Version Control”.
4. Select “Git” as the version control system.
5. In the “Repository URL” field, paste the URL that you copied from your GitHub repository webpage.
6. Choose a local directory where you want to clone the repository by clicking on the “Browse” button and navigating to the desired folder on your computer.
7. Click on “Create Project” to clone the GitHub repository to your local computer.

## 6.2 Tutorial 2: Creating a new GitHub repository using usethis R package (RStudio to Github) (Works ONLY on local RStudio)

### 6.2.1 Prerequisites

1. Install the usethis package if you haven't already: `install.packages("usethis")`
2. Make sure you have a GitHub account, and you are logged in.
3. Configure Git with your name and email address if you haven't already. Run the following commands in the R console, replacing "Your Name" and "youremail@example.com" with your information:

```
usethis::use_git_config(user.name = "Your Name", user.email = "youremail@example.com")
```

4. Create a new R project in RStudio by clicking on "File" > "New Project" > "New Directory" > "New Project." Give your project a name and choose a location on your computer to save it. Click "Create Project."
5. Make a new file or copy and paste a .Rmd file that you want to have in your repo and save it to your requirement.
6. In the R console, load the usethis package:

```
library(usethis)
```

7. Initialize a Git repository for your project by running:

```
usethis::use_git()
```

8. Now, let's create a new GitHub repository using the `usethis::use_github()` function. Run the following command:

```
usethis::use_github()
```

9. Follow the instructions in the R console, and your GitHub repository will be created. Note the repository URL, as you will need it in the next activity.



# Class Activities



## Chapter 7

# Class Activity 1

The R package `babynames` provides data about the popularity of individual baby names from the US Social Security Administration. Data includes all names used at least 5 times in a year beginning in 1880.

```
#install.packages("babynames") # uncomment to install  
library(babynames)
```

Below is the list for first few cases of baby names.

```
head(babynames)
```

```
# A tibble: 6 x 5  
  year sex  name      n  prop  
  <dbl> <chr> <chr>   <int> <dbl>  
1  1880 F    Mary    7065 0.0724  
2  1880 F    Anna    2604 0.0267  
3  1880 F    Emma    2003 0.0205  
4  1880 F  Elizabeth 1939 0.0199  
5  1880 F   Minnie   1746 0.0179  
6  1880 F  Margaret 1578 0.0162
```

1. How many cases and variables are in the dataset `babynames`?

**Answer:**

```
dim(babynames)
```

```
[1] 1924665      5
```

There are 1924665 cases and 5 variables in the dataset `babynames`.

Let's use the package `tidyverse` to do some exploratory data analysis.

```
#install.packages("tidyverse") # uncomment to install
library(tidyverse)
babynames %>% filter(name=='Aimee')
```

```
# A tibble: 150 x 5
  year sex  name      n      prop
  <dbl> <chr> <chr> <int>   <dbl>
1  1880 F    Aimee     13 0.000133
2  1881 F    Aimee     11 0.000111
3  1882 F    Aimee     13 0.000112
4  1883 F    Aimee     11 0.0000916
5  1884 F    Aimee     15 0.000109
6  1885 F    Aimee     17 0.000120
7  1886 F    Aimee     17 0.000111
8  1887 F    Aimee     18 0.000116
9  1888 F    Aimee     12 0.0000633
10 1889 F    Aimee     16 0.0000846
# i 140 more rows
```

```
filtered_names <- babynames %>% filter(name=='Aimee')
```

```
#install.packages("ggplot2") # uncomment to install
library(ggplot2)
```

```
ggplot(data=filtered_names, aes(x=year, y=prop)) +
  geom_line(aes(colour=sex)) +
  xlab('Year') +
  ylab('Prop. of Babies Named Aimee')
```

2. What do you see in the Figure 1? Explain in a few sentences.

Click for answer

**Answer:**

In Figure 1, we can see the proportion of babies named Aimee by year for both males and females. We notice that the name Aimee has been more popular among females than males throughout the years. There is a peak in popularity around the 1970s for female babies, and then the popularity declines.

3. Repeat question 2 to infer how does the proportion of babies with your first name trend over time. Examine the generated plot and describe the trend of your name's popularity over time. Consider the following points:

Has the popularity of your name increased, decreased, or remained stable over the years? Is there a noticeable difference in popularity between sexes? Are there any interesting patterns or trends, such as sudden increases or decreases in popularity?



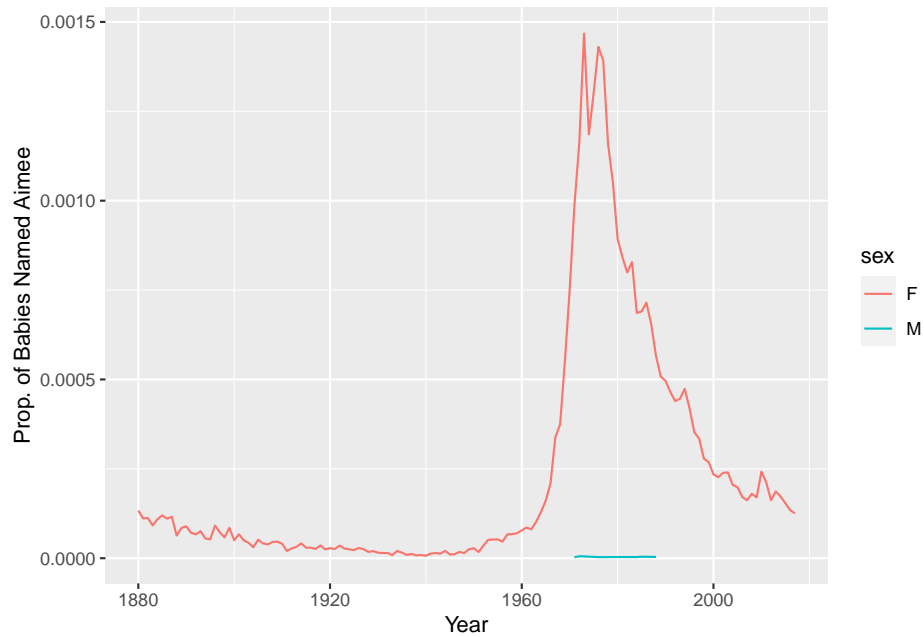


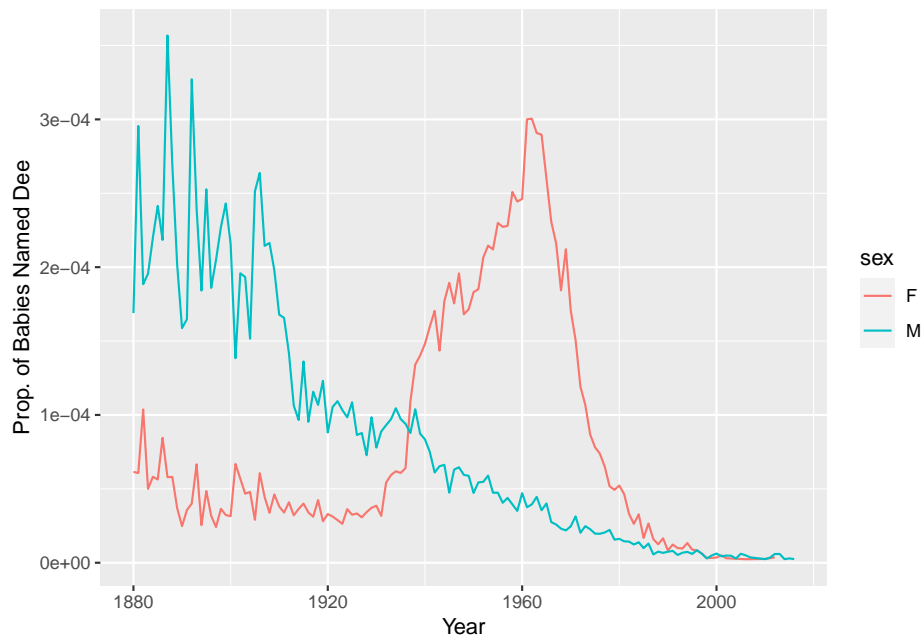
Figure 7.1: A trend chart

**Answer:** Answers will vary.

```
# Replace 'YourName' with your first name
your_name <- "Dee"

your_name_data <- babynames %>% filter(name == your_name)

ggplot(data=your_name_data, aes(x=year, y=prop)) +
  geom_line(aes(colour=sex)) +
  xlab('Year') +
  ylab(paste('Prop. of Babies Named', your_name))
```



4 (Optional). Compare the popularity of your first name with a randomly chosen name from the dataset. Examine the generated plot and compare the popularity of your first name with the randomly chosen name. Consider the following points:

Are there differences in popularity trends between the two names? Is one name consistently more popular than the other, or do their popularity levels change over time? Are there any interesting patterns or trends in the data, such as periods of rapid increase or decrease in popularity?

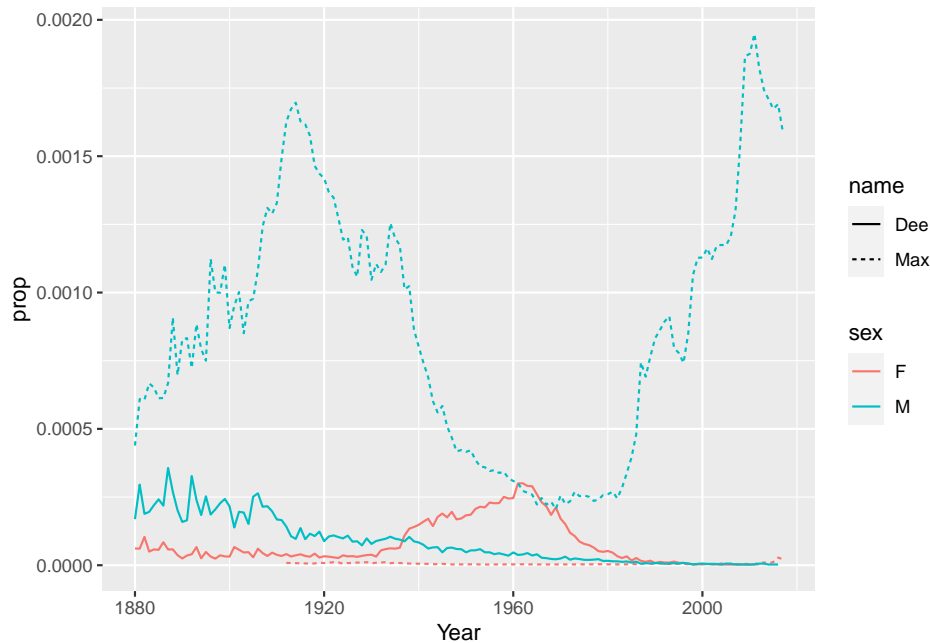
**Answer** Answers will vary

```
# Replace 'YourName' with your first name
your_name_data <- babynames %>% filter(name == 'Dee')

# Replace 'RandomName' with a randomly chosen name from the dataset
random_name_data <- babynames %>% filter(name == 'Max')

# Combine the two datasets
combined_data <- bind_rows(your_name_data, random_name_data)

# Plot the data
ggplot(data=combined_data, aes(x=year, y=prop)) +
  geom_line(aes(colour=sex, linetype=name)) +
  xlab('Year')
```



## 7.1 Extras (optional)

### 7.1.1 Part 1: Setting Working Directory and Loading Data

1. Set your working directory to a folder on your computer where you would like to save your R scripts and data files.

```
# Replace 'your_directory_path' with the path to your desired folder
# setwd("your_directory_path")
```

2. Load the mtcars dataset which comes preloaded with R. This dataset consists of various car features and their corresponding miles per gallon (mpg) values.

```
data(mtcars)
head(mtcars)
```

|                   | mpg  | cyl | disp | hp  | drat | wt    | qsec  | vs | am |
|-------------------|------|-----|------|-----|------|-------|-------|----|----|
| Mazda RX4         | 21.0 | 6   | 160  | 110 | 3.90 | 2.620 | 16.46 | 0  | 1  |
| Mazda RX4 Wag     | 21.0 | 6   | 160  | 110 | 3.90 | 2.875 | 17.02 | 0  | 1  |
| Datsun 710        | 22.8 | 4   | 108  | 93  | 3.85 | 2.320 | 18.61 | 1  | 1  |
| Hornet 4 Drive    | 21.4 | 6   | 258  | 110 | 3.08 | 3.215 | 19.44 | 1  | 0  |
| Hornet Sportabout | 18.7 | 8   | 360  | 175 | 3.15 | 3.440 | 17.02 | 0  | 0  |
| Valiant           | 18.1 | 6   | 225  | 105 | 2.76 | 3.460 | 20.22 | 1  | 0  |

gear carb

|                   |   |   |
|-------------------|---|---|
| Mazda RX4         | 4 | 4 |
| Mazda RX4 Wag     | 4 | 4 |
| Datsun 710        | 4 | 1 |
| Hornet 4 Drive    | 3 | 1 |
| Hornet Sportabout | 3 | 2 |
| Valiant           | 3 | 1 |

### 7.1.2 Part 2: Downloading Packages

1. Install the “tidyverse” package, which is a collection of useful R packages for data manipulation, exploration, and visualization.

```
# Uncomment the line below to install the package  
# install.packages("tidyverse")
```

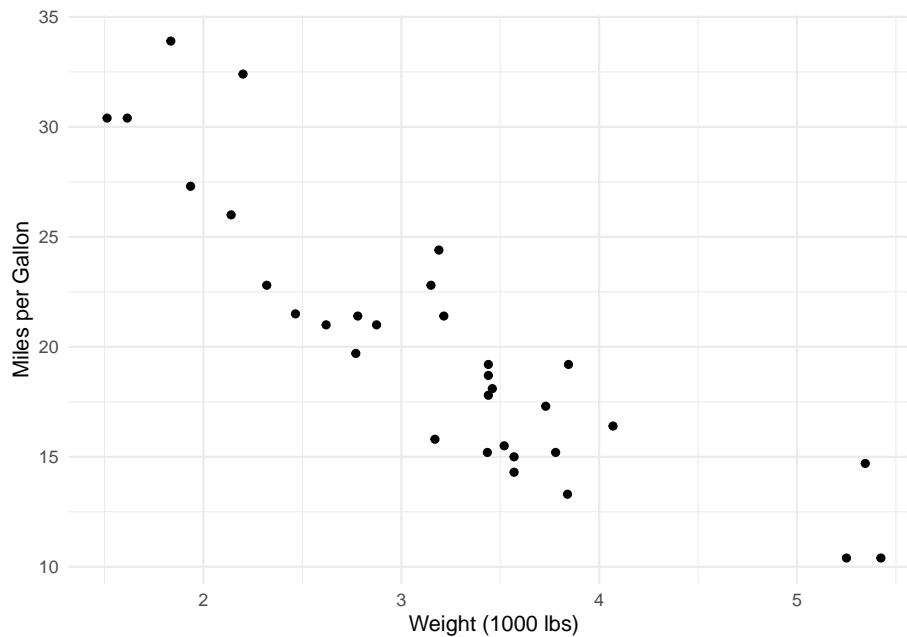
2. Load the “tidyverse” package into your R session.

```
library(tidyverse)
```

### 7.1.3 Part 3: Creating and Compiling an R Markdown File

1. Create a new R Markdown file in RStudio by clicking on “File” > “New File” > “R Markdown...”. Save the file in your working directory.
2. Add the following code to your R Markdown file to create a scatter plot of the mtcars dataset, showing the relationship between miles per gallon (mpg) and the weight of the car (wt).

```
# Create a scatter plot  
ggplot(data = mtcars, aes(x = wt, y = mpg)) +  
  geom_point() +  
  xlab("Weight (1000 lbs)") +  
  ylab("Miles per Gallon") +  
  theme_minimal()
```



3. Knit your R Markdown file to create an output document. Click the “Knit” button at the top of the RStudio script editor, and choose the output format you prefer (e.g., HTML, PDF, or Word).

## 7.2 Questions

### 7.2.1 1. How does the weight of a car (wt) affect its miles per gallon (mpg) based on the scatter plot you created?

[Click for answer](#)

**Answer:**

Based on the scatter plot, there appears to be a negative relationship between the weight of a car (wt) and its miles per gallon (mpg). As the weight of a car increases, its fuel efficiency (mpg) tends to decrease.

### 7.2.2 2. What is the importance of setting a working directory in R?

Click for answer

**Answer:**

Setting a working directory in R is important because it determines the default location where R will read from or write to when loading or saving files. This makes it easier to keep your files organized and ensures that your R scripts can access the necessary files without needing to specify the full file paths. It also simplifies sharing your R projects with others since the file paths within your scripts will be relative to the working directory.

### 7.2.3 3. Explain the role of R Markdown in creating reproducible research documents.

Click for answer

**Answer:**

R Markdown plays a crucial role in creating reproducible research documents by allowing you to combine text, code, and output (e.g., tables, figures) within a single document. This integration of narrative, data, and results makes it easier to document your data analysis process, ensuring that others can easily understand, reproduce, and build upon your work. R Markdown also supports various output formats (e.g., HTML, PDF, Word) to make it easy to share your research findings with others.

## Chapter 8

### Class Activity 2

Let's practice some common data assignments and manipulations in R.

- a. Create a vector of all integers from 4 to 10, and save it as **a1**.

Click for answer

```
a1 <- 4:10  
a1
```

```
[1] 4 5 6 7 8 9 10
```

- b. Create a vector of *even* integers from 4 to 10, and save it as **a2**.

Click for answer

```
a2 <- seq(4, 10, by=2)  
a2
```

```
[1] 4 6 8 10
```

- c. What do you get when you add **a1** to **a2**?

Click for answer

```
a1_plus_a2 <- a1 + a2  
a1_plus_a2
```

```
[1] 8 11 14 17 12 15 18
```

**Answer:** When you add **a1** to **a2**, you get a vector containing the element-wise sum: 8, 11, 14, 17, 12, 15, 18.

- d. What does the command **sum(a1)** do?

Click for answer

```
sum_a1 <- sum(a1)
sum_a1
```

```
[1] 49
```

**Answer:** The command `sum(a1)` calculates the sum of all elements in the vector `a1`. In this case, it returns 49.

e. What does the command `length(a1)` do?

Click for answer

```
length_a1 <- length(a1)
length_a1
```

```
[1] 7
```

**Answer:** The command `length(a1)` returns the number of elements in the vector `a1`. In this case, there are 7 elements.

f. Use the `sum` and `length` commands to calculate the average of the values in `a1`.

Click for answer

```
average_a1 <- sum(a1) / length(a1)
average_a1
```

```
[1] 7
```

**Answer:** The average of the values in `a1` is 7.



## Chapter 9

# Class Activity 3

```
# some interesting data objects
x <- c(3,6,9,5,10)
x.mat <- cbind(x, 2*x)
x.df <- data.frame(x=x,double.x=x*2)
my.list <- list(myVec=x, myDf=x.df, myString=c("hi","bye"))
```

### 9.1 Question 1: data types

- What data type is x?

Click for answer

*Answer:*

```
# code
typeof(x)
```

```
[1] "double"
```

- What data type is c(x, x/2)?

Click for answer

*Answer:*

```
# code
typeof(c(x, x/2))
```

```
[1] "double"
```

- What data type is c(x,NA)? What data type is c(x,"NA")?

Click for answer

Answer:

```
# code
typeof(c(x, NA))

[1] "double"

typeof(c(x, "NA"))

[1] "character"
```

## 9.2 Question 2: Subsetting and coercion

- How can we reverse the order of entries in `x`?

Click for answer

Answer:

```
# code
rev(x)

[1] 10  5  9  6  3

x[length(x):1]

[1] 10  5  9  6  3
```

- What does `which(x < 5)` equal?

Click for answer

Answer:

```
# code
which(x<5)

[1] 1
```

- Extract the element of `x` that corresponds to the location in the preceding question.

Click for answer

Answer:

```
# code
x[which(x<5)]

[1] 3
```

- What does `sum(c(TRUE,FALSE,TRUE,FALSE))` equal?

Click for answer

Answer:

```
# code  
sum(c(TRUE,FALSE,TRUE,FALSE))
```

```
[1] 2
```

- What does `sum(x[c(TRUE,FALSE,TRUE,FALSE)])` equal?

Click for answer

Answer:

```
# code  
sum(x[c(TRUE,FALSE,TRUE,FALSE, TRUE)])
```

```
[1] 22
```

- What does `sum(x < 5)` equal?

Click for answer

Answer:

```
# code  
sum(x < 5)
```

```
[1] 1
```

- What does `sum(x[x < 5])` equal?

Click for answer

Answer:

```
# code  
sum(x[x < 5])
```

```
[1] 3
```

- Why `dim(x.mat[1:2,1])` return NULL while `dim(x.mat[1:2,1:2])` returns a dimension?

Click for answer

Answer:

```
# code  
dim(x.mat[1:2,1])
```

NULL

```
dim(x.mat[1:2,1:2])
```

```
[1] 2 2
```

### 9.3 Question 3: Lists

- Using `my.list`, show three ways to write one command that gives the 3rd entry of variable `x` in data frame `myDf`

Click for answer

*Answer:*

```
# code  
my.list[[1]][3]
```

```
[1] 9
```

```
my.list[["myVec"]][3]
```

```
[1] 9
```

```
my.list[1]$myVec[3]
```

```
[1] 9
```

```
my.list$myVec[3]
```

```
[1] 9
```

- What class of object does the command `my.list[3]` return?

Click for answer

*Answer:*

```
# code  
class(my.list[3])
```

```
[1] "list"
```

- What class of object does the command `my.list[[3]]` return?

Click for answer

*Answer:*

```
# code  
class(my.list[[3]])
```

```
[1] "character"
```

- What class of object does the command `unlist(my.list)` return? Why are all the entries **characters**?

Click for answer

*Answer:*

```
# code  
class(unlist(my.list))
```

```
[1] "character"
```



## Chapter 10

### Class Activity 4

```
# Load the required libraries  
library(tidyverse)  
library(ggplot2)  
library(datasauRus)
```

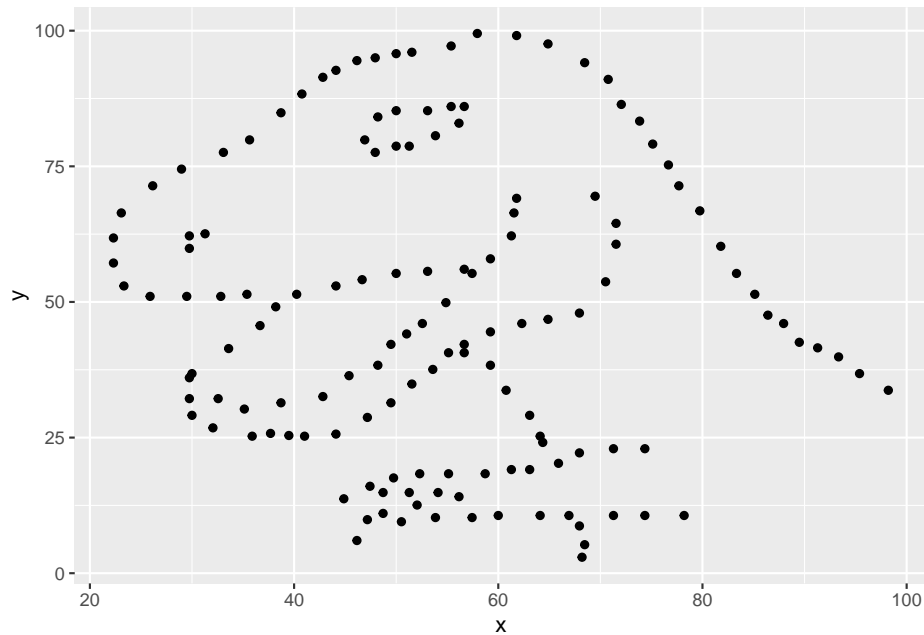
#### 10.1 Your turn 1

This worksheet will guide you through creating various plots using the `ggplot2` package in R. We will be using the `datasaurus_dozen` dataset from the `datasauRus` package for demonstration purposes. The dataset contains 13 different datasets, and we'll use them to create a variety of plots.

##### 10.1.1 Scatterplot

- a. Run the following code.

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



- b. You *must* remember to put the aesthetic mappings in the `aes()` function! What happens if you forget?

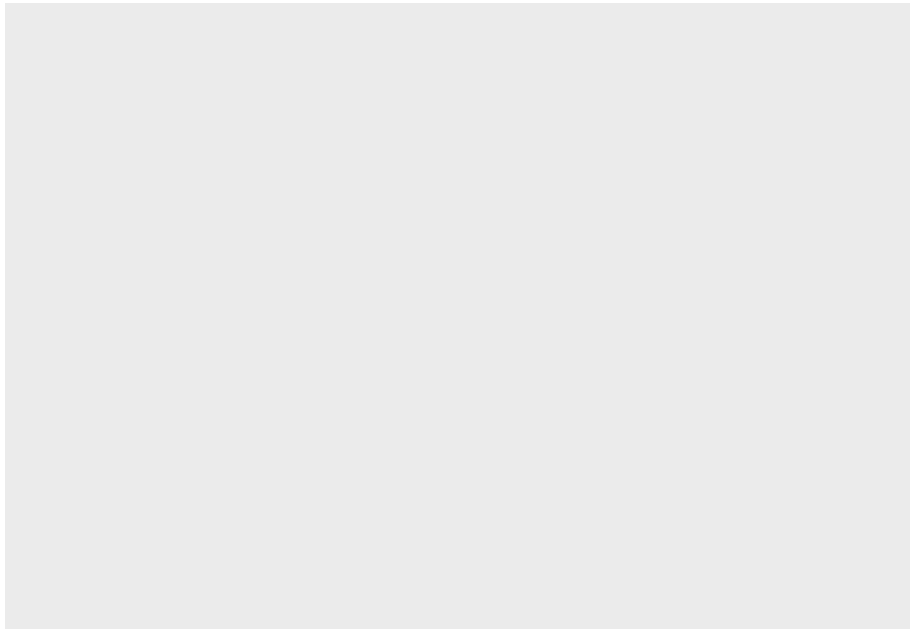
[Click for answer](#)

*Answer:*

If you forget to put the aesthetic mappings inside the `aes()` function, `ggplot2` will not be able to map the variables to the aesthetics correctly, and you might encounter an error or unexpected behavior in your plot.

```
# Add a layer and see what happens  
ggplot(data = dino_data , x = x , y = y)
```



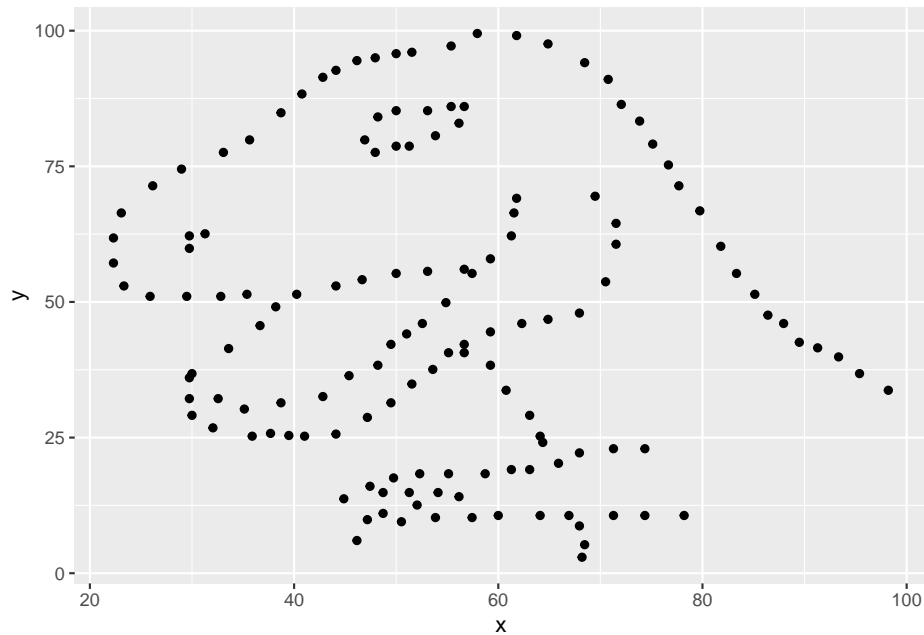


- c. The aesthetic mappings can be specified in the geom layer if you prefer, instead of the main `ggplot()` call. Give it a try:

Click for answer

*Answer:*

```
# Rebuild the scatterplot with your aesthetic mapping in the geom layer
ggplot(data = dino_data) +
  geom_point(aes(x = x, y = y))
```



### 10.1.2 Bar Plot

In this problem, we'll explore creating a bar plot using the `datasaurus_dozen` dataset.

- Create a new data frame containing the count of observations in each dataset.

Click for answer

*Answer:*

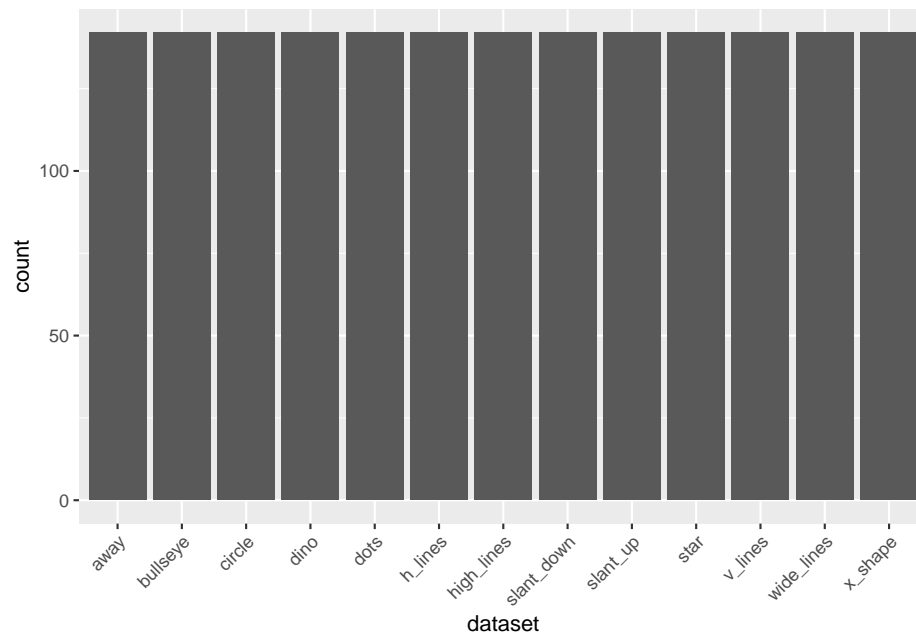
```
dataset_counts <- datasaurus_dozen %>%
  group_by(dataset) %>%
  summarise(count = n())
```

- Create a bar plot showing the number of observations in each dataset.

Click for answer

*Answer:*

```
ggplot(data = dataset_counts, aes(x = dataset, y = count)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



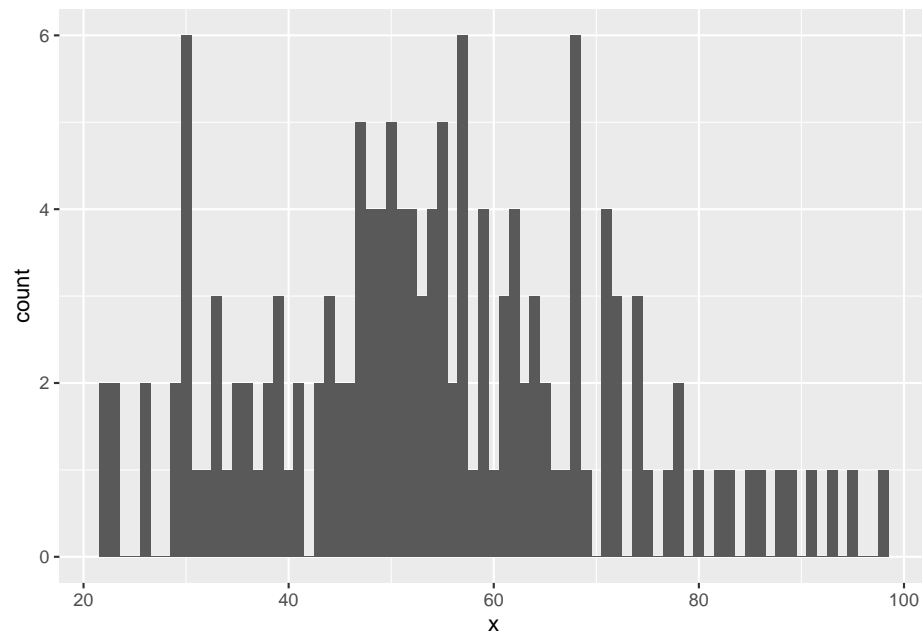
### 10.1.3 Histogram

- a. Create a histogram of the `x` variable for the `dino` dataset.

Click for answer

Answer:

```
ggplot(data = dino_data, aes(x = x)) +  
  geom_histogram(binwidth = 1)
```

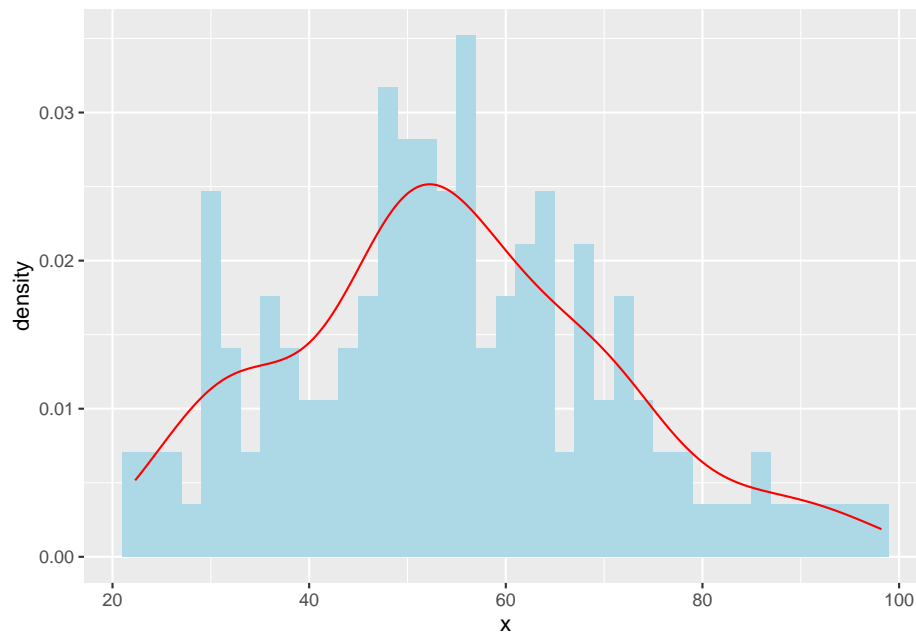


b. Overlay a density curve on the histogram.

Click for answer

*Answer:*

```
ggplot(data = dino_data, aes(x = x)) +
  geom_histogram(aes(y = after_stat(density)), binwidth = 2, fill = "lightblue") +
  geom_density(color = "red")
```



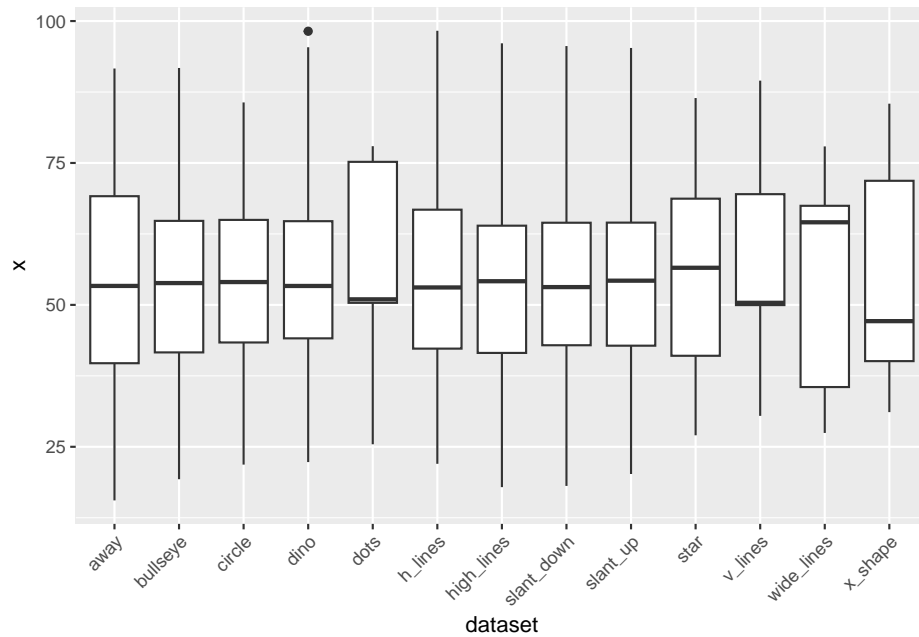
### 10.1.4 Boxplot

- a. Create a boxplot of the x variable for each dataset in `datasaurus_dozen`.

Click for answer

Answer:

```
ggplot(data = datasaurus_dozen, aes(x = dataset, y = x)) +  
  geom_boxplot() +  
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



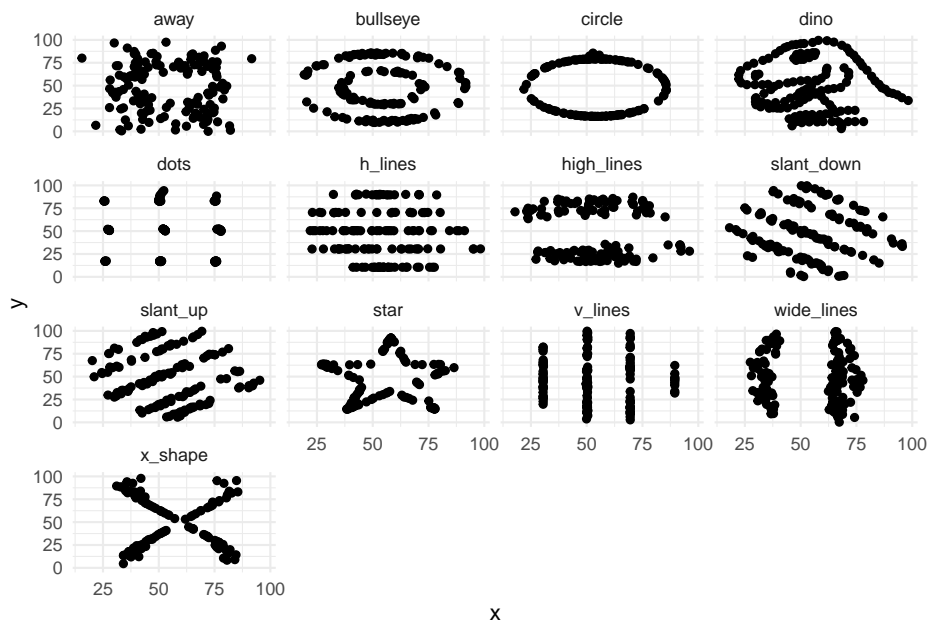
### 10.1.5 Faceting

Click for answer

*Answer:*

- Create a scatterplot of  $x$  vs.  $y$  for each dataset in `datasaurus_dozen` using `facet_wrap()`.

```
ggplot(data = datasaurus_dozen, aes(x = x, y = y)) +
  geom_point() +
  facet_wrap(~ dataset) +
  theme_minimal()
```



### 10.1.6 Variable Transformation

- a. The scatterplot of the `dino` dataset without any transformations is given below.

Click for answer

Answer:

```
ggplot(data = dino_data, aes(x = x, y = y)) +
  geom_point() +
  theme_minimal() -> p1
```

- b. Now, apply the square root transformation to both the x and y axes using the `scale_x_sqrt()` and `scale_y_sqrt()` functions in the `dino` dataset.

Click for answer

Answer:

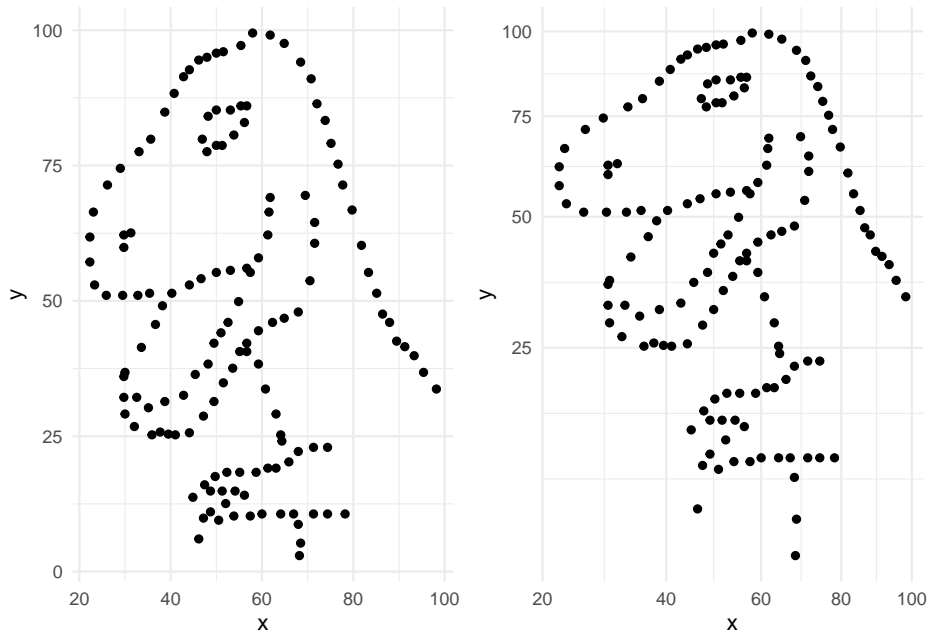
```
ggplot(data = dino_data, aes(x = x, y = y)) +
  geom_point() +
  scale_x_sqrt() +
  scale_y_sqrt() +
  theme_minimal() -> p2
```

- c. Finally, use `grid.arrange()` function from `gridExtra` package to plot the above two plots side-by-side. Which plot do you prefer and why?

Click for answer

*Answer:* The second plot is more revealing of a dinosaur than the first plot.

```
library(gridExtra)
grid.arrange(p1, p2, nrow = 1)
```



### 10.1.7 (Optional) Lne plot

- a. Create a line plot of the `x` variable over the `y` variable for the `dino` dataset. To make it more interesting, let's first calculate the rolling mean of the `y` variable.

Click for answer

*Answer:*

```
dino_data <- dino_data %>%
  arrange(x) %>%
  mutate(rolling_mean_y = zoo::rollmean(y, k = 5, fill = NA))

# Line plot
ggplot(data = dino_data, aes(x = x, y = rolling_mean_y)) +
  geom_line(color = "blue") +
  theme_minimal()
```



