Foundations of Data Science I: An Introduction to R/RStudio and Version Control for Data Science

Isabella Gollini

isabella.gollini@ucd.ie

What will we do today?

1. Introduction to R and RStudio
2. Data import and handling
3. Data manipulation and summaries with `dplyr`
4. Graphics with `ggplot2`

What will we do today?

1. Introduction to R and RStudio
2. Data import and handling
3. Data manipulation and summaries with `dplyr`
4. Graphics with `ggplot2`
5. RStudio Projects
6. Dynamic documents with Quarto
7. Version Control with GitHub

What can R do?

- Data Handling
- Analysis
- Reporting
- Programming

R Ecosystem

Base

base

create R objects
summaries
math functions

recommended

statistics
graphics
example data

Contributed Packages

CRAN
cran.r-project.org

main repos
~20000 pkgs

Bioconductor
bioconductor.org

bioinformatics
>2200 pkgs

GitHub
github.com

devel pkgs
GitHub-only pkgs

R Commands

We can type commands directly into the R console:

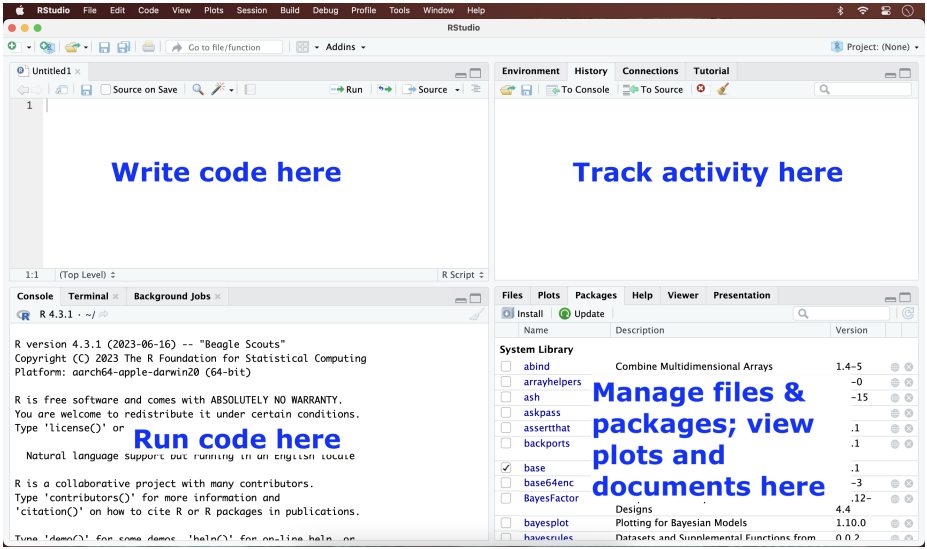
```
3 + 4
?"+" # look up help for "+". You need quotes!
x <- 3 + 4 # store 3 + 4 into the object x
x # print the object R
y <- log(x) # store the natural log of x in the object y
log(x) -> y # same as before using -> instead of <-
y = log(x) # same as before using the = instead of <-
3 == 4 # == is the comparison operator for equal
3 != 4 # != is the comparison operator for not equal
?log # look up help for the function log
ls() # list of objects in the current workspace
rm(x) # remove the object x
```

Linear Algebra

We can easily do vector operations and matrix operations in R:

```
M <- matrix(c(4, 2, 2, 3), 2, 2, byrow = FALSE) # create a matrix
M # print the matrix
solve(M) # finds the matrix inverse or a matrix equation solution
t(M) # matrix transpose
C <- chol(M) # Cholesky decomposition
t(C) %*% C # matrix multiplication
qr(M) # QR decomposition
eigen(M) # eigen decomposition
det(M) # Determinant
diag(M) # Find or set the matrix diagonal
```

RStudio IDE


RStudio File Edit Code View Plots Session Build Debug Profile Tools Window Help
RStudio
Go to file/function
Addins
Project: (None)
Untitled1
Source on Save
Run
Source
1
Write code here
1:1
(Top Level)
R Script
Environment History Connections Tutorial
To Console
To Source
Track activity here
Console Terminal Background Jobs
R 4.3.1 · ~/
R version 4.3.1 (2023-06-16) -- "Beagle Scouts"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or
Run code here
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Files Plots Packages Help Viewer Presentation
Install
Update
Name
Description
Version
System Library
abind
Combine Multidimensional Arrays
1.4-5
arrayhelpers
-0
ash
-15
askpass
assertthat
.1
backports
.1
base
.1
base64enc
-3
BayesFactor
.12-
Designs
4.4
bayesplot
Plotting for Bayesian Models
1.10.0
Manage files & packages; view plots and documents here

RStudio Features

Features provided by RStudio include:

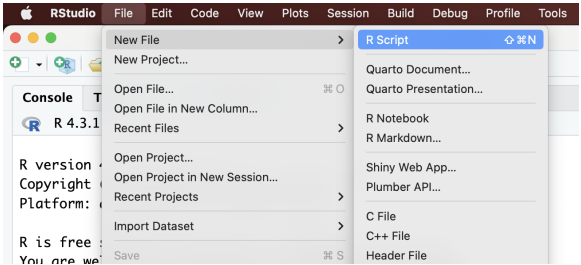
- syntax highlighting, code completion, smart indentation
- interactively send code chunks from editor to R
- organise multiple scripts, help files, plots
- search code and help files

RStudio Shortcuts & R Code Style

- RStudio provides a few shortcuts to help write code in the R console go to `Help - Keyboard Shortcuts Help`
- The goal of the R Programming Style Guide is to make our R code easier to read, share, and verify `https://style.tidyverse.org/`

R Scripts

You can store commands in the R Script in the source window and run these en bloc or line by line.



Text files saved with a `.R` suffix are recognised as R code.

More R Commands

```
data() # find out what standard data sets there are
plot(iris) # plot Fisher's iris data
head(iris, 4) # print the first 4 rows of the iris data
View(iris) # view the iris dataset on the viewer
summary(iris,  # summaries of the iris dataset up to the 2nd digit
        digits = 2) # you can split the command in two lines
sum(3, 4) # do the sum of 3 and 4
log(sum(1:10)) # Sum the numbers from 1 to 10 and then takes the natural log
```

Data Structures

Data structures are the building blocks of code. In R there are four main types of structure:

- vectors and factors
- matrices and arrays
- lists
- data frames

Vectors

A single number is a special case of a numeric vector. Vectors of length greater than one can be created using the concatenate function, c.

```
x <- c(1, 3, 6)
```

Vectors

A single number is a special case of a numeric vector. Vectors of length greater than one can be created using the concatenate function, c.

```
x <- c(1, 3, 6)
```

The elements of the vector must be of the same type: common types are numeric, character and logical

```
y <- c("red", "yellow", "green")
z <- c(TRUE, FALSE)
```

Vectors

A single number is a special case of a numeric vector. Vectors of length greater than one can be created using the concatenate function, `c`.

```
x <- c(1, 3, 6)
```

The elements of the vector must be of the same type: common types are numeric, character and logical

```
y <- c("red", "yellow", "green")
z <- c(TRUE, FALSE)
```

Missing values (of any type) are represented by the symbol `NA`.

Data Frames

Data sets are stored in R as **data frames**. These are structured as a list of objects, typically vectors, of the same length.

Data Frames

Data sets are stored in R as **data frames**. These are structured as a list of objects, typically vectors, of the same length.

```
str(iris)

# 'data.frame': 150 obs. of  5 variables:
#  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
#  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
#  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
#  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
#  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1
```

Data Frames

Data sets are stored in R as **data frames**. These are structured as a list of objects, typically vectors, of the same length.

```
str(iris)

# 'data.frame': 150 obs. of  5 variables:
#  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
#  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
#  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
#  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
#  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1
```

Here `Species` is a factor, a special data structure for categorial variables.

Write your own function

Calculate the sample standard deviation of the values in a vector.

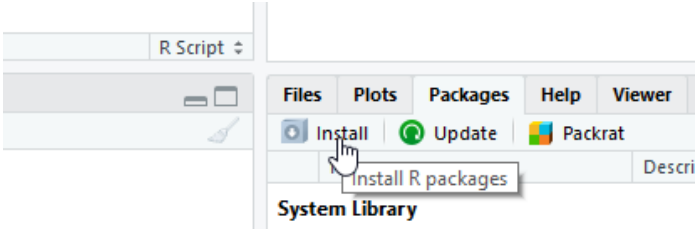
```
standard_deviation <- function(x){
  if(!is.numeric(x)) { # check the input is numeric
    stop('This function needs a numeric input!\n',
         'Not an object of class: ', class(x)[1])
  }
  xbar <- mean(x) # sample mean
  N <- length(x) # sample size
  sqrt(sum((x - xbar)^2) / (N - 1)) # output
}
```

Write your own function - Test it!

```
standard_deviation(iris)

# Error in standard_deviation(iris):  This function needs a numeric input!
# Not an object of class:  data.frame

standard_deviation(iris$Sepal.Length)

# [1] 0.8281

sd(iris$Sepal.Length)

# [1] 0.8281

apply(iris[,-5], 2, standard_deviation)

# Sepal.Length  Sepal.Width Petal.Length  Petal.Width
#       0.8281       0.4359       1.7653       0.7622
```

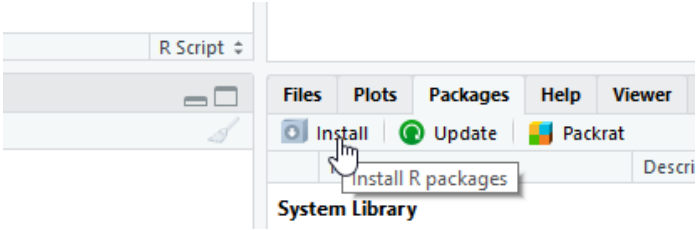
Install Packages

Most day-to-day work will require at least one contributed package.
CRAN packages can be installed from the **Packages** tab



Install Packages

Most day-to-day work will require at least one contributed package.
CRAN packages can be installed from the **Packages** tab

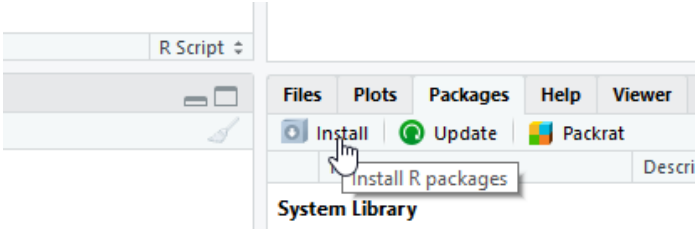


To use an installed package in your code, you must first load it from your package library.

```
library(ggplot2)
```

Install Packages

Most day-to-day work will require at least one contributed package.
CRAN packages can be installed from the **Packages** tab



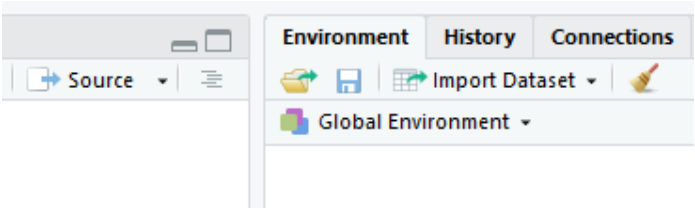
To use an installed package in your code, you must first load it from your package library.

```
library(ggplot2)
```

Sometimes an RStudio feature will require a contributed package. A pop-up will ask permission to install the package the first time, after that RStudio will load it automatically.

Data Input via Import Dataset

Using the **Import Dataset** dialog in RStudio



we can import files stored locally or online in the following formats:

- `.txt`/`.csv` via `read_delim`/`read_csv` from **readr**.
- `.xlsx` via `read_excel` from **readxl**.
- `.sav`/`.por`, `.sas7bdat` and `.dta` via `read_spss`, `read_sas` and `read_stata` respectively from **haven**.

Most of these functions also allow files to be compressed, e.g. as `.zip`.

Tibbles

The functions used by *Import Dataset* return data frames of class `"tbl_df"`, aka **tibbles**. The main differences are:

	data.frame	tibble
Printing (default)	Whole table	10 rows; columns to fit Prints column type
Subsetting	`dat[, 1]`, `dat$X1`, `dat[[1]]` all return vector	`dat[,1]` returns tibble `dat$X1`, `dat[[1]]` return vector
Strings	Converted to factor (default)	Left as character
Variable names	Made *syntactically valid* e.g. `Full name` -> `Full.name`	Left as is use e.g. `` dat$`Full name` ``

17

Data Input via Code

The **rio** package provides a common interface to the functions used by *Import Dataset* as well as many others.

Data Input via Code

The **rio** package provides a common interface to the functions used by *Import Dataset* as well as many others.
The data format is automatically recognised from the file extension. To read the data in as a tibble, we use the `setclass` argument.

Data Input via Code

The **rio** package provides a common interface to the functions used by *Import Dataset* as well as many others.
The data format is automatically recognised from the file extension. To read the data in as a tibble, we use the `setclass` argument.

```
library(rio)
compsci <- import("compsci.csv", setclass = "tibble")
cyclist <- import("cyclist.xlsx", setclass = "tibble")
```

Data Input via Code

The **rio** package provides a common interface to the functions used by *Import Dataset* as well as many others.
The data format is automatically recognised from the file extension. To read the data in as a tibble, we use the `setclass` argument.

```
library(rio)
compsci <- import("compsci.csv", setclass = "tibble")
cyclist <- import("cyclist.xlsx", setclass = "tibble")
```

See `?rio` for the underlying functions used for each format and the corresponding optional arguments, e.g. the skip argument to `read_excel` to skip a certain number of rows.

RStudio Projects

An RStudio project is a context for work on a specific project

- automatically sets working directory to project root
- has separate workspace and command history
- works well with version control via git or svn

RStudio Projects

An RStudio project is a context for work on a specific project

- automatically sets working directory to project root
- has separate workspace and command history
- works well with version control via git or svn

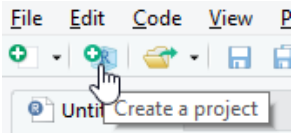
Create a project from a new/existing directory via the *File* Menu or the *New Project* button.

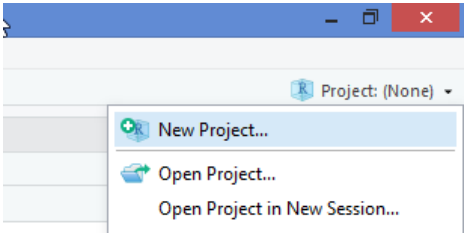
RStudio Projects

An RStudio project is a context for work on a specific project

- automatically sets working directory to project root
- has separate workspace and command history
- works well with version control via git or svn

Create a project from a new/existing directory via the *File* Menu or the *New Project* button. Switch project, or open a different project in a new RStudio instance via the Project menu.





Quarto Documents

Quarto documents (`.qmd`) intersperse code chunks (R, Python, Javascript, and Julia etc.) with markdown text (similar to R Markdown)

YAML Header

```
---
title: "Report"
format: html
---
```

Markdown Text

```
## First section

This report summarises the `cars` dataset.
```

R code chunk

````
```{r}
#| label: summary-cars
summary(cars)
```
````

Rendering

The `.qmd` file can be rendered to produce a document (HTML, PDF, docx) integrating the code output.

Report

First section

This report summarises the `cars` dataset.

```
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
```

Quarto for R Markdown Users

- Quarto and R Markdown are similar, but
  - R Markdown is fundamentally tied to R
  - Quarto is at its core multi-language and multi-engine (supporting Knitr, Jupyter, and Observable).

Quarto for R Markdown Users

- Quarto and R Markdown are similar, but
  - R Markdown is fundamentally tied to R
  - Quarto is at its core multi-language and multi-engine (supporting Knitr, Jupyter, and Observable).
- Quarto combines the functionality of R Markdown, bookdown, distill, xaringan.
- R Markdown will continue to work and be supported, but
- New features will be implemented in Quarto only.

See: `https://quarto.org/docs/faq/rmarkdown.html`

Quarto in JupyterLab

- You don't need R/RStudio to use Quarto, you can use Quarto in JupyterLab.

See: `https://quarto.org/docs/tools/jupyter-lab.html`

Getting Started with Quarto

1. To create a new Quarto document; go to *File - New File - Quarto Document*
   - The first time you use Quarto on your machine you may be asked to install some R packages; if so, press `Yes`.
2. Select the type of output document you want to create.
   - You are able to produce `HTML` output on any computer.
   - To produce `pdf` output you need to have LaTeX installed.

```
install.packages("tinytex")
tinytex::install_tinytex()
```

3. Open the "Markdown Quick Reference": *Help - Markdown Quick Reference*
   Quarto is a very powerful tool; an extended guide with tutorials is available on the Quarto website: `https://quarto.org/docs/get-started/hello/rstudio.html`

Your Turn

1. Open `infant.Rproj`, which is an RStudio project file.
2. From the Files tab, open the `infant.qmd` Quarto file.
3. In the chunk labelled `import-data`, write some code that will import the `infant.xlsx` file and create a tibble named `infant`.
4. Load any required packages in the `setup` chunk.
5. Run the code in the `import-data` chunk (the `setup` chunk is run automatically).
6. Use `View()` in the console to inspect the result.
7. Install the **skimr** package and use the `skim` function to summarise the data set.

Data Pre-processing

The imported data are a subset of records from a US Child Health and Development Study, corresponding to live births of a single male foetus.
The data requires a lot of pre-processing

- converting numeric variables to factors
- converting coded values to missing values
- filtering rows and selecting columns

The *dplyr* package can be used to help with many of these steps

dplyr

The *dplyr* package provides the following key functions to operate on data frames

- `filter()`
- `arrange()`
- `select()` (and `rename()`)
- `mutate()`
- `summarise()`

filter()

`filter()` selects rows of data by criteria.

filter()

`filter()` selects rows of data by criteria.

```
library(dplyr)
infant <- filter(infant, smoke != 9 & age > 18)
```

filter()

`filter()` selects rows of data by criteria.

```
library(dplyr)
infant <- filter(infant, smoke != 9 & age > 18)
```

Building block	R code
Binary comparisons	>, <, ==, <=, >=, !=
Logical operators	or | and &, not !
Value matching	e.g. `x %in% 6:9`
Missing indicator	e.g. `is.na(x)`

`select()`

`select()` selects variables from the data frame.

select()

`select()` selects variables from the data frame.

- Select named columns:

```
select(infant, gestation, sex)
```

select()

`select()` selects variables from the data frame.

- Select named columns:

```
select(infant, gestation, sex)
```

- Select a sequence of columns, along with an individual column

```
select(infant, pluralty:gestation, parity)
```

select()

`select()` selects variables from the data frame.

- Select named columns:

```
select(infant, gestation, sex)
```

- Select a sequence of columns, along with an individual column

```
select(infant, pluralty:gestation, parity)
```

- Select all columns *except* specified columns

```
select(infant, -(id:outcome), -sex)
```

mutate()

`mutate()` computes new columns based on existing columns. Re-using an existing name replaces the old variable.

mutate()

`mutate()` computes new columns based on existing columns. Re-using an existing name replaces the old variable.

```
mutate(infant,
       wt = ifelse(wt == 999, NA, wt),
       wt = wt * 0.4536)
```

Chaining

We can use |> to pipe the data from one step to the next.

Chaining

We can use |> to pipe the data from one step to the next.

```
infant <- infant |>
    filter(smoke != 9 & age > 18) |>
    select(-(id:outcome), -sex) |>
    mutate(wt = ifelse(wt == 999, NA, wt),
           wt = wt * 0.4536)
```

Chaining

We can use |> to pipe the data from one step to the next.

```
infant <- infant |>
    filter(smoke != 9 & age > 18) |>
    select(-(id:outcome), -sex) |>
    mutate(wt = ifelse(wt == 999, NA, wt),
           wt = wt * 0.4536)
```

Any function with data as the first argument can be added to the data pipeline. See the help file ?"|>" for more options.

Chaining |> vs %>%

- |> is the native pipe operator (available in R 4.1.0 or later)
- You might be familiar with the %>% pipe provided by the `magrittr`/`tidyverse` package.
- For most cases |> and %>% behave identically.
- It's preferable to use |> since it's part of "base R", and it's simpler.
- For more info look at this blog post: `https://www.tidyverse.org/blog/2023/04/base-vs-magrittr-pipe/`

summarise()

`summarise()` function is for computing single number summaries of variables, so typically comes at the end of a pipeline or in a separate step

```
stats <- summarise(infant,
                   `Mean mother's weight (kg)` = mean(wt, na.rm = TRUE),
                   `Sample size` = sum(!is.na(wt)),
                   `Total sample size` = n())
stats

# # A tibble: 1 x 3
#   `Mean mother's weight (kg)` `Sample size` `Total sample size`
#                         <dbl>         <int>                <int>
# 1                        58.4          1167                 1203
```

Output as a Markdown table

In a Quarto document we can convert the result to a markdown table using `kable` from the **knitr** package

```
library(knitr)
kable(stats, align = "ccc")
```

or

```
knitr::kable(stats, align = "ccc")
```

Grouped Operations

Grouping can be set on a data frame using `group_by`. This is most useful `summarise()`

```
infant <- infant |> filter(race != 10) |>
    mutate(`Ethnic group` = recode(race, `6` = "Latino", `7` = "Black",
                                   `8` = "Asian", `9` = "Mixed",
                                   .default = "White"))
res <- infant |> group_by(`Ethnic group`) |>
    summarise(`Mean weight (kg)` = mean(wt, na.rm = TRUE))
kable(res, align = "lc")
```

Ethnic group	Mean weight (kg)
Asian	49.84
Black	62.63
Latino	54.12
Mixed	58.46
White	57.86

Your Turn

`cut()` (in the **base** package) can be used to create a factor from a continuous variable, e.g.:

```
cut(c(0.12, 1.37, 0.4, 2.3), breaks = c(0, 1, 2, 3))

# [1] (0,1] (1,2] (0,1] (2,3]
# Levels: (0,1] (1,2] (2,3]
```

where $(0, 1]$; is the set of numbers > 0 and ≤ 1, etc.

1. The infant birth weight `bwt` is given in ounces. In the `table-bwt` chunk use `mutate()` to create a new factor called `Birth weight (g)`, by converting `bwt` to grams through multiplication by 28.35 and then converting the result to a factor with the following categories: $(1500, 2000]$, $(2000, 2500]$, $(2500, 3000]$, $(3000, 3500]$, and $(3500, 5000]$.
2. An infant is categorised as low weight if its birth weight is ≤ 2500 grams, regardless of gestation. Use `group_by()` to group the data by the new factor weight factor, then pipe to `summarise()` to count the number of infants in each weight category.

Plots

- In RStudio, graphs are displayed in the Plots window. The plot is sized to fit the window and will be rescaled if the size of the window is changed.

Plots

- In RStudio, graphs are displayed in the Plots window. The plot is sized to fit the window and will be rescaled if the size of the window is changed.
- Back and forward arrows allow you to navigate through graphs that have been plotted.

Plots

- In RStudio, graphs are displayed in the Plots window. The plot is sized to fit the window and will be rescaled if the size of the window is changed.
- Back and forward arrows allow you to navigate through graphs that have been plotted.
- Graphs can be saved in various formats using the Export drop down menu, which also has an option to copy to the clipboard.

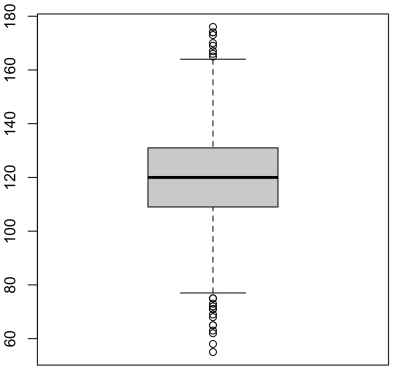
Plots

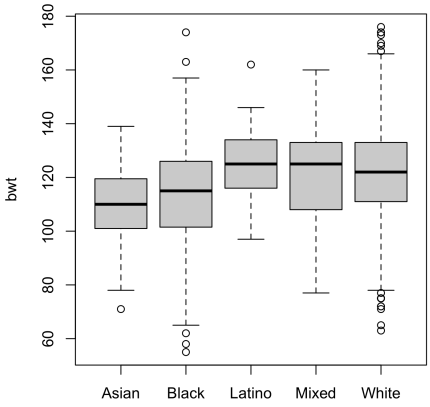
- In RStudio, graphs are displayed in the Plots window. The plot is sized to fit the window and will be rescaled if the size of the window is changed.
- Back and forward arrows allow you to navigate through graphs that have been plotted.
- Graphs can be saved in various formats using the Export drop down menu, which also has an option to copy to the clipboard.
- First we consider "no-frills" plots, for quick exploratory plots.

```
infant <- infant |>
    mutate(gestation = ifelse(gestation == 999,
                              NA, gestation))
```

Boxplots

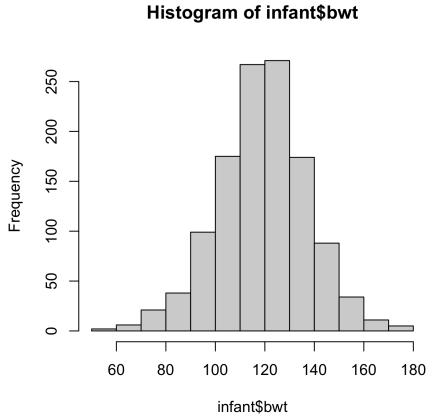
```
boxplot(infant$bwt)
with(infant, boxplot(bwt ~ `Ethnic group`))
```

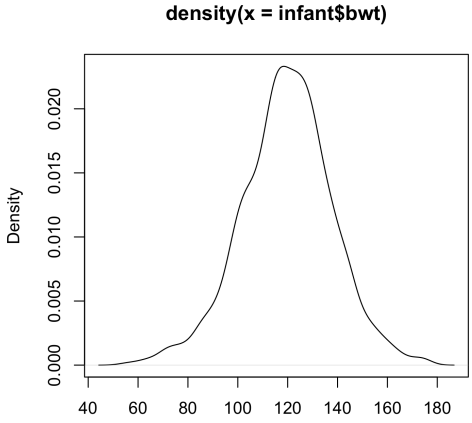




Histogram/Density

```
hist(infant$bwt)
plot(density(infant$bwt))
```



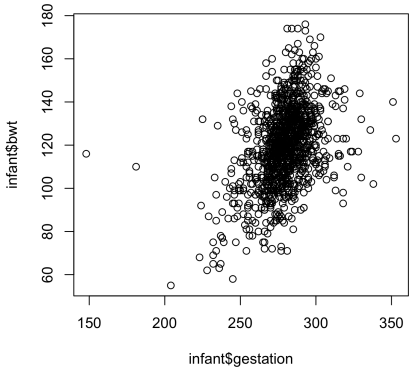


Scatterplots

```
plot(bwt ~ gestation, data = infant)
```

or

```
plot(infant$gestation, infant$bwt)
```



ggplot2

ggplot2 is a package that produces plots based on the *grammar of graphics* (Leland Wilkinson), the idea that you can build every graph from the same components

- a data set
- a co-ordinate system
- and *geoms* visual marks that represent data points

ggplot2

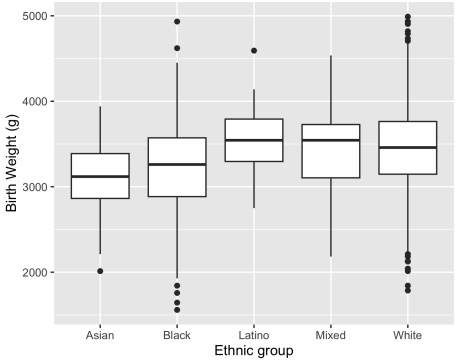
ggplot2 is a package that produces plots based on the *grammar of graphics* (Leland Wilkinson), the idea that you can build every graph from the same components

- a data set
- a co-ordinate system
- and *geoms* visual marks that represent data points

To display values, you map variables in the data to visual properties of the geom (*aesthetics*), like size, colour, x-axis and y-axis.
It provides a unified system for graphics, simplifies tasks such as adding legends, and is fully customisable.

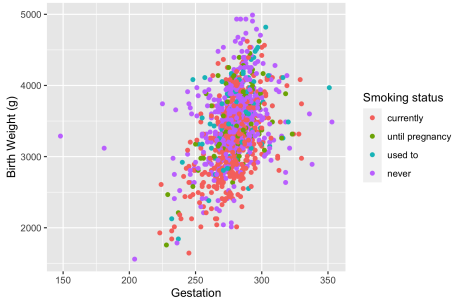
ggplot2 boxplots

```
library(ggplot2)
ggplot(infant, aes(y = bwt * 28.35, x = `Ethnic group`)) +
    geom_boxplot() +
    ylab("Birth Weight (g)")
```



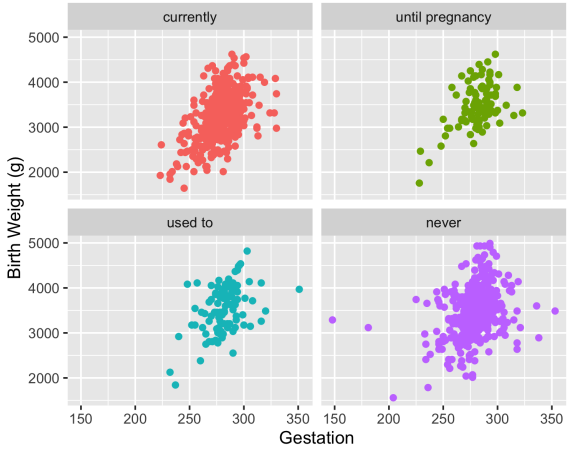
ggplot2 with Colour

```
infant <- mutate(infant, smoke = recode_factor(smoke,
                                    `1` = "currently", `2`= "until pregnancy",
                                    `3` = "used to", `0` = "never"))
p <- ggplot(infant, aes(y = bwt * 28.35, x = gestation, color = smoke)) +
    geom_point() + labs(x = "Gestation", y = "Birth Weight (g)",
                        color = "Smoking status")
p
```



ggplot2 Facetting

```
p + facet_wrap(~ smoke) + guides(color = "none")
```

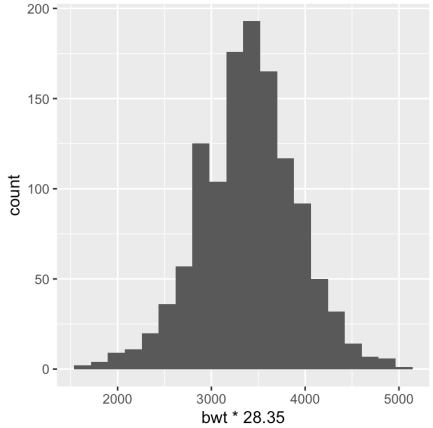


Plots in Quarto

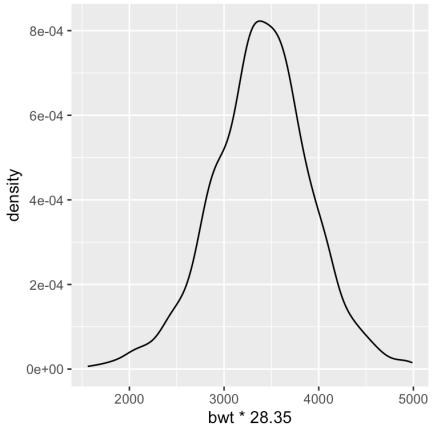
The chunk options enable you to arrange plots:

````
```{r}
#| label: plots-bwt
#| layout-ncol: 2
#| fig-width: 4
#| fig-height: 4
#| fig-cap:
#|   - "`ggplot` histogram"
#|   - "`ggplot` density curve"
ggplot(infant, aes(x = bwt * 28.35)) +
  geom_histogram(bins = 20)
ggplot(infant, aes(x = bwt * 28.35)) +
  geom_density()
```
````

Plots in Quarto



`ggplot` histogram

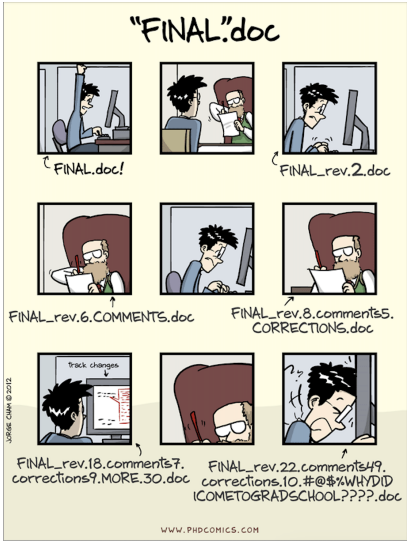


`ggplot` density curve

Your Turn!

1. Do some nice graph!
2. Include them into your Quarto file, select appropriate size for the figure and caption.
3. Briefly comment your findings in your Quarto file.

Why Version Control



Why Version Control (& GitHub)

- Keep track of all changes (complete change history for each single file)
- Backup repository
- Collaborate
- `R`: Distribution/dissemination
- `R`: Communication with users (issues, pull requests)
- `R`: Easily build websites for R package with GitHub Pages and `pkgdown`

Version Control

- Git: free and open source distributed version control system. It tracks changes in the files.

Version Control

- Git: free and open source distributed version control system. It tracks changes in the files.
- GitHub: is the most popular website to host your Git repository. Other hosting services include Bitbucket, and Gitlab.

Version Control

- Some of the following slides are from Garrett Grolemund (https://github.com/rstudio/webinars/blob/master/15-RStudio-essentials/5-Git/5-git.pdf) made for this webminar: https://posit.co/resources/videos/managing-part-2-github-and-rstudio/

- The best reference to learn version control with R is: "Happy Git and Github for the useR" https://happygitwithr.com/


git

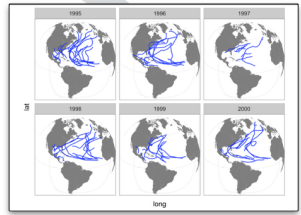
History

1st Commit → 2nd Commit → 3rd Commit → 4th Commit → 5th Commit

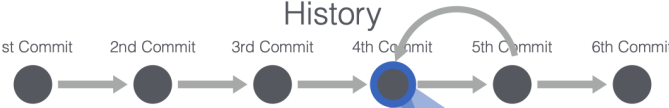
```
"name","year","time","lat","long"
"Allison",1995,1995-06-03 00:00:00,17.4,-84.3
"Allison",1995,1995-06-03 06:00:00,18.3,-84.9
"Allison",1995,1995-06-03 12:00:00,19.3,-85.7
"Allison",1995,1995-06-03 18:00:00,20.6,-85.8
"Allison",1995,1995-06-04 00:00:00,22,-86
"Allison",1995,1995-06-04 06:00:00,23.3,-86.3
"Allison",1995,1995-06-04 12:00:00,24.7,-86.2
"Allison",1995,1995-06-04 18:00:00,26.2,-86.2
"Allison",1995,1995-06-05 00:00:00,27.6,-86.1
"Allison",1995,1995-06-05 06:00:00,28.5,-85.6
"Allison",1995,1995-06-05 12:00:00,29.6,-84.7
"Allison",1995,1995-06-05 18:00:00,30.7,-83.8
"Allison",1995,1995-06-06 00:00:00,31.8,-82.8
"Allison",1995,1995-06-06 06:00:00,32.7,-81.5
"Allison",1995,1995-06-06 12:00:00,33.6,-80
"Allison",1995,1995-06-06 18:00:00,34.5,-78.1
"Allison",1995,1995-06-07 00:00:00,35.6,-75.9
"Allison",1995,1995-06-07 06:00:00,37.1,-73.6
"Allison",1995,1995-06-07 12:00:00,38.5,-71
"Allison",1995,1995-06-07 18:00:00,39.8,-69.2
"Allison",1995,1995-06-08 00:00:00,41,-67.7
"Allison",1995,1995-06-08 06:00:00,42.4,-66
"Allison",1995,1995-06-08 12:00:00,43.8,-63.7
```

```
# 0-Clean.R

library(dplyr)
library(lubridate)

storms <- read.csv("storms.csv")

storms <- storms %>%
  mutate(time = ymd_h(paste(year, month, day,
    hour))) %>%
  select(name, year, time, lat, long,
pressure, wind, type)

write.csv(storms, file = "storms.csv",
  row.names = FALSE)
```

```
# 1-Plot.R

library(ggplot2)
library(dplyr)

map <- map_data("world") %>%
  filter(region != "USSR")

ggplot(storms, aes(x = long, y = lat)) +
  geom_polygon(aes(group = group),
    fill = "grey50", data = map) +
  geom_path(aes(group = name),
    color = "blue" ) +
  facet_wrap(~ year) +
  theme_bw() +
  coord_map(projection = "ortho",
    orientation = c(21, -60, 0))

ggsave("storms.png", width = 7, height = 5)
```



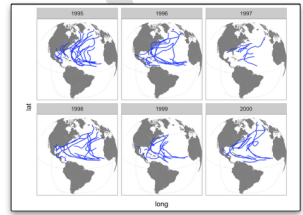
Project



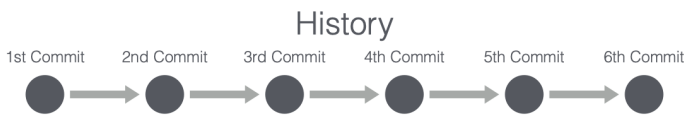

```
"name","year","time","lat","long"
"Allison",1995,1995-06-03 00:00:00,17.4,-84.3
"Allison",1995,1995-06-03 06:00:00,18.3,-84.9
"Allison",1995,1995-06-03 12:00:00,19.3,-85.7
"Allison",1995,1995-06-03 18:00:00,20.6,-85.8
"Allison",1995,1995-06-04 00:00:00,22,-86
"Allison",1995,1995-06-04 06:00:00,23.3,-86.3
"Allison",1995,1995-06-04 12:00:00,24.7,-86.2
"Allison",1995,1995-06-04 18:00:00,26.2,-86.2
"Allison",1995,1995-06-05 00:00:00,27.6,-86.1
"Allison",1995,1995-06-05 06:00:00,28.5,-85.6
"Allison",1995,1995-06-05 12:00:00,29.6,-84.7
"Allison",1995,1995-06-05 18:00:00,30.7,-83.8
"Allison",1995,1995-06-06 00:00:00,31.8,-82.8
"Allison",1995,1995-06-06 06:00:00,32.7,-81.5
```

```
# 0-Clean.R

library(dplyr)
library(lubridate)

storms <- read.csv("storms.csv")

storms <- storms %>%
  mutate(time = ymd_h(paste(year, month, day,
    hour))) %>%
  select(name, year, time, lat, long,
pressure, wind, type)

write.csv(storms, file = "storms.csv",
  row.names = FALSE)
```

```
# 1-Plot.R

library(ggplot2)
library(dplyr)

map <- map_data("world") %>%
  filter(region != "USSR")

ggplot(storms, aes(x = long, y = lat)) +
  geom_polygon(aes(group = group),
    fill = "grey50", data = map) +
  geom_path(aes(group = name),
    color = "blue" ) +
  facet_wrap(~ year) +
  theme_bw() +
  coord_map(projection = "ortho",
    orientation = c(21, -60, 0))

ggsave("storms.png", width = 7, height = 5)
```



Project


History
1st Commit
2nd Commit
3rd Commit
4th Commit
5th Commit
6th Commit

.git

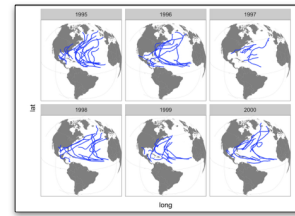
History

1st Commit 2nd Commit 3rd Commit 4th Commit 5th Commit 6th Commit

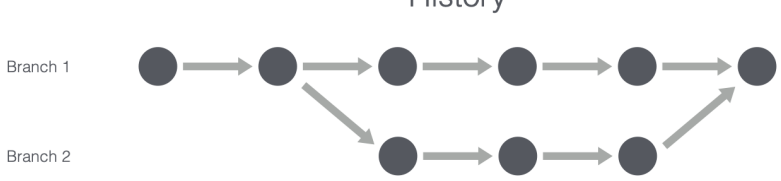
```
"name","year","time","lat","long"
"Allison",1995,1995-06-03 00:00:00,17.4,-84.3
"Allison",1995,1995-06-03 06:00:00,18.3,-84.9
"Allison",1995,1995-06-03 12:00:00,19.3,-85.7
"Allison",1995,1995-06-03 18:00:00,20.6,-85.8
"Allison",1995,1995-06-04 00:00:00,22,-86
"Allison",1995,1995-06-04 06:00:00,23.3,-86.3
"Allison",1995,1995-06-04 12:00:00,24.7,-86.2
"Allison",1995,1995-06-04 18:00:00,26.2,-86.2
"Allison",1995,1995-06-05 00:00:00,27.6,-86.1
"Allison",1995,1995-06-05 06:00:00,28.5,-85.6
"Allison",1995,1995-06-05 12:00:00,29.6,-84.7
"Allison",1995,1995-06-05 18:00:00,30.7,-83.8
"Allison",1995,1995-06-06 00:00:00,31.8,-82.8
"Allison",1995,1995-06-06 06:00:00,32.7,-81.5
"Allison",1995,1995-06-06 12:00:00,33.6,-80
"Allison",1995,1995-06-06 18:00:00,34.5,-78.1
"Allison",1995,1995-06-07 00:00:00,35.6,-75.9
"Allison",1995,1995-06-07 06:00:00,37.1,-73.6
"Allison",1995,1995-06-07 12:00:00,38.5,-71
"Allison",1995,1995-06-07 18:00:00,39.8,-69.2
"Allison",1995,1995-06-08 00:00:00,41,-67.7
"Allison",1995,1995-06-08 06:00:00,42.4,-66
"Allison",1995,1995-06-08 12:00:00,43.8,-63.7
```

```
# 0-Clean.R

library(dplyr)
library(lubridate)

storms <- read.csv("storms.csv")

storms <- storms %>%
  mutate(time = ymd_h(paste(year, month, day,
    hour))) %>%
  select(name, year, time, lat, long,
pressure, wind, type)

write.csv(storms, file = "storms.csv",
  row.names = FALSE)
```

```
# 1-Plot.R

library(ggplot2)
library(dplyr)

map <- map_data("world") %>%
  filter(region != "USSR")

ggplot(storms, aes(x = long, y = lat)) +
  geom_polygon(aes(group = group),
    fill = "grey50", data = map) +
  geom_path(aes(group = name),
    color = "blue" ) +
  facet_wrap(~ year) +
  theme_bw() +
  coord_map(projection = "ortho",
    orientation = c(21, -60, 0))

ggsave("storms.png", width = 7, height = 5)
```

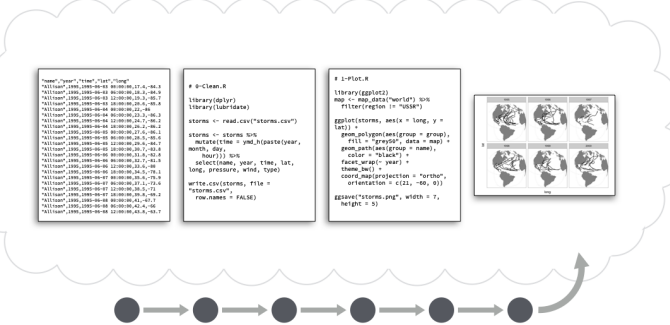


Project


History
Branch 1
Branch 2

"Official" Version

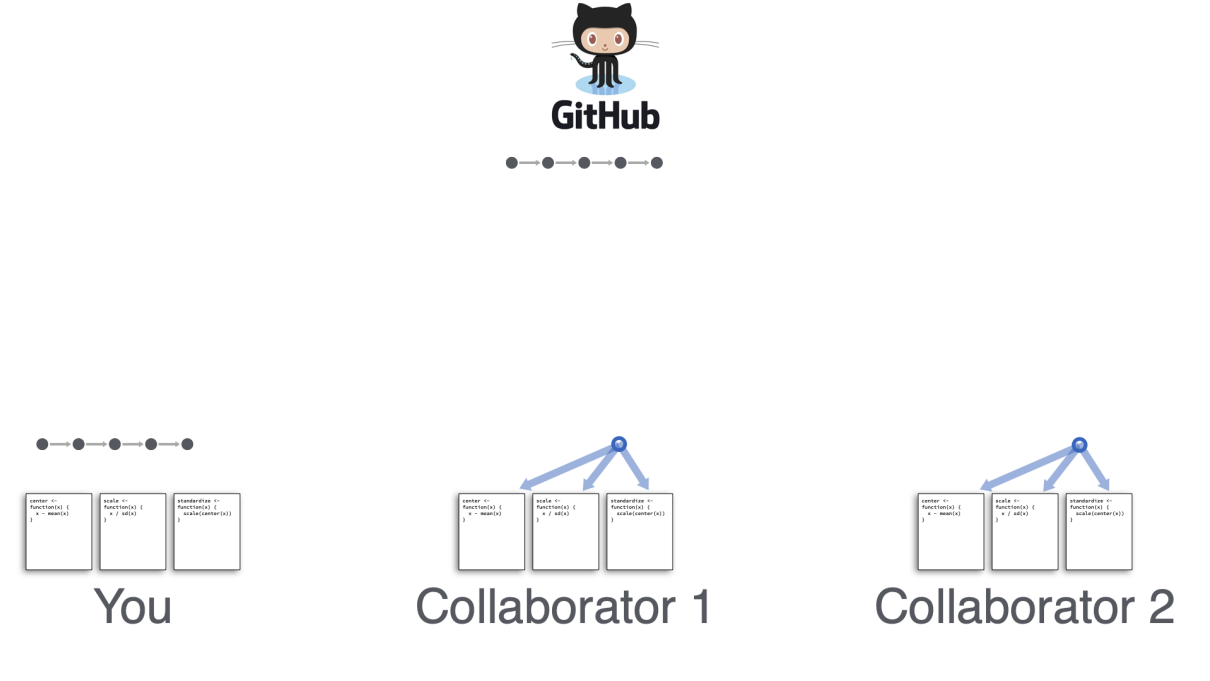
implied by commit history

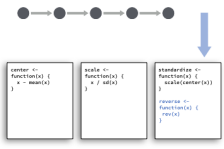


Real Life Version

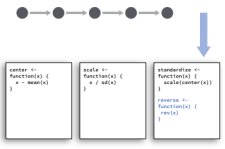
uncommitted changes
in blue

www.github.com

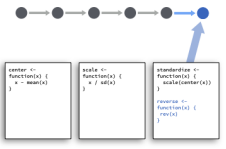

GitHub
center <-
function(x) {
x - mean(x)
}
scale <-
function(x) {
x / sd(x)
}
standardize <-
function(x) {
scale(center(x))
}
You
Collaborator 1
Collaborator 2



You

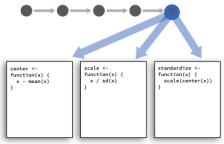


Collaborator 1

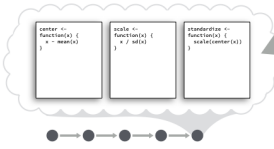


Collaborator 2

3. Github Version
available to public



2."Official" Version
implied by commit history



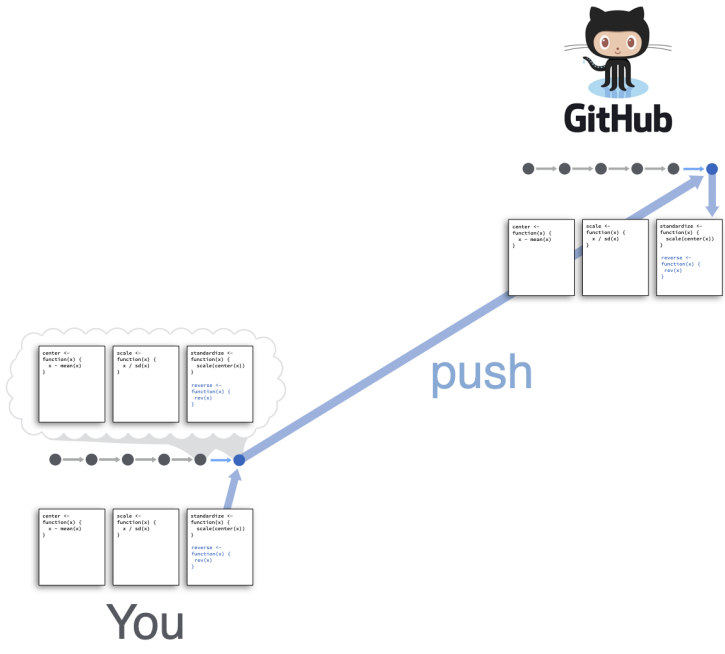
1.Real Life Version
in your working directory

You

3. Github Version
available to public

2."Official" Version
implied by commit history

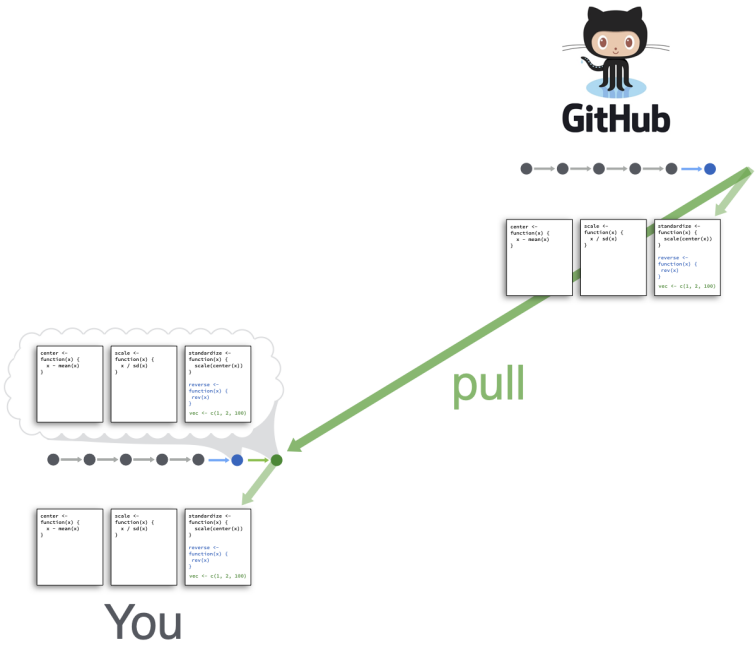
1.Real Life Version
in your working directory



3. Github Version
available to public

2."Official" Version
implied by commit history

1.Real Life Version
in your working directory



Git & GitHub glossary

- A **repository** contains all of the project files, and stores each file's revision history.

Git & GitHub glossary

- A **repository** contains all of the project files, and stores each file's revision history.
- A **commit** is an individual change to a file (or set of files). It saves changes on local repository.

Git & GitHub glossary

- A **repository** contains all of the project files, and stores each file's revision history.
- A **commit** is an individual change to a file (or set of files). It saves changes on local repository.
- To **push** means to send your committed changes to a remote repository on GitHub.com.

Git & GitHub glossary

- A **repository** contains all of the project files, and stores each file's revision history.
- A **commit** is an individual change to a file (or set of files). It saves changes on local repository.
- To **push** means to send your committed changes to a remote repository on GitHub.com.
- **Pull** refers to when you are fetching in changes (from GitHub) and merging them.

Git & GitHub glossary

- A **repository** contains all of the project files, and stores each file's revision history.
- A **commit** is an individual change to a file (or set of files). It saves changes on local repository.
- To **push** means to send your committed changes to a remote repository on GitHub.com.
- **Pull** refers to when you are fetching in changes (from GitHub) and merging them.
- A **fork** is a personal copy of another user's repository that lives on your account. Forks allow you to freely make changes to a project without affecting the original upstream repository.

Git & GitHub glossary

- A **repository** contains all of the project files, and stores each file's revision history.
- A **commit** is an individual change to a file (or set of files). It saves changes on local repository.
- To **push** means to send your committed changes to a remote repository on GitHub.com.
- **Pull** refers to when you are fetching in changes (from GitHub) and merging them.
- A **fork** is a personal copy of another user's repository that lives on your account. Forks allow you to freely make changes to a project without affecting the original upstream repository.
- A **clone** is a copy of a repository that lives on your computer.

Git & GitHub glossary

- A **repository** contains all of the project files, and stores each file's revision history.
- A **commit** is an individual change to a file (or set of files). It saves changes on local repository.
- To **push** means to send your committed changes to a remote repository on GitHub.com.
- **Pull** refers to when you are fetching in changes (from GitHub) and merging them.
- A **fork** is a personal copy of another user's repository that lives on your account. Forks allow you to freely make changes to a project without affecting the original upstream repository.
- A **clone** is a copy of a repository that lives on your computer.
- A **branch** is a parallel version of a repository. It is contained within the repository, but does not affect the main branch allowing you to work freely without disrupting the "live" version.

Git & GitHub glossary

- A **repository** contains all of the project files, and stores each file's revision history.
- A **commit** is an individual change to a file (or set of files). It saves changes on local repository.
- To **push** means to send your committed changes to a remote repository on GitHub.com.
- **Pull** refers to when you are fetching in changes (from GitHub) and merging them.
- A **fork** is a personal copy of another user's repository that lives on your account. Forks allow you to freely make changes to a project without affecting the original upstream repository.
- A **clone** is a copy of a repository that lives on your computer.
- A **branch** is a parallel version of a repository. It is contained within the repository, but does not affect the main branch allowing you to work freely without disrupting the "live" version.
- A **merge** takes the changes from one branch (in the same repository or from a fork), and applies them into another.

Source: `https://docs.github.com/en/get-started/quickstart/github-glossary`

Git & GitHub glossary

- **Merge conflicts** happen when people make different changes to the same line of the same file, or when one person edits a file and another person deletes the same file. The merge conflict must be resolved before you can merge the branches.

Git & GitHub glossary

- **Merge conflicts** happen when people make different changes to the same line of the same file, or when one person edits a file and another person deletes the same file. The merge conflict must be resolved before you can merge the branches.
- **Pull requests** are proposed changes to a repository submitted by a user and accepted or rejected by a repository's collaborators. Pull requests have their own discussion forum.

Git & GitHub glossary

- **Merge conflicts** happen when people make different changes to the same line of the same file, or when one person edits a file and another person deletes the same file. The merge conflict must be resolved before you can merge the branches.
- **Pull requests** are proposed changes to a repository submitted by a user and accepted or rejected by a repository's collaborators. Pull requests have their own discussion forum.
- **Issues** are suggested improvements, tasks or questions related to the repository. Issues can be created by anyone (for public repositories), and are moderated by repository collaborators. Each issue contains its own discussion thread.

Source: `https://docs.github.com/en/get-started/quickstart/github-glossary`

The hardest part: Setting up!

- Register for a free GitHub account (`https://github.com/`) and apply for the student benefits by signing up for the global campus (all free): `https://education.github.com/students`. A useful guide on creating your GitHub account can be found here: `https://happygitwithr.com/github-acct`
- Install Git: Follow one of these guides: `https://happygitwithr.com/install-git` or `https://github.com/git-guides/install-git`

The hardest part: Setting up!

To do this in the easiest way we can use the R packages `usethis` and `gitcreds`.

1. Check setting - if it's ready skip the following steps:

```
usethis::git_sitrep()
```

2. Set your configuration (use your GitHub username and the email)

```
usethis::use_git_config(
    user.name = "igollini", # Your username
    user.email = "isabella.gollini@ucd.ie") # Your email
```

3. Get a Personal Authorisation Token from GitHub

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

4. Copy the token and give the token as the password when requested:

```
gitcreds::gitcreds_set()
```

5. Restart R and check again the setting (step 1).

Your Turn!

1. Add git as version control system in your `infant` project
2. Commit your changes with RStudio a few times. Look at the diff between files.
3. Create a public repository on GitHub and connect it to your local repo (see `https://happygitwithr.com/existing-github-last`).
4. Modify a file both on GitHub and on your local repo without pulling/pushing.
5. Now pull from RStudio e try to resolve the conflict between copies (see `https://happygitwithr.com/git-branches.html?q=conflicts#dealing-with-conflicts`).
6. Clone the repository `infant_tutor` from my GitHub, and ask other participant for their account and start to collaborate! (see `https://happygitwithr.com/existing-github-first`).

Some extra tips: LaTeX + R

- `knitr` is an R package that allows you to process LaTeX files that contain code chunks.
- The code chunks can be in one of two formats:
  1. either a format introduced by knitr (with extension `.Rtex`)
  2. traditional Sweave format (with extension `.Rnw`).
- Overleaf allows you to use either `.Rtex` or `.Rnw` style code chunk formatting within a LaTeX document.
  - Warning: Strangely, regardless of which format you use, you need to have your document name end with `.Rtex`.
- see: `https://www.overleaf.com/learn/latex/Knitr`

Some extra tips: Write your Own R package!

- Official guide:
  `https://cran.r-project.org/doc/manuals/r-release/R-exts.html`
- Guide on posit/RStudio website: `https://support.posit.co/hc/en-us/articles/200486488-Developing-Packages-with-RStudio`
- Before start you have to check that you have:
  - GNU software development tools including a C/C++ compiler; and
  - LaTeX for building R manuals and vignettes.
  - more details are at: `https://support.posit.co/hc/en-us/articles/200486498-Package-Development-Prerequisites`

Some extra tips: Write your Own R package!

- Official guide: `https://cran.r-project.org/doc/manuals/r-release/R-exts.html`
- Guide on posit/RStudio website: `https://support.posit.co/hc/en-us/articles/200486488-Developing-Packages-with-RStudio`
- Before start you have to check that you have:
  - GNU software development tools including a C/C++ compiler; and
  - LaTeX for building R manuals and vignettes.
  - more details are at: `https://support.posit.co/hc/en-us/articles/200486498-Package-Development-Prerequisites`
- I suggest you to use the `devtools` package which contains **Tools to Make Developing R Packages Easier** `https://devtools.r-lib.org/`
- Great book, available online for Free: **R Packages** by Hadley Wickham and Jennifer Bryan `https://r-pkgs.org/`

Learning more/getting support

- RStudio cheatsheets (`dplyr`, `ggplot2`)
  `https://www.rstudio.com/resources/cheatsheets/`
- R for Data Science (data handling, basic programming and modelling, Quarto)
  `https://r4ds.hadley.nz/`
- RStudio Community (friendly forum focused on "tidyverse" packages, including `dplyr`, `ggplot2`) `https://community.rstudio.com/`

Learning more/getting support

- Task views `https://cran.r-project.org/web/views/` provide an overview of R's support for different topics, e.g. Bayesian inference, survival analysis, ...
- Package vignettes, many books, e.g. on `https://bookdown.org/`.

Journals

- The R Journal `https://journal.r-project.org/`
- Journal of Statistical Software `https://www.jstatsoft.org/`
- Several other journals have special sessions or special issues dedicated to software development.