Data Management and Manipulation

Lecture 02.2: Data Import

Lauren Sullivan

Module: Data Management, Visualization & Reproducibility

Readings

Required for class:

- NA

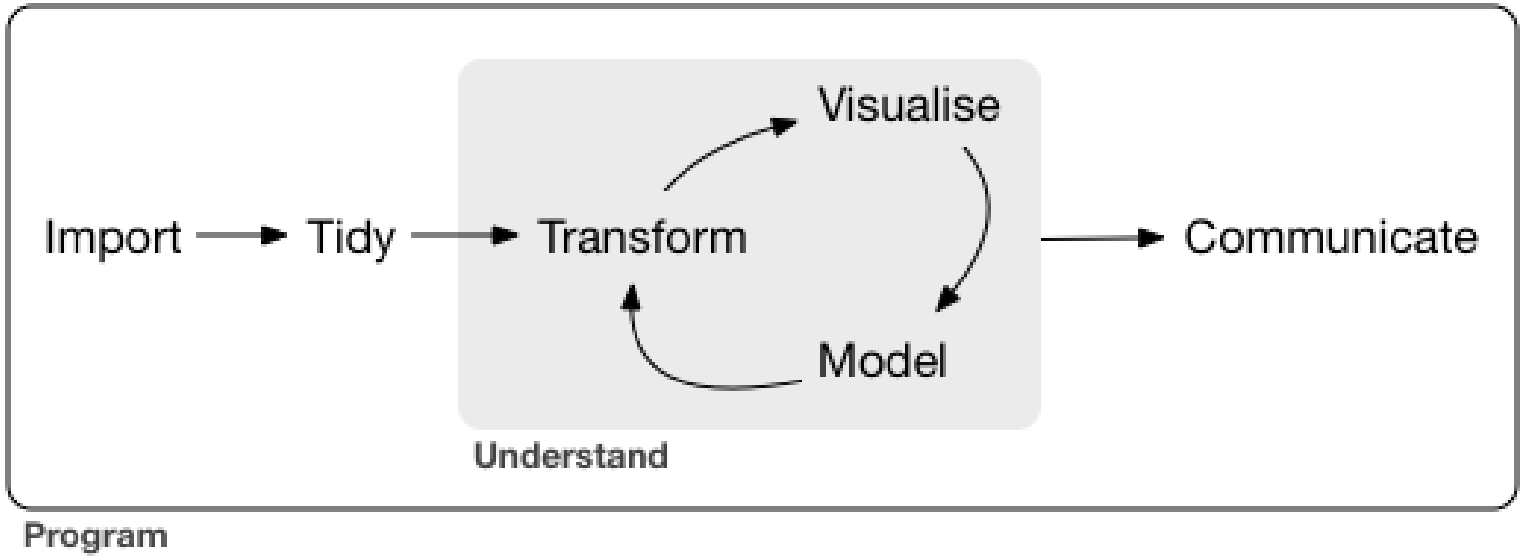
Optional:

- Tidyverse
- Grolemund & Wickham (2017) *R for Data Science. - Data Import*
- Data Import::Cheat Sheet - for readr and tidyr

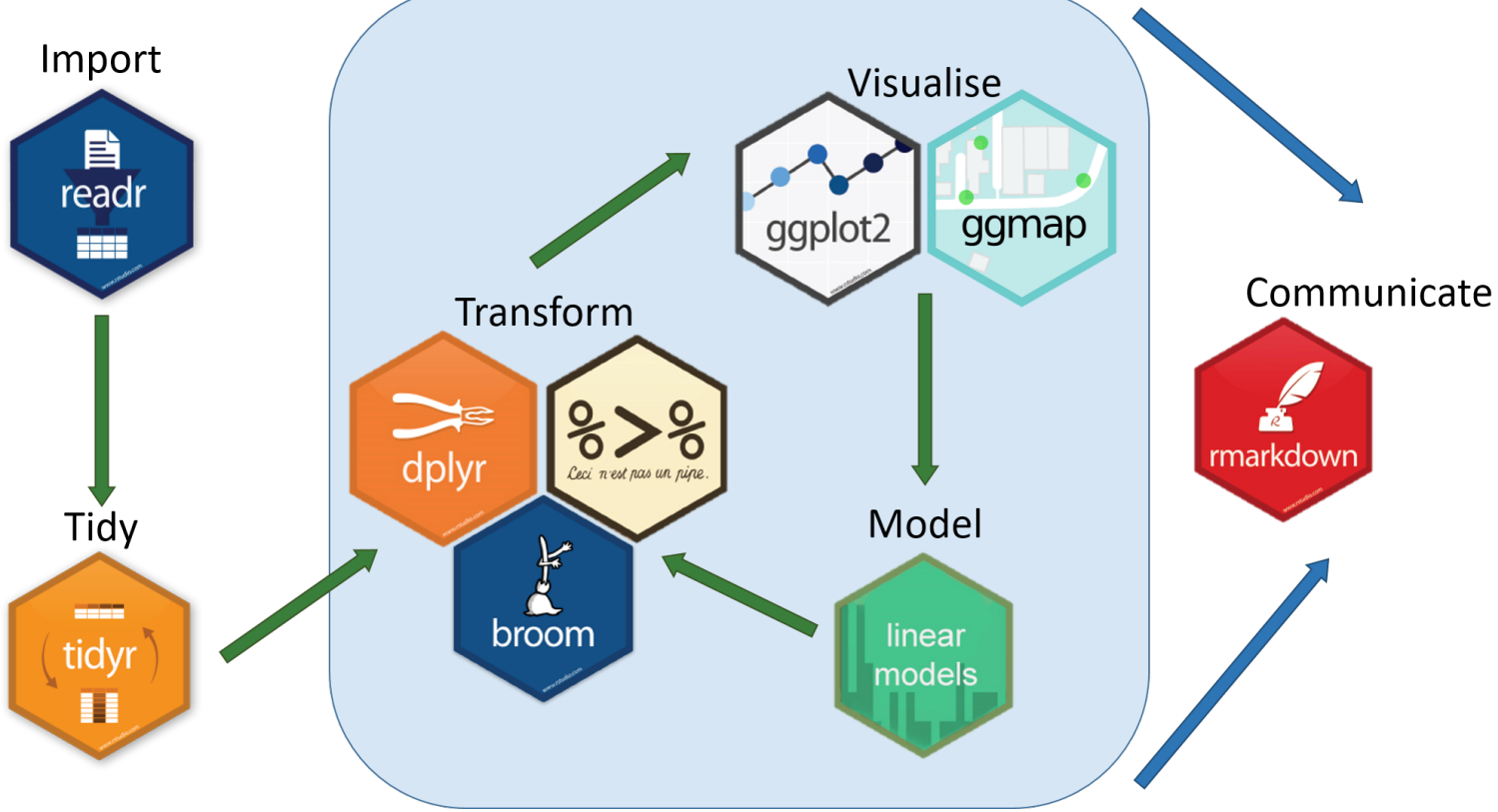
Tidyverse

- A collection of R packages that are all based on the same underlying design philosophy, grammer and data structure.
  - Core Tidyverse packages: ggplot2, dplyr, tidyr, readr, purr, tibble, stringr, forcats.
  - `install.packages("tidyverse")`

Remember our Project Pipeline?



Tidyverse in the Project Pipeline



Important Note About the Tidyverse

You may notice when you load the tidyverse you get a message that says that some libraries overwrite functions in base R.

- For example, dplyr overwrites `filter()` and `lag()`.
  - To use the base functions, you will need to use their full names: `stats::filter()` and `stats::lag()`.

Data Import Using readr

Readr is how the tidyverse wants you to import files.

- `read_csv()` reads in comma delimited files
- `read_csv2()` reads in semicolon separated files
- `read_tsv()` reads in tab delimited files
- `read_delim()` reads in any delimited files

See this page for more information on how to read other file types.

read_csv()

To import data, you can either:

1. Use `read_csv()` with a direct path to the data file.

```
dat <- read_csv("../data/dispersalrate_data.csv")
print(dat, n = 4)
```

```
## # A tibble: 1,343 x 4
##   plot_no subplot species notes
##     <dbl>   <dbl> <chr>   <chr>
## 1       1       6 poapra  <NA>
## 2       1       6 solcan  <NA>
## 3       1       6 sornut  seedling
## 4       1       6 solspe  <NA>
## # ... with 1,339 more rows
```

read_csv()

Or:

2. Use `setwd()` to set your working directory folder, and then use `read_csv()` to read in any .csv file by its name (not path name) that is in that folder.
   - This allows for cleaner data management.

```
#setwd("~/Desktop/DispRate_StCroix/data/cleaned data")
dat <- read_csv("../data/dispersalrate_data.csv")
print(dat, n = 4)
```

```
## # A tibble: 1,343 x 4
##   plot_no subplot species notes
##     <dbl>   <dbl> <chr>   <chr>
## 1       1       6 poapra  <NA>
## 2       1       6 solcan  <NA>
## 3       1       6 sornut  seedling
## 4       1       6 solspe  <NA>
## # ... with 1,339 more rows
```

Syntax Note

A quick reminder that it is always best to save objects in R with `<-` instead of `=`. The `<-` is unidirectional, but the `=` is birectional, and this can cause problems when adding mathematical equations to your work in R, or can be confusing when thinking about logicals for TRUE/FALSE in functions, or boolean operators.

Tibbles

You will see that `read_csv()` reads in files as tibbles. The tidyverse uses tibbles instead of data.frames. They are essentially the same thing, but a tibble is more modern and works better with the data structure of the tidyverse. For more detailed info see Tibbles, or `vignette("tibble")`.

A few notes on tibbles

- You can coerce a data.frame into a tibble with `as.tibble(data.frame)`
- Tibbles never change the type of an input (e.g. no conversion of strings to factors), it never changes the name of a variable, and it never creates row names.
- You can transpose a tibble with `tribble()`

Tibble vs. Data.frame

Two big differences between tibbles and data.frames

1. Tibbles only print 10 rows, so you can work with large datsets easily, and columns display their type (e.g. “chr”, “int”).
   - You can specify what parts of the tibble you want printed with `print(tibble, n=x, width=x)` where `n` is the number of rows, and `width` is the width to display.
   - You can look at the start of a tibble with `head(<tibble_name>)`, or the end of a tibble with `tail(<tibble_name>)`.
   - See a full scrollable version of the data with `View()`.
2. Subsetting is done by name with `$`, or by name or position with `[[]]`.

Subsetting a Tibble

```
df
```

```
## # A tibble: 5 x 2
##         x       y
##     <dbl>   <dbl>
## 1 0.838  -0.917
## 2 0.0728  0.848
## 3 0.111  -0.353
## 4 0.0934 -1.08
## 5 0.962  -0.0480
```

```
#Extracts by name
df$x
df[["x"]]

#Extracts by position
df[[1]]
```

Subsetting a Tibble

Note, when using the pipe %>%, which we will talk about in a bit, you need to use a special placeholder . when subsetting:

```
#Here the . stands in for the tibble named "df"
df %>% .$x
```

```
## [1] 0.83807265 0.07276818 0.11074717 0.09344685 0.96225547
```

```
df %>% .[["x"]]
```

```
## [1] 0.83807265 0.07276818 0.11074717 0.09344685 0.96225547
```

Parsing Data

The `parse_*()` function is a building block of readr. We won't go into too much detail here, but if this seems useful to your work, you can read about it here.

- Parsing can be helpful for:
  - splitting out numbers or logicals (`parse_logical()`, `parse_integer()`)
  - splitting up strings of characters (`parse_character()`)
  - splitting out factors based on predetermined levels (`parse_factor()`)
  - splitting out dates and times (`parse_datetime()`, `parse_date()`, `parse_time()`)

Writing a File

You save a dataset as a .csv with `write_csv()`. First specify the dataset you want to save, then write the name of the new file.

```
head(dat)
```

```
## # A tibble: 6 x 4
##   plot_no subplot species notes
##     <dbl>   <dbl> <chr>   <chr>
## 1       1       6 poapra  <NA>
## 2       1       6 solcan  <NA>
## 3       1       6 sornut  seedling
## 4       1       6 solspe  <NA>
## 5       1       6 schsco  <NA>
## 6       1       6 andger  <NA>
```

```
write_csv(dat, "dat-2.csv")
read_csv("dat-2.csv")
```

```
## Rows: 1343 Columns: 4
## -- Column specification --------------------------------------------------
## Delimiter: ","
## chr (2): species, notes
## dbl (2): plot_no, subplot
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this mes
```

Writing a File

Note that some times the `write_csv()` function will not save the correct type for your data columns (e.g. dates might be stored as characters). Thus you will need to re-specify that column type.

- Alternatively, you could save your files as .rds files, which is R's customary binary format and use `write_rds()`, and `read_rds()`.