Data wrangling, visualization, and mapping in R

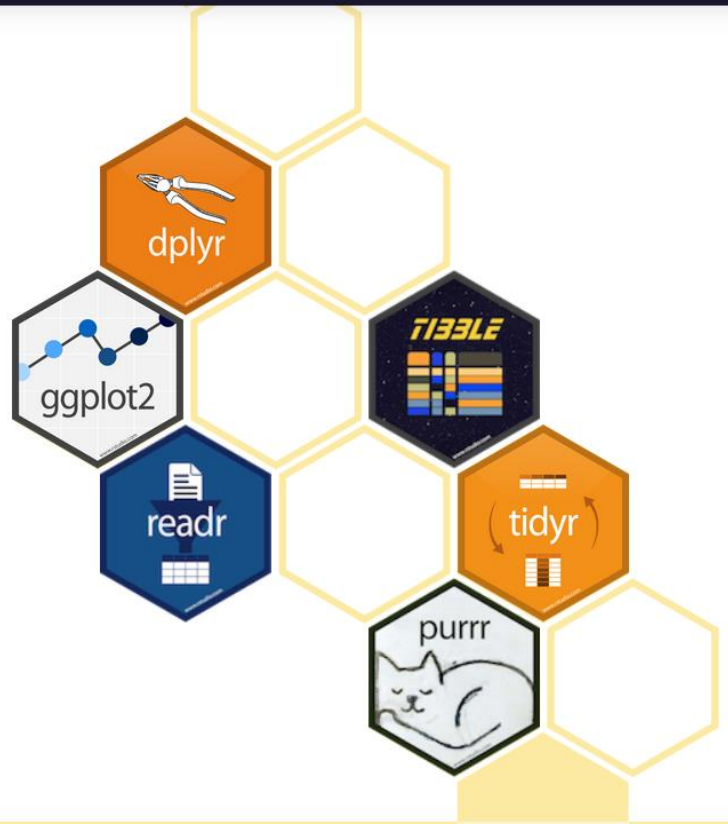
Student Conference on Conservation Science

Brad Woodworth (b.woodworth@uq.edu.au)

Felipe Suárez Castro (a.suarezcastro@uq.edu.au)

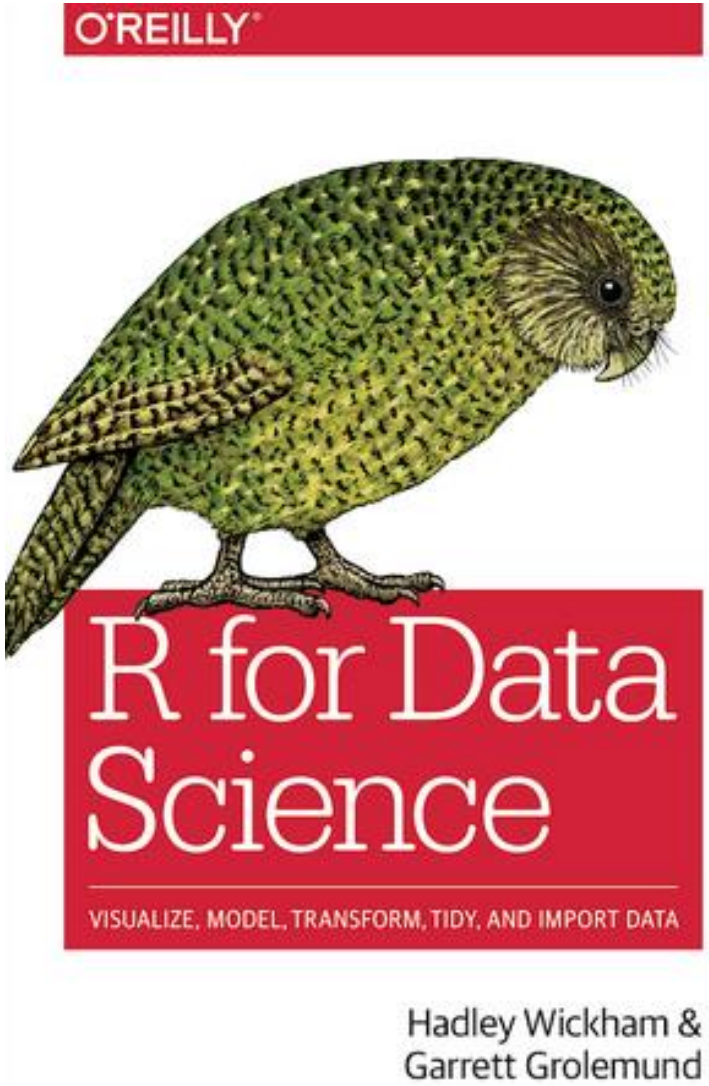
2019-07-10

Tidyverse

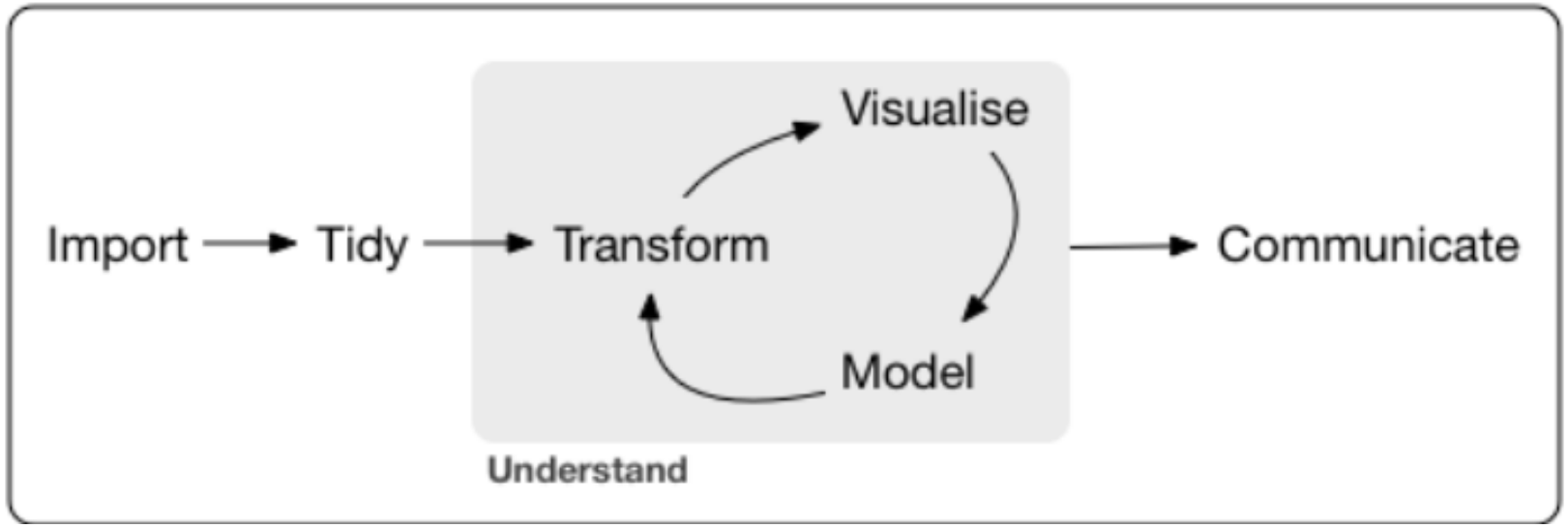


Highly flexible, not just statistics!

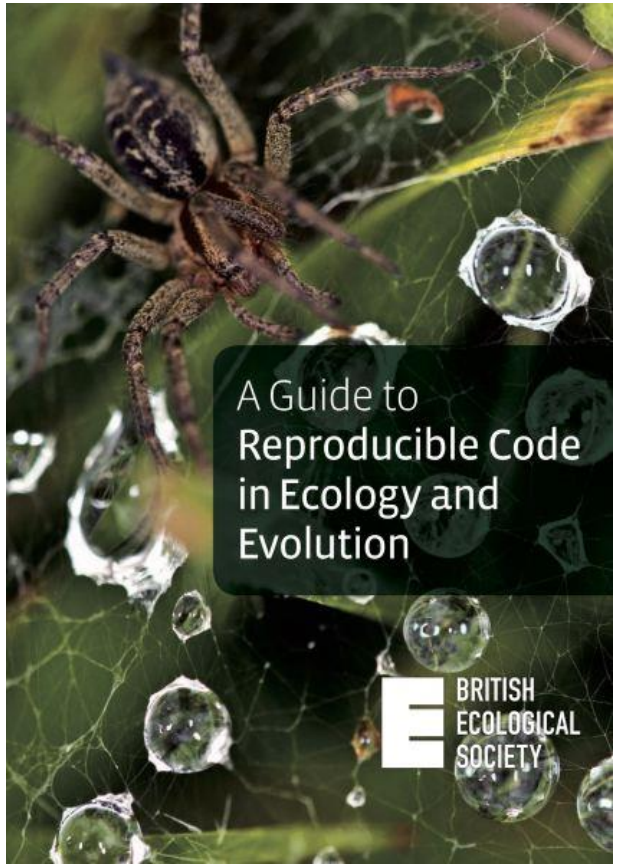
- Data tidying and transformation
  - numbers, text, dates/times, spatial
- Data visualization: exploratory and publication-quality
- Tools to facilitate reproducible workflow
  - R projects, scripts, version control, markdown
- Spatial analysis and mapping tools (GIS)

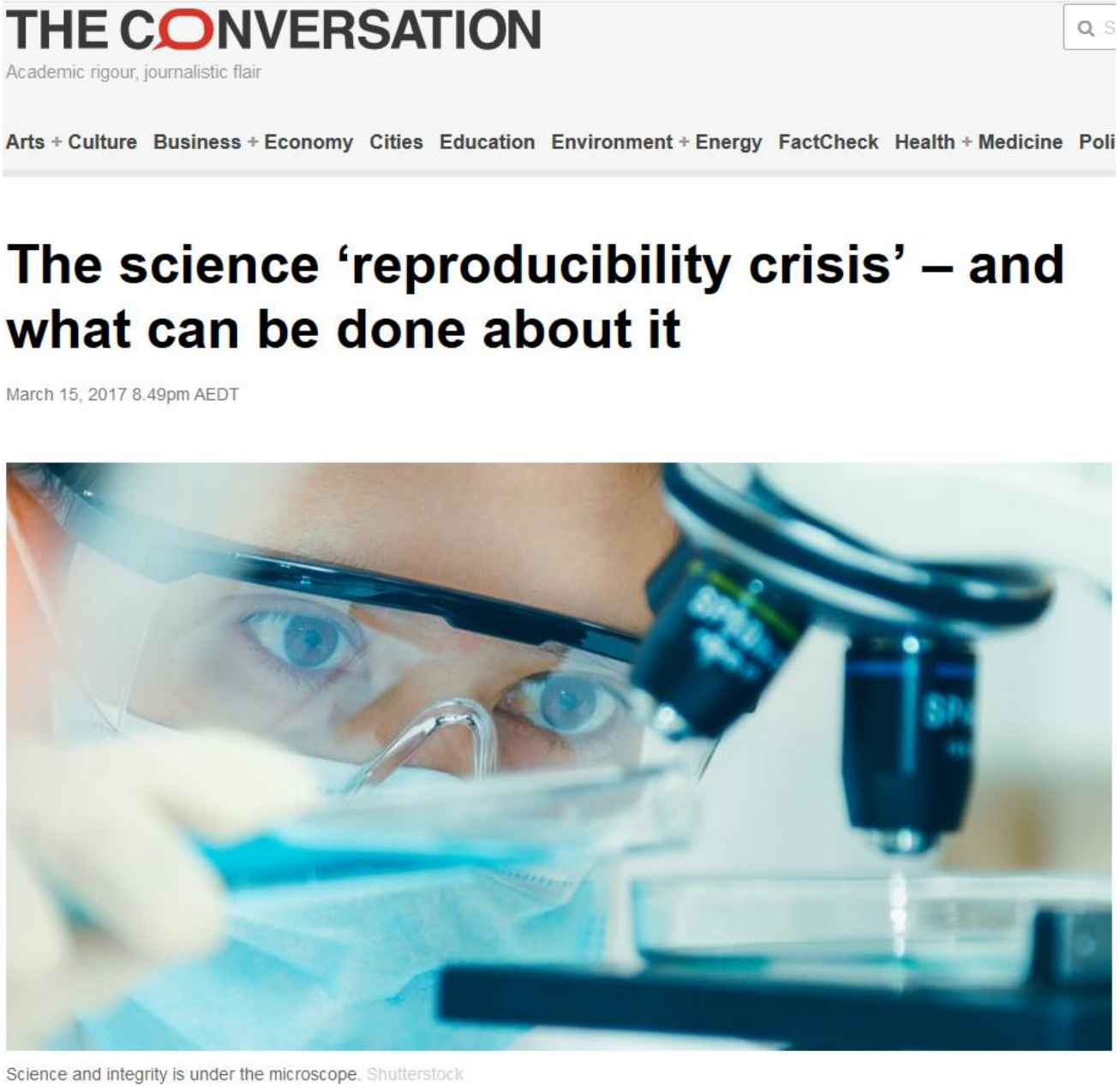


Freely available at: r4ds.had.co.nz



The repeatable, reproducible analysis workflow





THE CONVERSATION

Academic rigour, journalistic flair

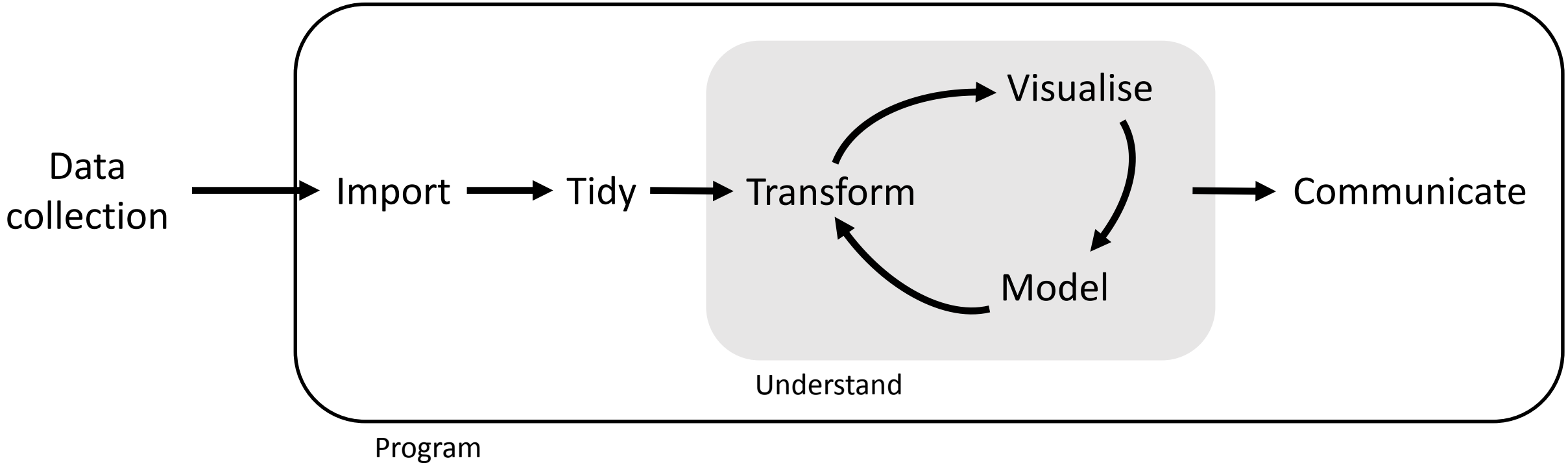
Arts + Culture Business + Economy Cities Education Environment + Energy FactCheck Health + Medicine Poli

The science 'reproducibility crisis' – and what can be done about it

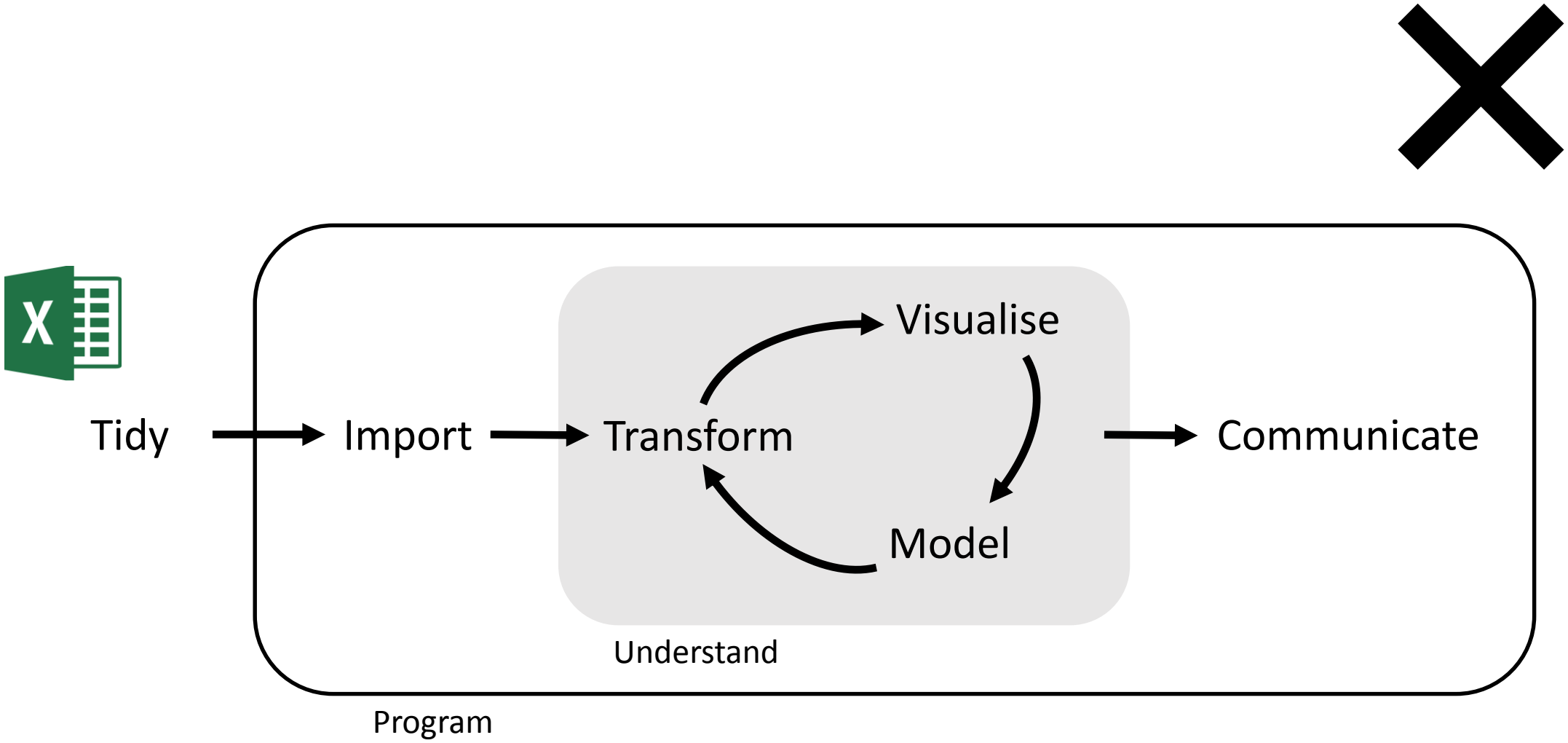
March 15, 2017 8.49pm AEDT

Science and integrity is under the microscope. Shutterstock

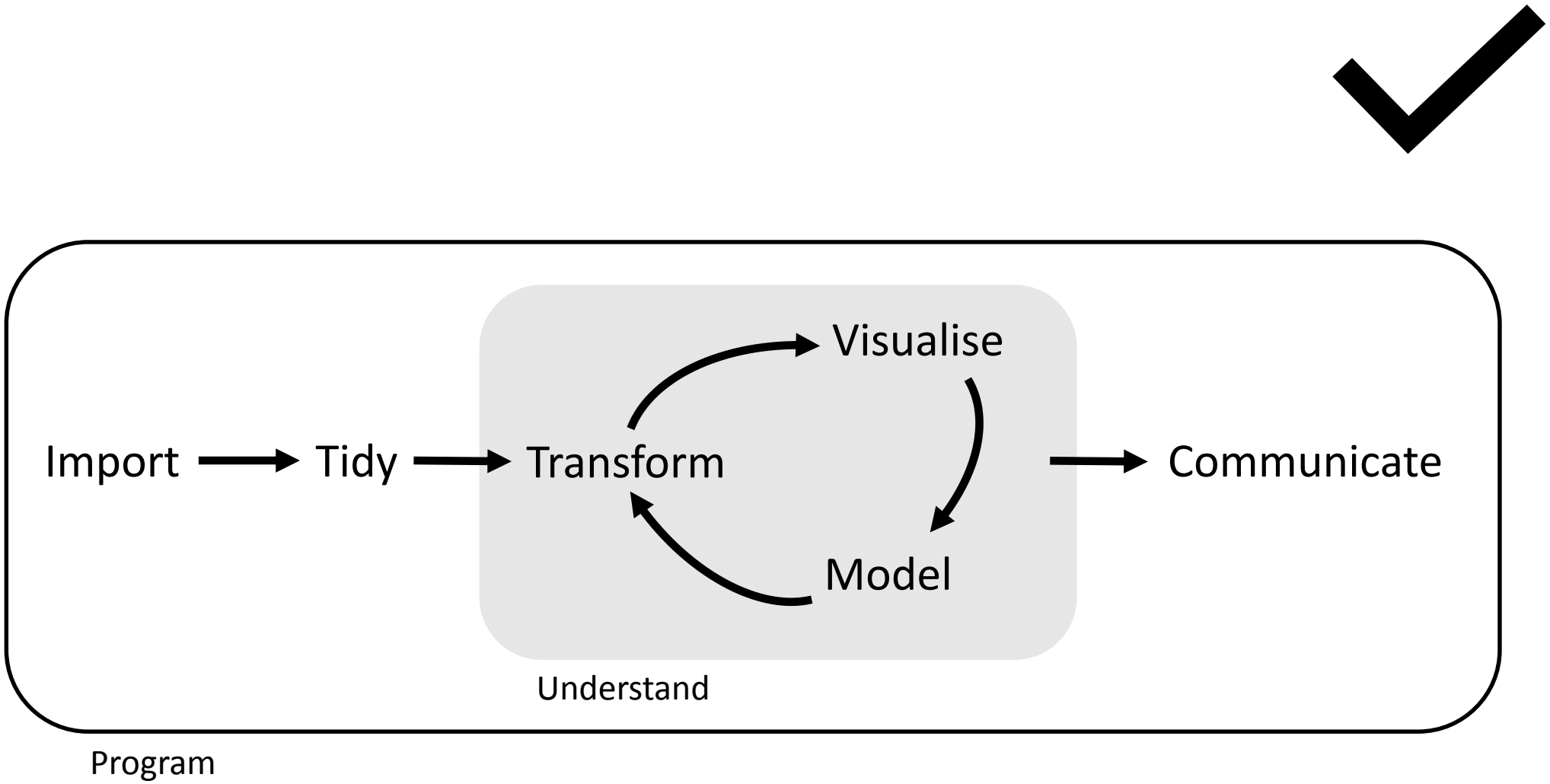
Irreproducible research can originate during data collection, but also during data analysis, especially with ecologists and conservation scientists using increasingly large, varied, and complex datasets



Adapted from R4DS (r4ds.had.co.nz)



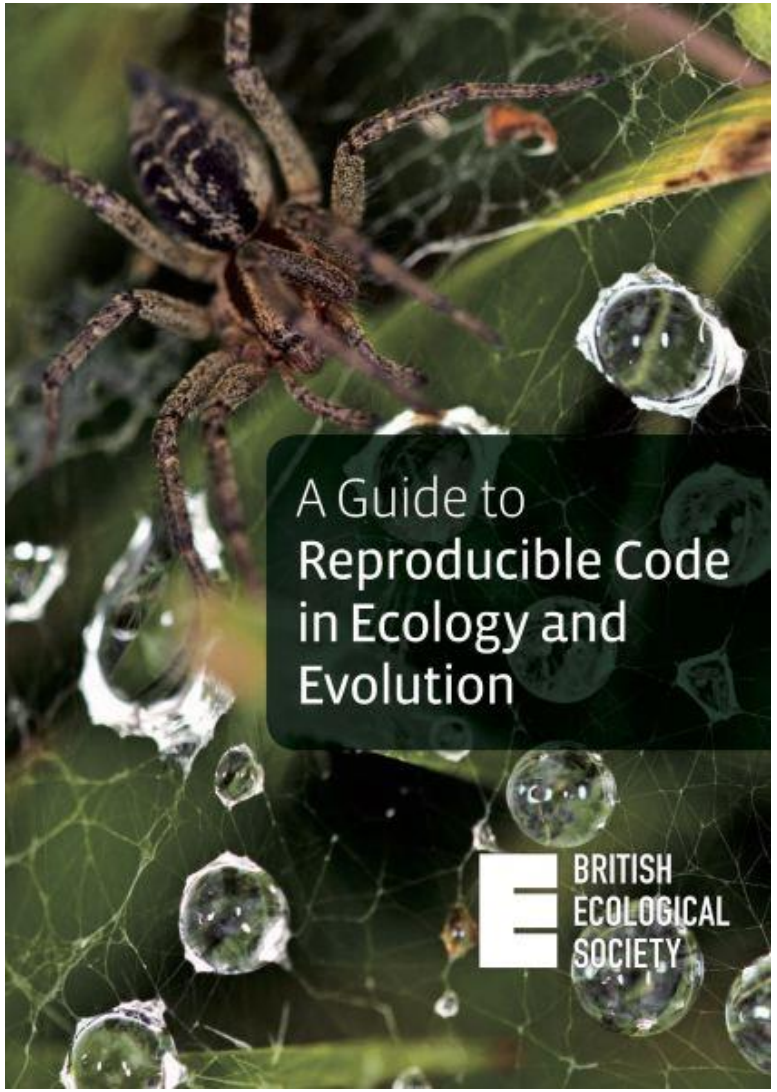
Adapted from R4DS (r4ds.had.co.nz)



Adapted from R4DS (r4ds.had.co.nz)

The repeatable, reproducible analysis workflow

"The fundamental idea behind a robust, reproducible analysis is a clean, repeatable **script-based** workflow (i.e. the sequence of tasks from the start to the end of a project) that links **raw data through to clean data and to final analysis outputs**."

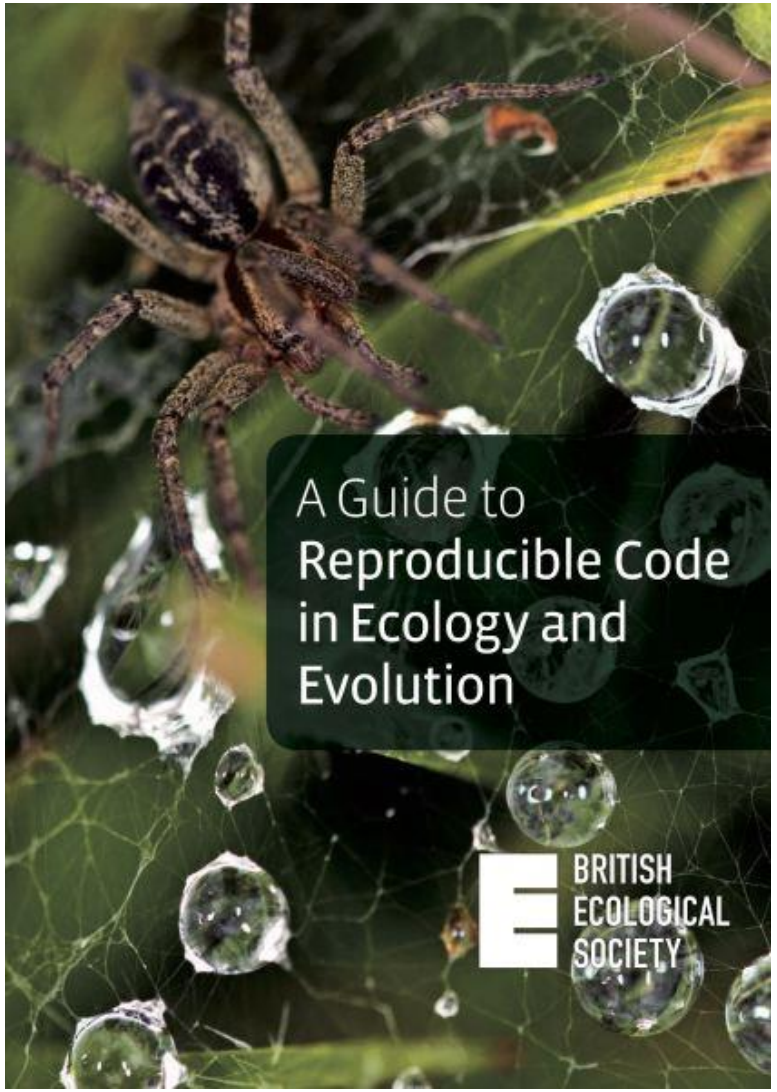


The repeatable, reproducible analysis workflow

Basic elements of a good analysis workflow

- Start your analysis from copies of your raw data.
- Any cleaning, merging, transforming, etc. of data should be done in scripts, not manually.
- Document your code and data as comments in your scripts or by producing separate documentation
- Any intermediary outputs generated by your workflow should be kept separate from raw data.

*** ***Keep all elements of project workflow together*** ***

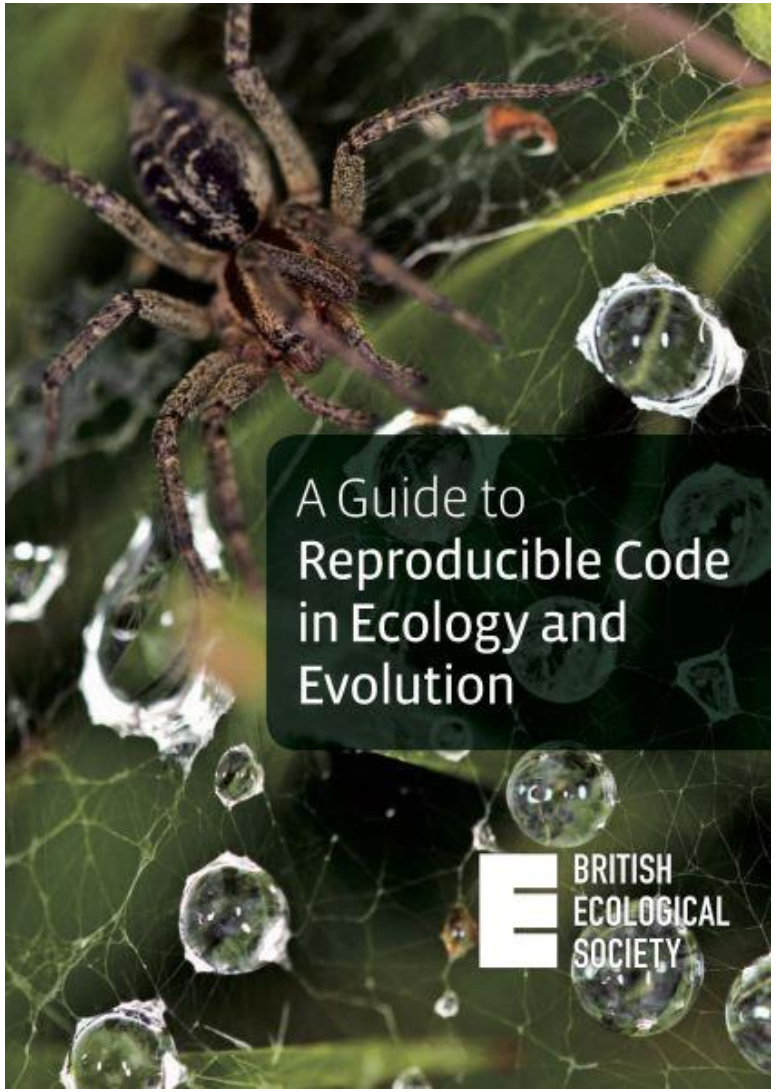


The repeatable, reproducible analysis workflow

Why important?

Most analyses will be re-run many times before they are finished, some times after long absences, so the smoother and more automated the workflow, the easier, faster and more robust the process of repeating it.

Reproducible workflow benefits science and you!



R project set-up

R projects house all elements of your workflow:

- Create a separate one for each data analysis project.
- Keep data files there.
- Keep scripts there; edit them, run them in bits or as a whole.
- Save your outputs (plots and cleaned data) there.
- Only ever use relative paths, not absolute paths

Everything you need is in one place and cleanly separated from all the other projects that you are working on.

Working directory: where your analysis lives

Absolute path:

```
"C:/Users/Brad/Brad's Documents/BW
research/SCCS_R_workshop/data"
```

Relative path:

```
"./data"
```

Why is this helpful?

Tools for wrangling data

Tidyverse

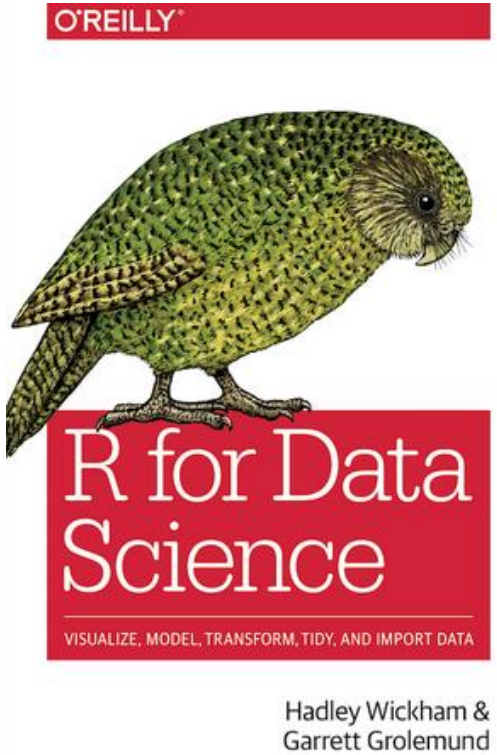
Packages Articles Learn Help Contribute

R packages for data science

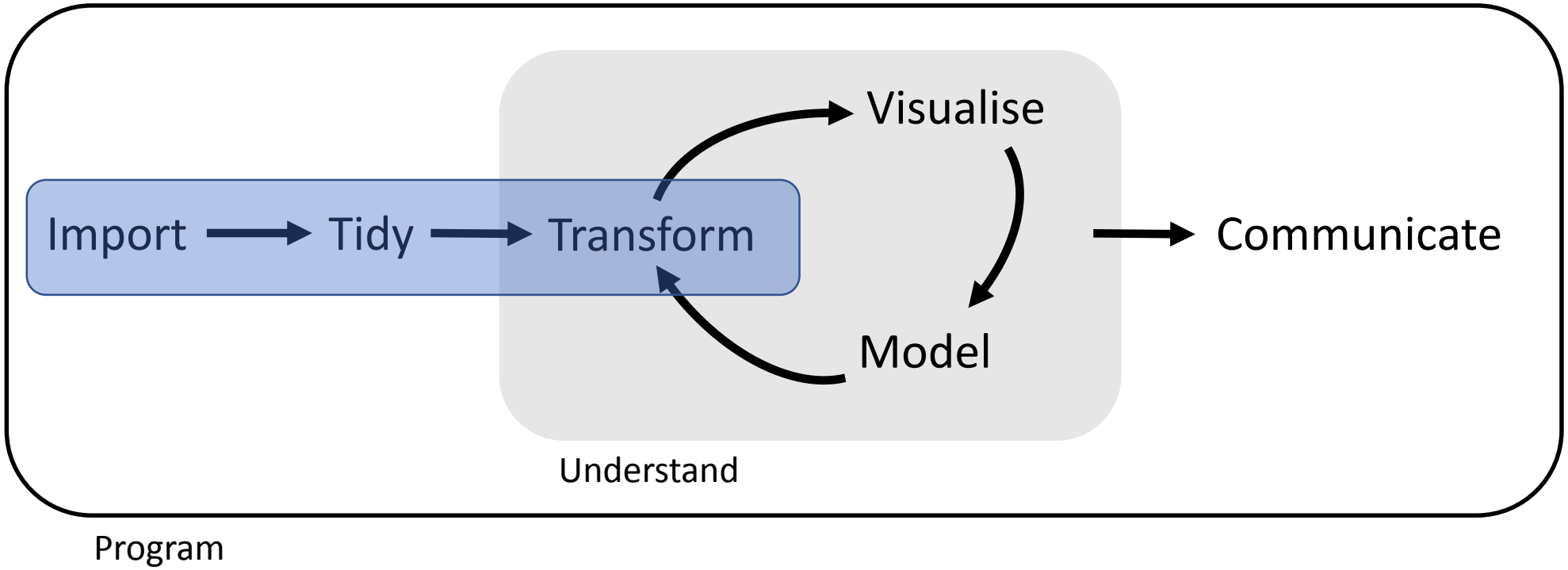
The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```



Data wrangling

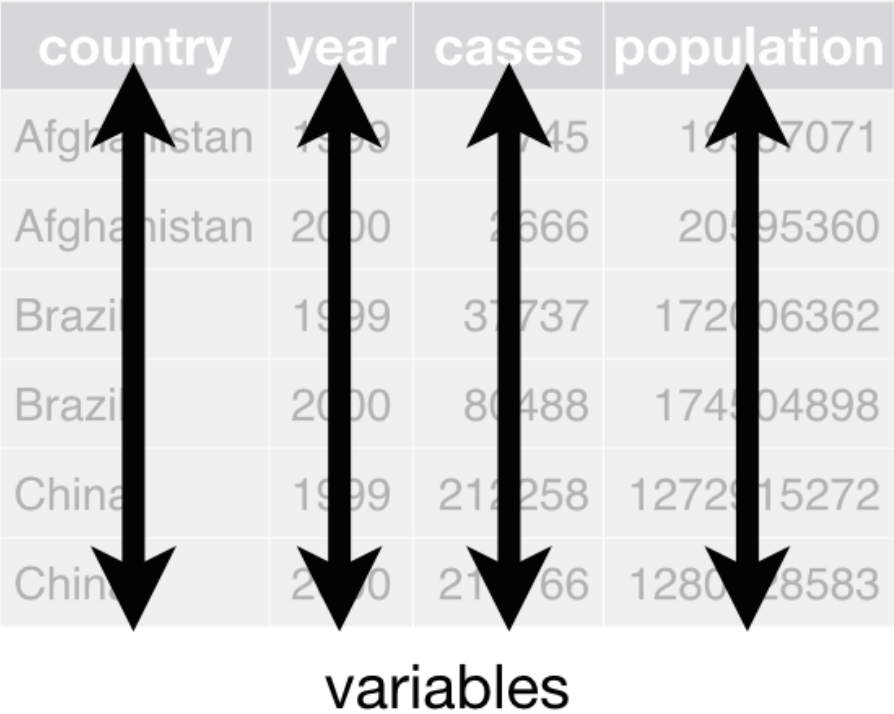


Adapted from R4DS (r4ds.had.co.nz)

What is ‘tidy’ data?

Consistent and flexible way of organising data. In tidy data:

1. **Each variable has its own column.**
2. Each observation has its own row.
3. Each value has its own cell

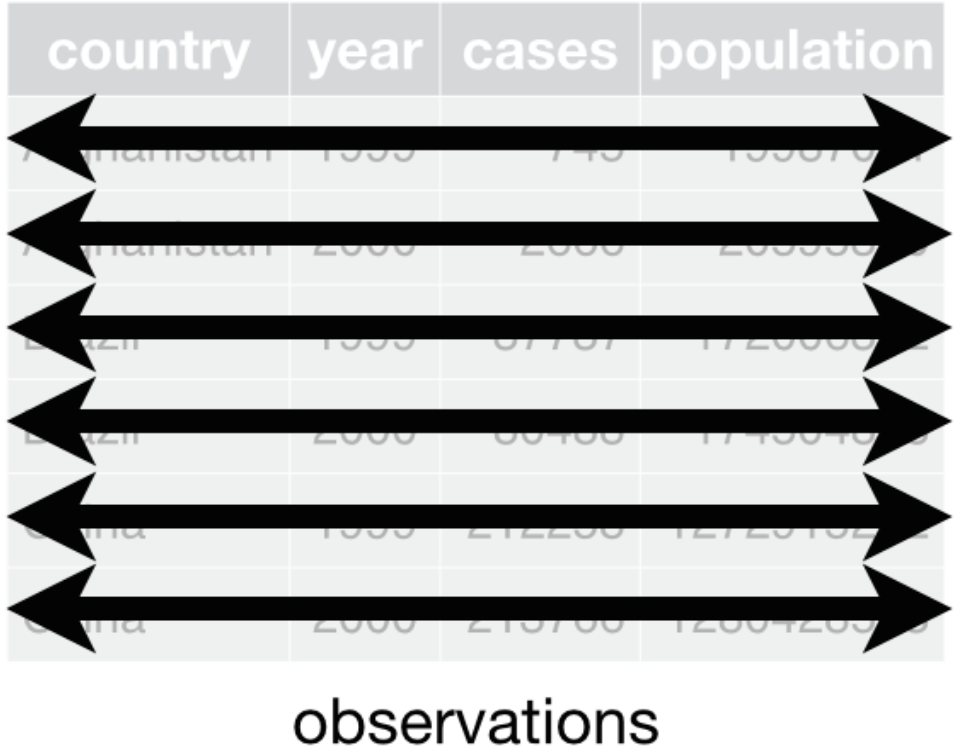


Wickham (2014) Tidy data. *J. Stat. Soft.* | *R for Data Science* (r4ds.had.co.nz)

What is ‘tidy’ data?

Consistent and flexible way of organising data. In tidy data:

1. Each variable has its own column.
2. **Each observation has its own row.**
3. Each value has its own cell



Wickham (2014) Tidy data. *J. Stat. Soft.* | *R for Data Science* (r4ds.had.co.nz)

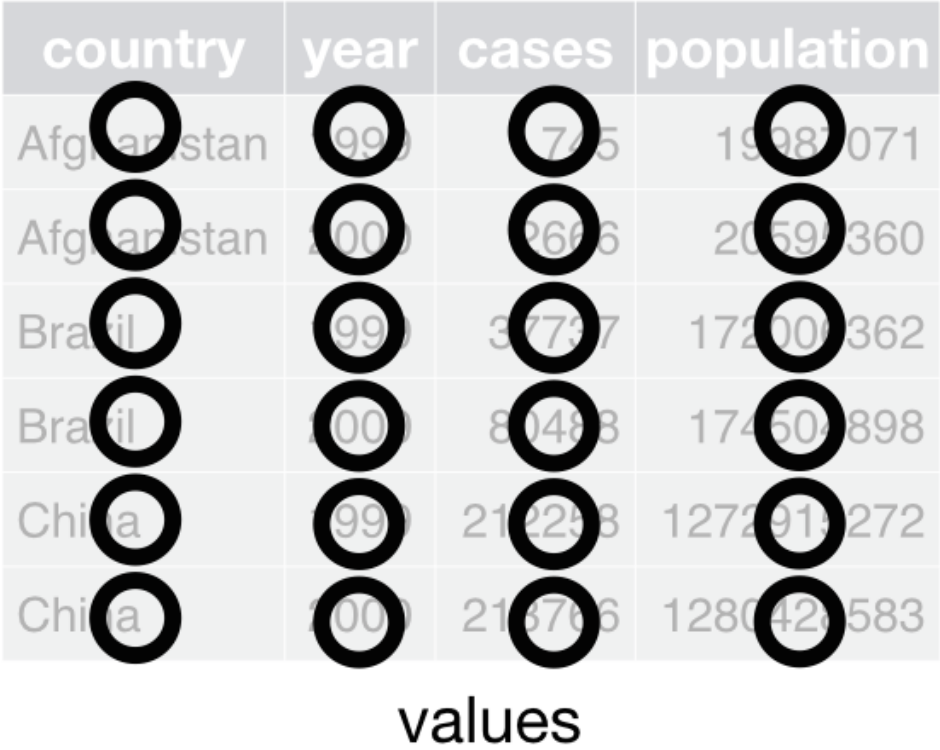
What is ‘tidy’ data?

Consistent and flexible way of organising data. In tidy data:

1. Each variable has its own column.

2. Each observation has its own row.

3. **Each value has its own cell**



Wickham (2014) Tidy data. *J. Stat. Soft.* | *R for Data Science* (r4ds.had.co.nz)

Why tidy data?

- Flexible, easily converted to other formats for specific applications
- Many analysis tools work best with tidy data structure:
  - lm(), glm(), glmer()

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

```
glmer(cases ~ year + (1 | country))
```

Tidy data

Real datasets can, and often do, violate the three rules of tidy data in almost every way imaginable. Occasionally a dataset will be ready to be analyzed immediately, but this is the exception. Common problems with real datasets:

- Column headers are values, not variable names.
- Multiple variables are stored in one column.
- Variables are stored in both rows and columns.
- Multiple types of observational units are stored in the same table.
- A single observational unit is stored in multiple tables.

Wickham (2014) Tidy data *J. Stat. Soft.* | *R for Data Science* (r4ds.had.co.nz)

Tools for tidying data

```
gather(data = table4, key = year, value = cases,
       `1999`, `2000`)
```

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

R for Data Science (r4ds.had.co.nz)

Tools for tidying data

```
spread(data = table2, key = key, value = value)
```

country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

table2

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

R for Data Science (r4ds.had.co.nz)

Data visualization using ggplot2

"R has several systems for making graphs, but ggplot2 is one of the most elegant and most versatile. ggplot2 implements the ***grammar of graphics****, a coherent system for describing and building graphs. With ggplot2, you can do more faster by learning one system and applying it in many places." – R4DS*

A Layered Grammar of Graphics

Hadley WICKHAM

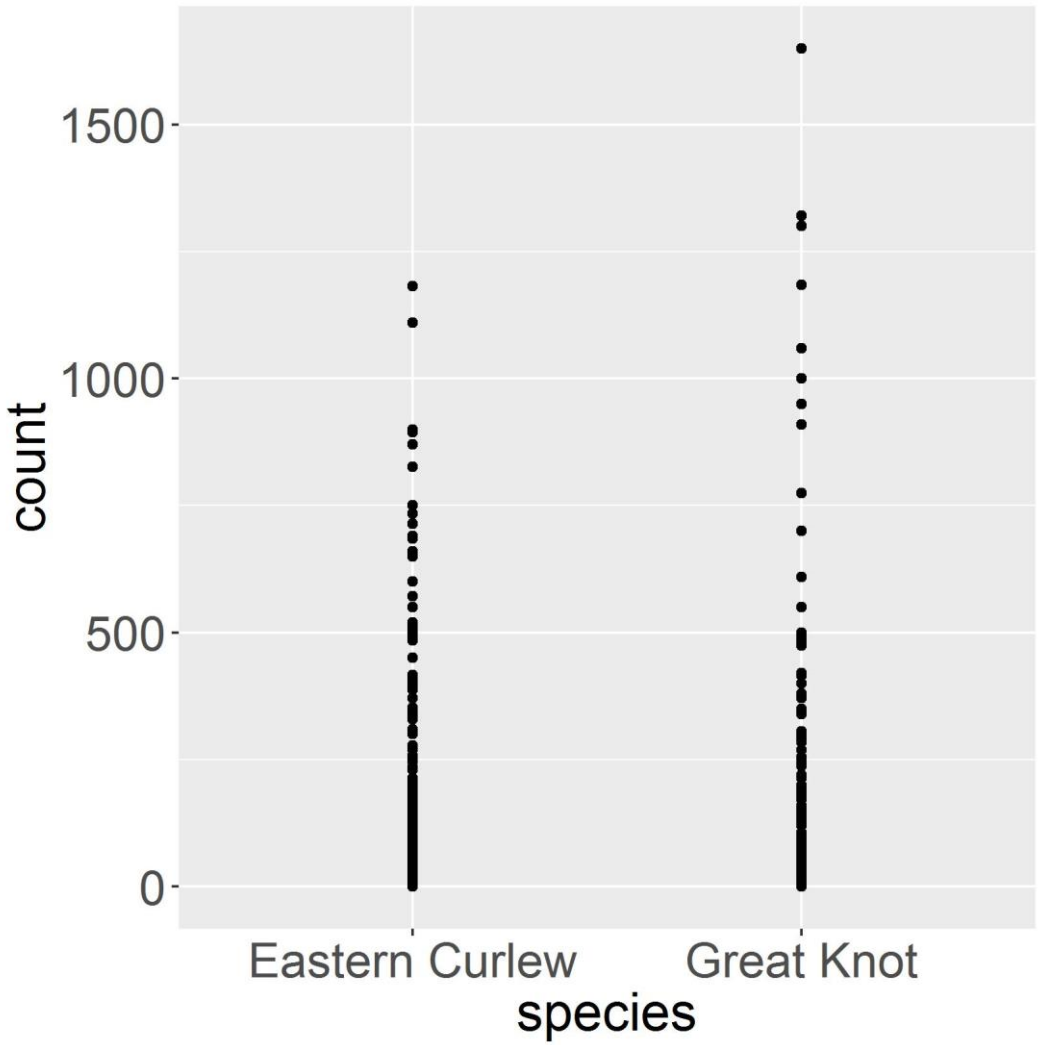
available at:
http://vita.had.co.nz/papers/layered-grammar.pdf.

```
ggplot(data = <DATA>) +
        <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

```
ggplot()
```

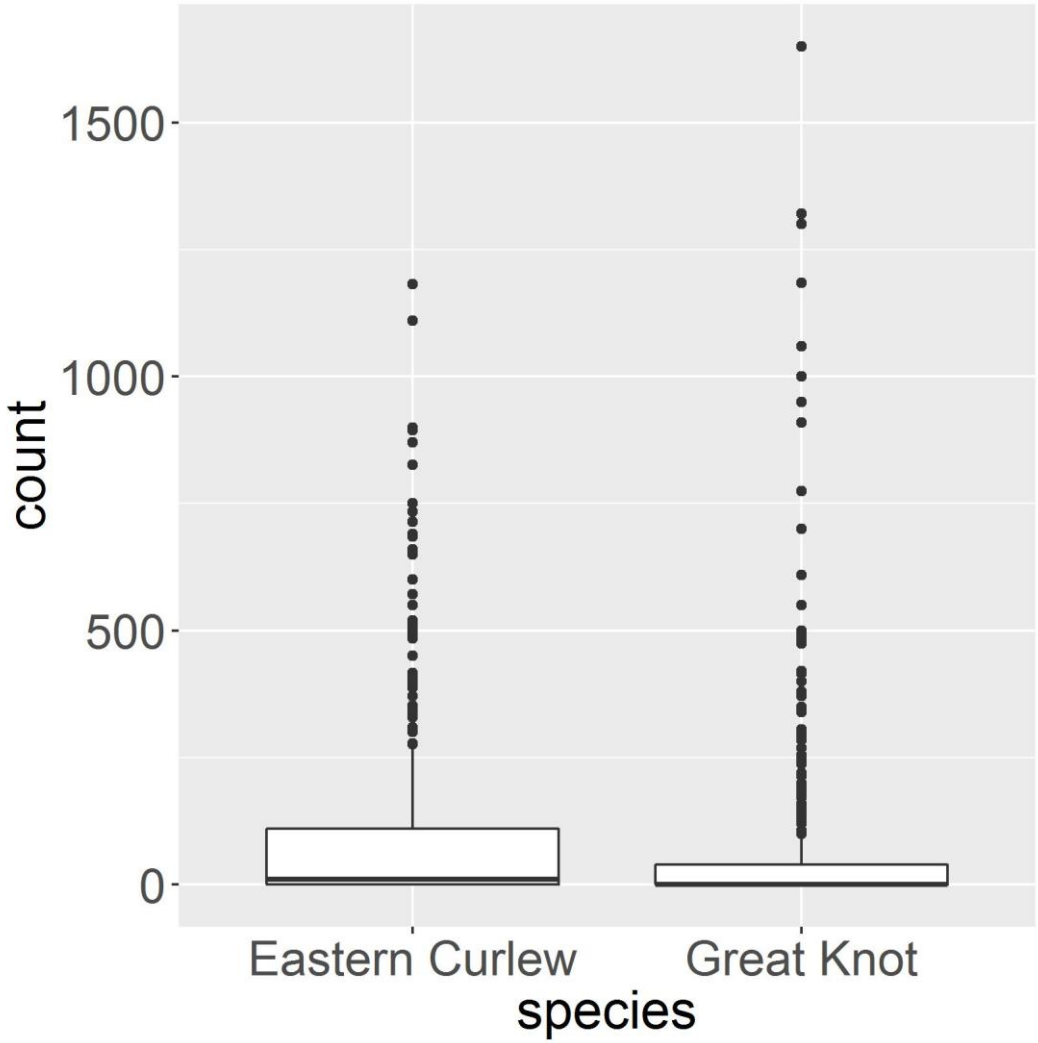
ggplot(data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))

```
# eastern curlew and great knot

ggplot(data = count_data) +
  geom_point(aes(x = species,
                 y = count))
```



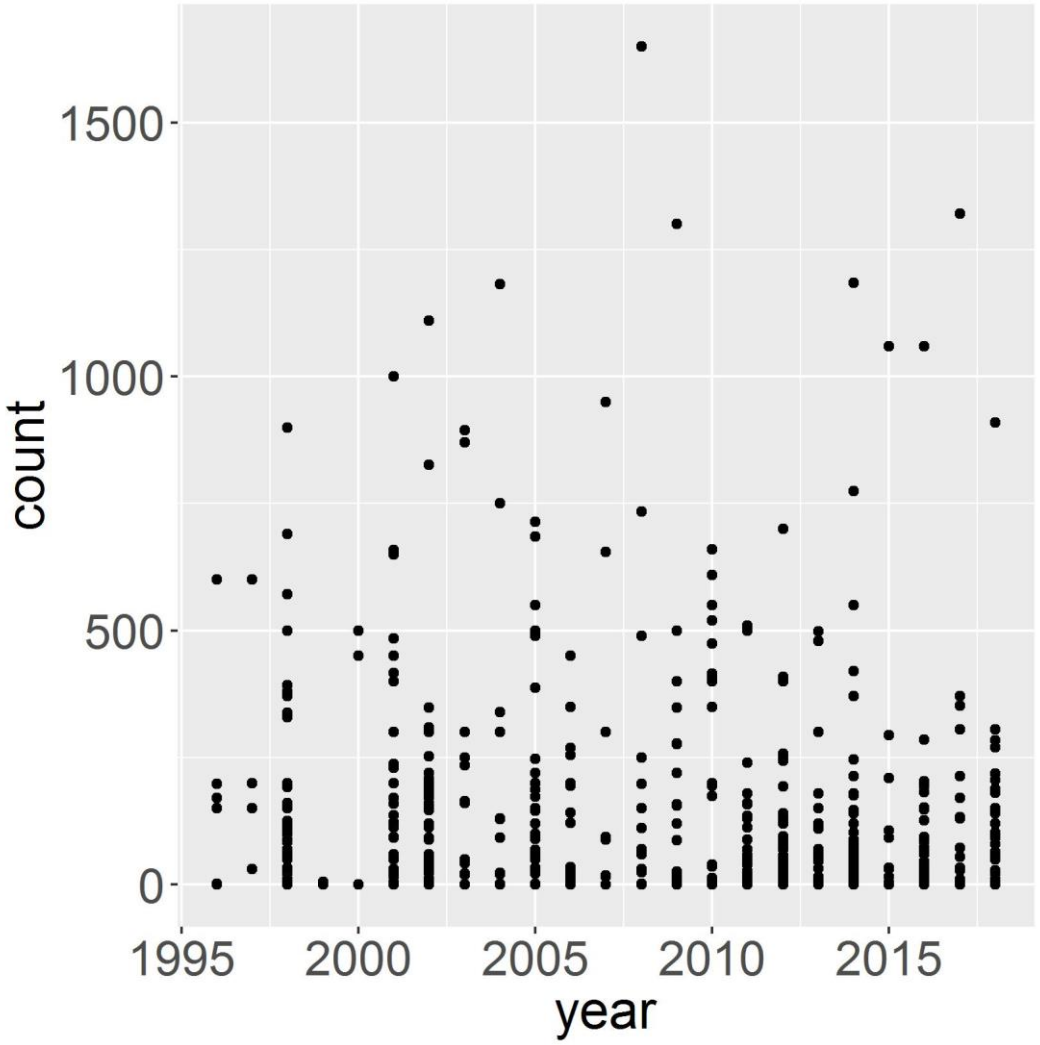
ggplot(data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))

```
# eastern curlew and great knot

ggplot(data = count_data) +
  geom_boxplot(aes(x = species,
                   y = count))
```



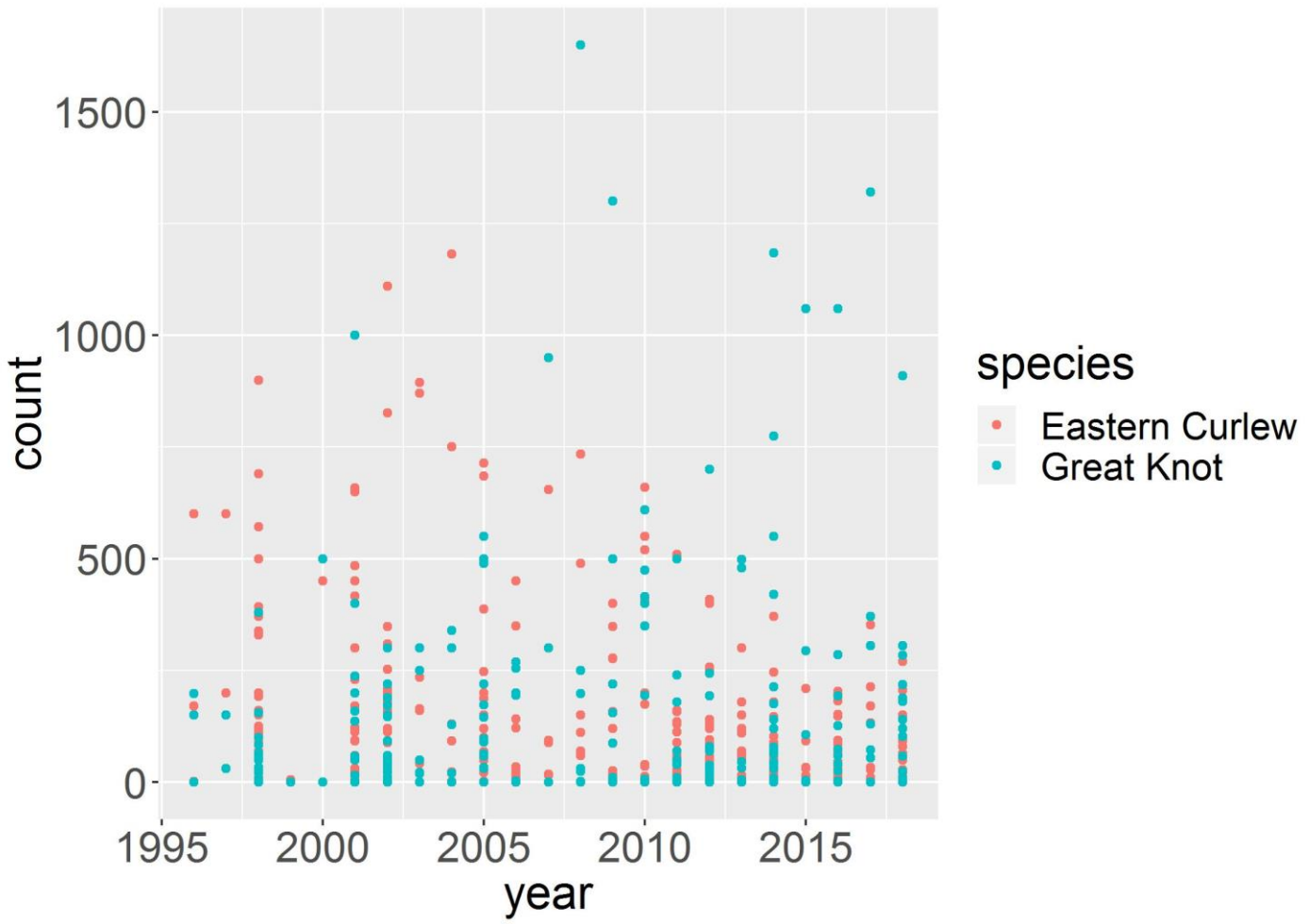
ggplot(data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))

```
# eastern curlew and great knot

ggplot(data = count_data) +
  geom_point(aes(x = year,
                 y = count))
```



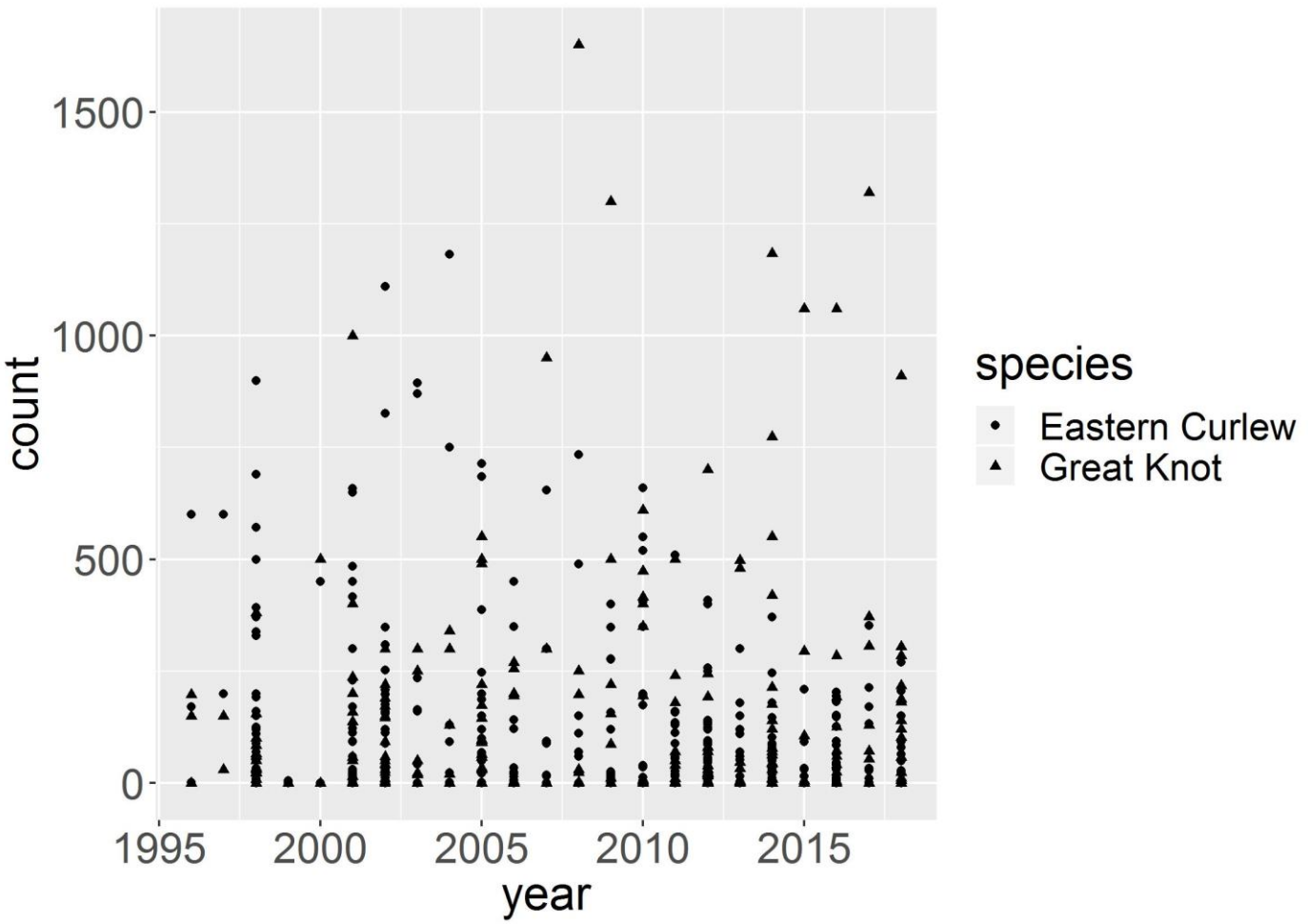
ggplot(data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))

```
# eastern curlew and great knot

ggplot(data = count_data) +
  geom_point(aes(x = year,
                 y = count,
                 colour = species))
```



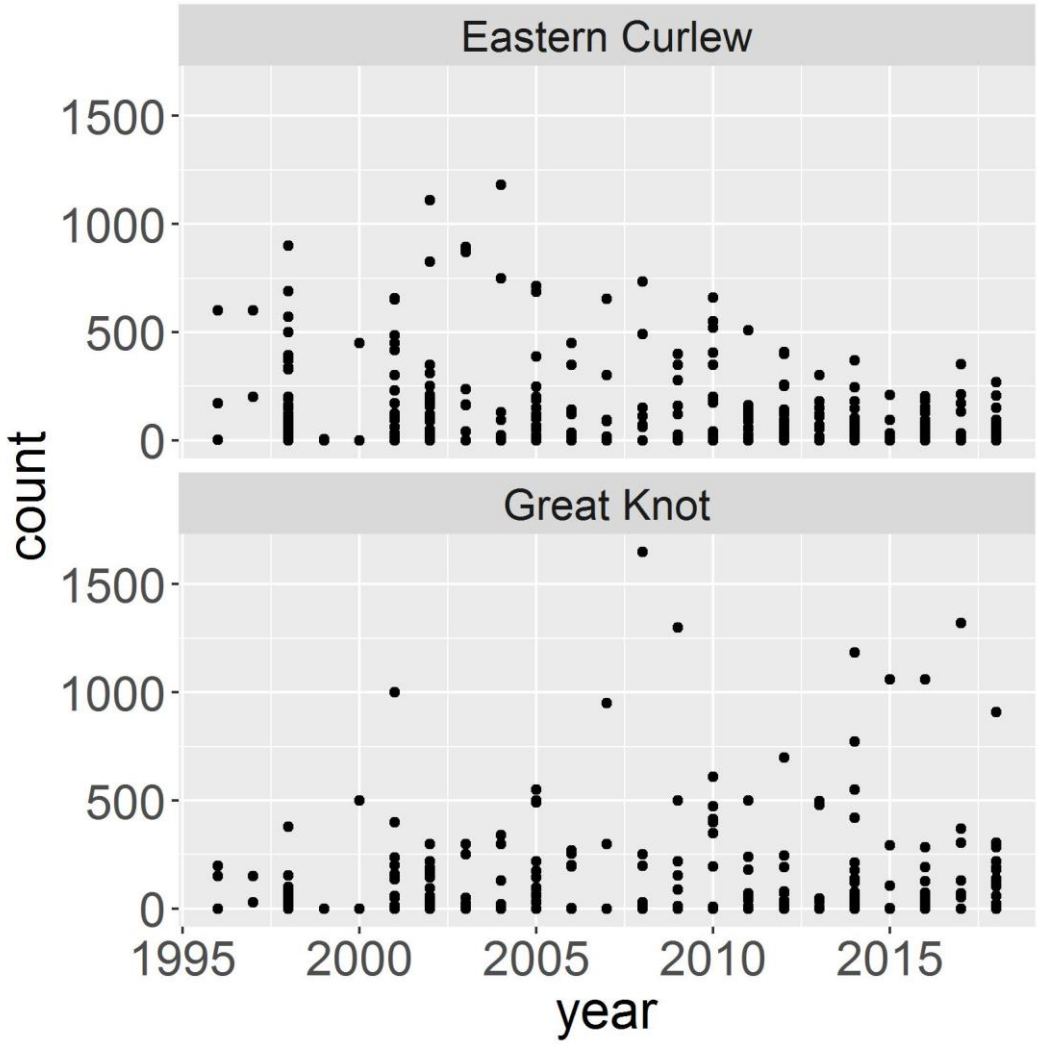
ggplot(data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))

```
# eastern curlew and great knot

ggplot(data = count_data) +
  geom_point(aes(x = year,
                 y = count,
                 shape = species))
```



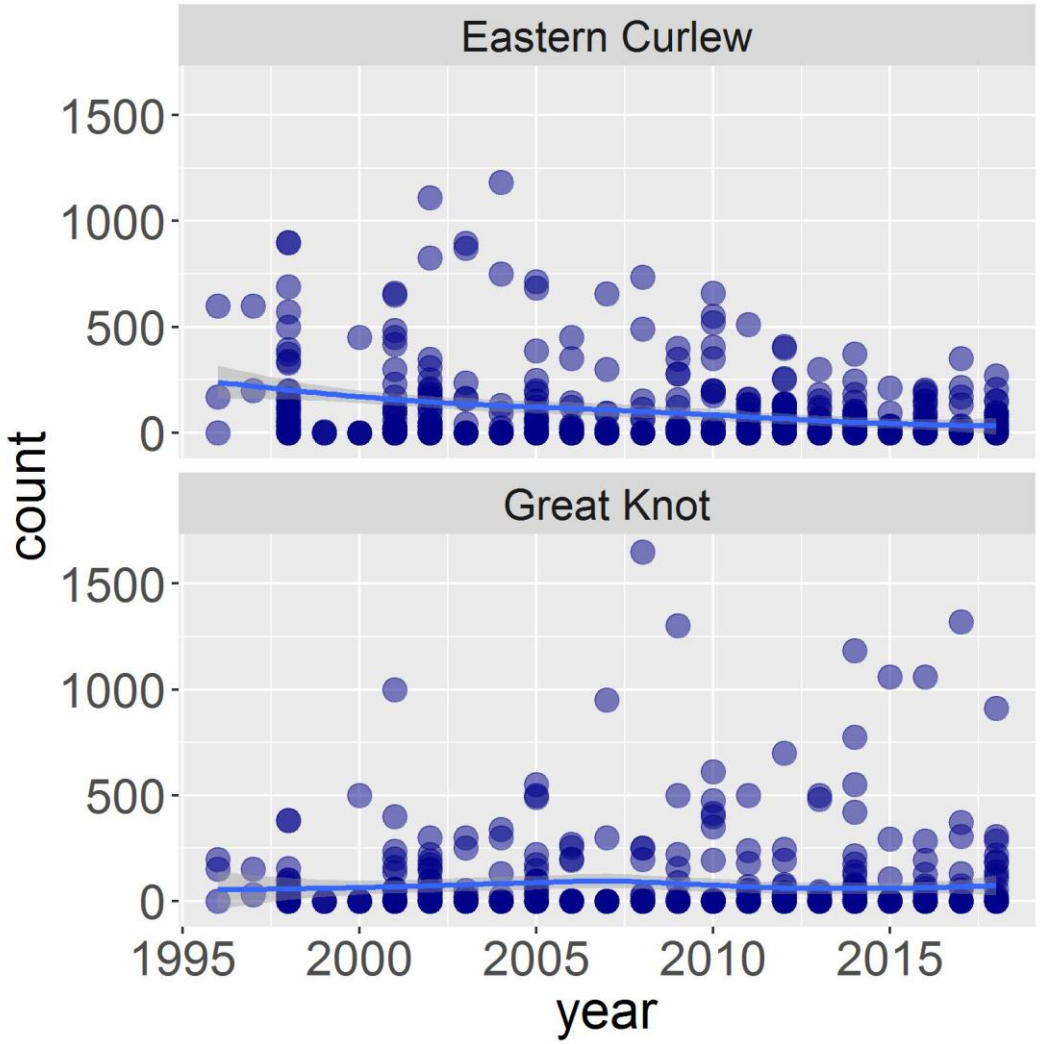
ggplot(data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))

```
# eastern curlew and great knot

ggplot(data = count_data) +
  geom_point(aes(x = species,
                 y = count)) +
  facet_wrap(~ species,
             ncol = 1)
```



ggplot(data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))

```
# eastern curlew and great knot

ggplot(data = count_data,
       aes(x = year, y = count)) +
  geom_point(colour = 'darkblue',
             alpha = 0.5,
             size = 4) +
  geom_smooth() +
  facet_wrap(~ species,
             ncol = 1)
```



filter()

```
# Select observations based on specific values of a
variable:

filter(count_data, species == ‘Eastern Curlew’)
```

R operators for logical comparisons:

< (less than) and **<=** (less than or equal to)

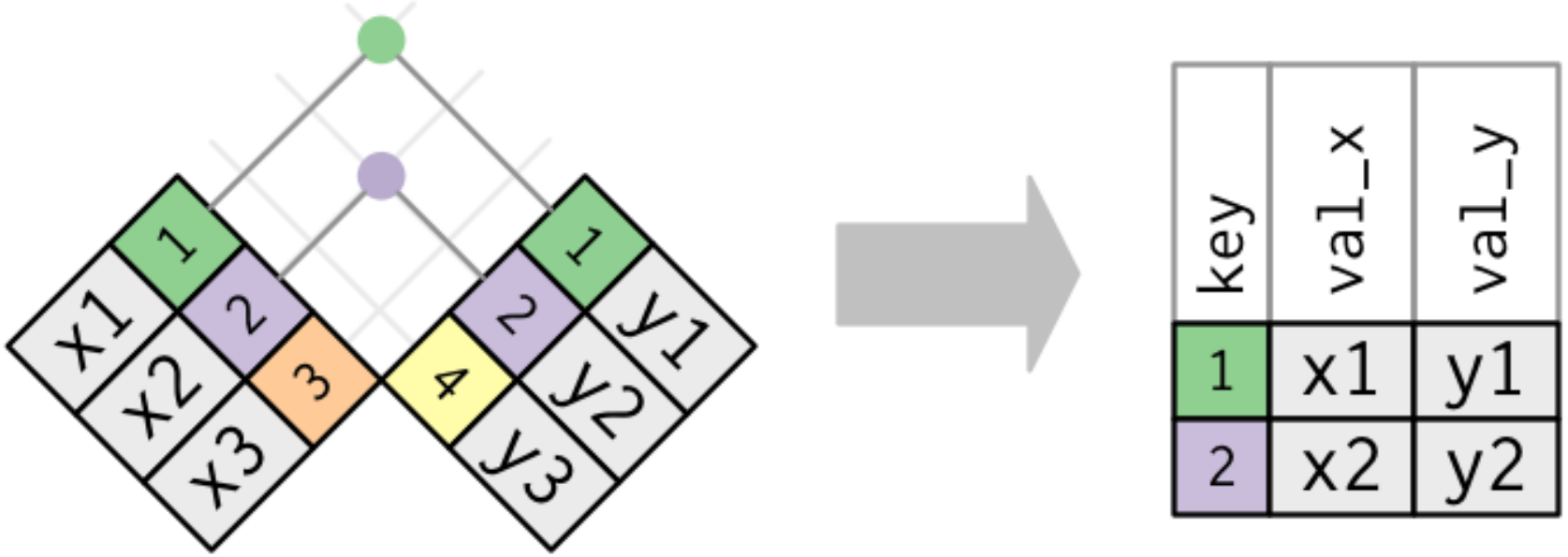
> (greater than) and **>=** (greater than or equal to)

== (equal to) and **!=** (equal to)

& (AND) and **|** (OR) and **!** (NOT)

inner_join()

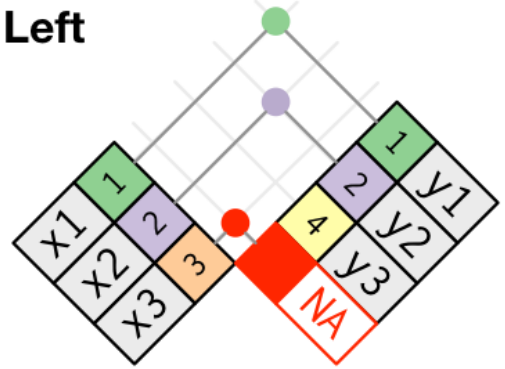
```
# Join datasets based on a shared variable
inner_join(count_data, weather_data, by = ‘date’)
```



Adapted from R4DS (r4ds.had.co.nz)

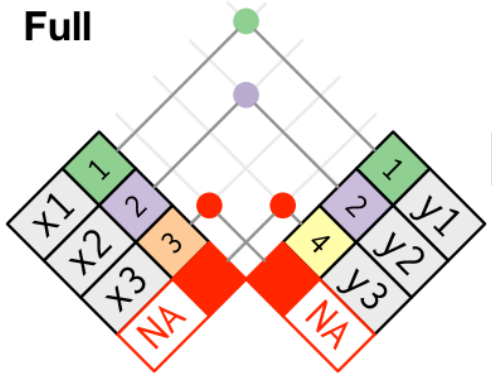
left_join(), right_join(), full_join()

Left



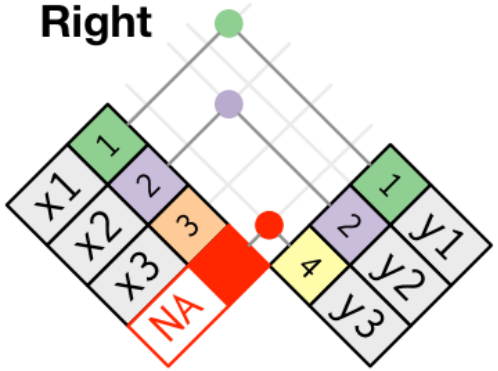
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

Full



key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

Right



key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

Joins are an incredibly useful data wrangling operation but need to be used cautiously; duplicated rows or slight mismatches in shared variables between datasets can cause un-wanted results!

Adapted from R4DS (r4ds.had.co.nz)

mutate()

```
# Add new variables (columns) to a dataset; number of
# rows in output equal to input
mutate(count_data, sqrt_count = sqrt(count))
```

group_by() and summarize()

```
# Group-wise data summaries; number of rows in output
# equal to the number of groups
count_datag <- group_by(count_data, species, site)
summarize(count_datag, max_count = max(count))
```

Pipes

Pipes are a powerful tool for expressing a sequence of multiple operations. Piping can make code easier to read and understand because:

1) code (contains (fewer (nested (functions))))
2) fewer intermediate objects to clutter up your memory and workspace

Pipes

No piping:

```
summarize(group_by(count_data, species, site),
          max_count = max(count))
```

Piping:

```
count_data %>%
     group_by(species, site) %>%
     summarize(max_count = max(count))
```

Pipes

No piping:

```
count_datag <- group_by(count_data, species, site)
summarize(count_datag, max_count = max(count))
```

Piping:

```
count_data %>%
     group_by(species, site) %>%
     summarize(max_count = max(count))
```

diverse, supportive user community

RStudio®

useR!

DataCamp

#rstats

R-bloggers

stackoverflow