ETC1010: Introduction to Data Analysis

Week 2, part B

Week of Tidy Data + Style

Lecturer: *Nicholas Tierney*
Department of Econometrics and Business Statistics
ETC1010.Clayton-x@monash.edu
11th Mar 2020

Update on how the class is delivered

How the class will now be delivered: Lectorials

- Lectorials are now recorded using Echo360
- **Do not come into class**, listen to the lectorials online and complete the exercises on rstudio cloud or locally.

How the class will now be delivered: Lab/quizzes

These will still be posted weekly, but we will give you an extra day or two to complete them

- Reading quizzes we expect you to complete before the lecture starts
  - So, Reading quiz 2A should be completed prior to lecture 2A
    - These will be closed shortly after lecture 2a starts (With some leeway as we transition into online classes to give you all a chance to get used to things)
- Lab quizzes require knowledge from the lecture - these need to be completed after the lecture
  - So, lab quiz 2A should be completed after Lecture 2a
  - Again with the same leeway as for reading quiz 2a above

How the class will now be delivered

Assessignments

- Assignment 1 will be posted today at the end of class
- Assignments will be submitted online
  - Please get in touch with us (if you haven't already) if you are a group of 1, or cannot get in touch with your group members.

Other assessments

- We will update you on this in more detail, but in short, these will be delivered and submitted online

Consult times

These will now be delivered online via a link to a zoom meeting, or other online video meeting service

There is a lot of change

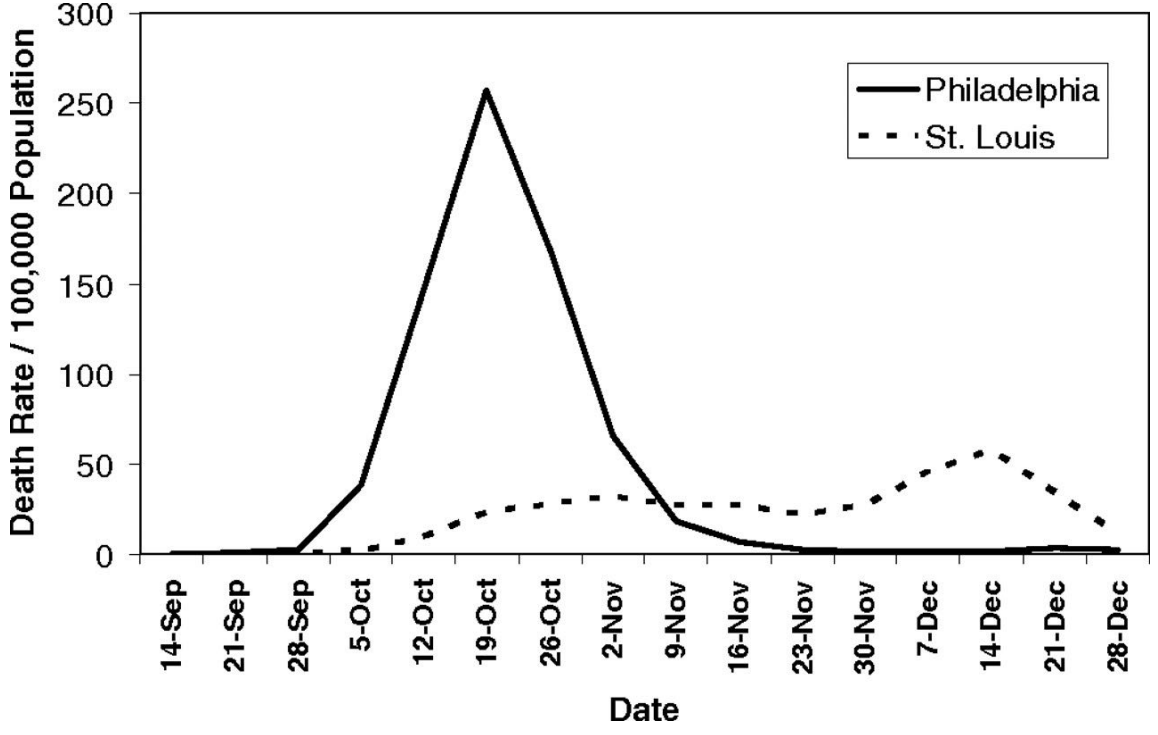
- There is a lot of change in the air, and things might seem uncertain.
- I am committed to helping you all learn how to do data analysis.
- Thank you all for your patience as we have changed this course. We are dealing with daily updates, and need to change on the fly.
- Perhaps now more than ever it is becoming so very relevant to our daily lives that we understand data, and that we can communicate it to others.
- Remember to get your information from reliable sources, like the WHO, the Australian Government, and see the latest data from Johns Hopkins.

Practice the most effective strategies we know

1. Wash your hands often, practice good cough & sneeze etiquette.
2. Try to touch your face as little as possible (mouth, nose, and eyes).
3. Practice social distancing (no hugs, kisses, handshakes, high fives)
4. Do not attend concerts, stage plays, sporting events, or any other mass entertainment events.
5. Refrain from visiting museums, exhibitions, movie theaters, night clubs, and other entertainment venues.
6. Stay away from social gatherings and events, (club meetings, religious services, parties)
7. Reduce travel to a minimum. Don't travel long distances if not absolutely necessary.
8. Do not use public transportation if not absolutely necessary.

Social distancing is hard

- How do we know it works?
- We have data from the last pandemic, the spanish flu.
- Places that practice social distancing vs those who did not had drastically different numbers:



(from (Hatchett et al, 2007))

There is a lot of change

To brighten things up, here are two youtubers I've been watching lately to destress and have "COVID19 free time"

- Lofty Pursuits
- SteveMRE1989

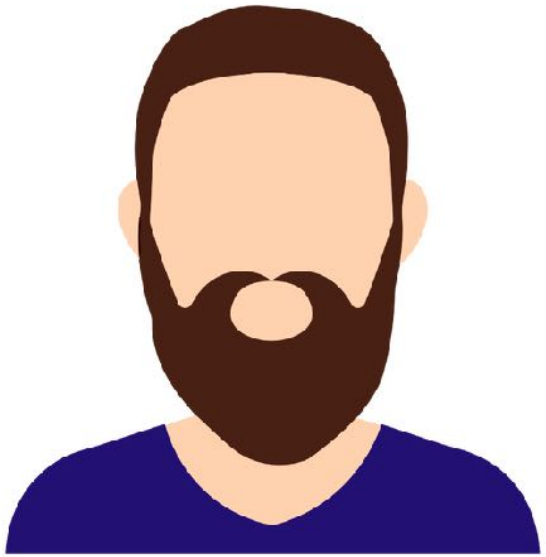
Your Turn: complete class survey

Available now on Ed, "Getting to know our class"

How to learn

I want to take some time to discuss ideas on learning, and how it ties into the course.

Beginner

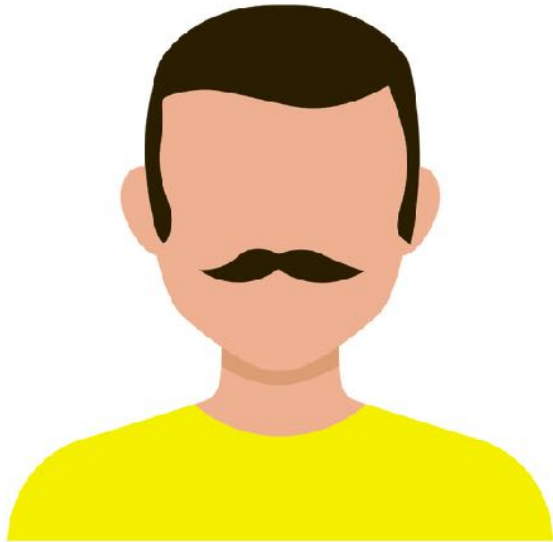


"I don't know what I don't know."

Competent Practitioner

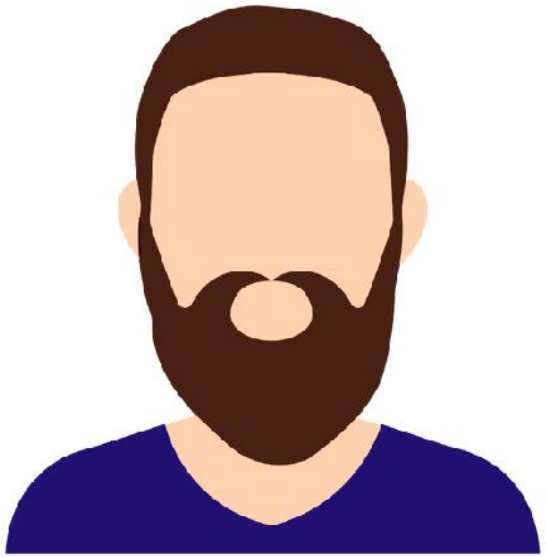
"I can do it, but I may look things up."

Expert



"I can handle anything you throw at me"

Beginner



"I don't know what I don't know."

Competent Practitioner

"I can do it, but I may look things up."

Expert

"I can handle anything you throw at me"

Beginner

"I don't know what I don't know."

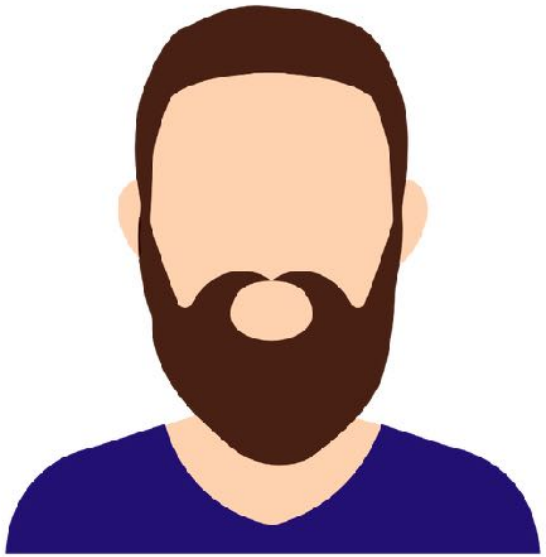
Competent Practitioner

"I can do it, but I may look things up."

Expert

"I can handle anything you throw at me"

Beginner

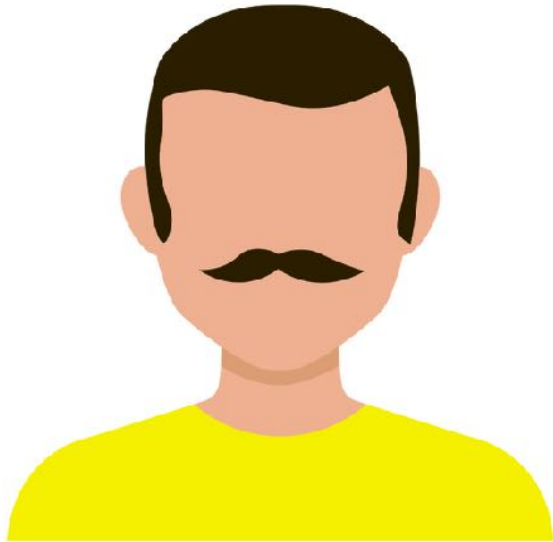


"I don't know what I don't know."

Competent Practitioner

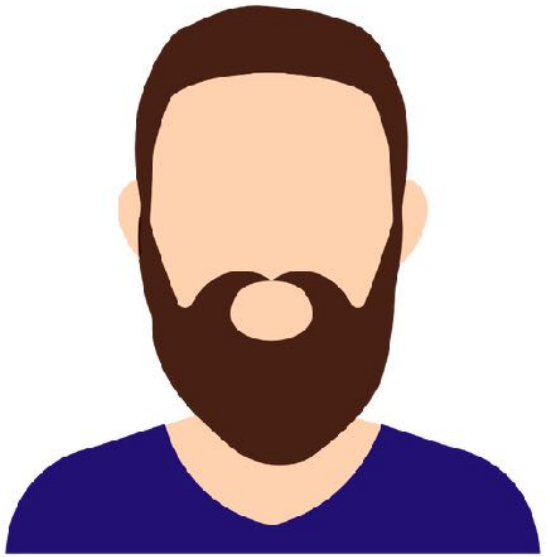
"I can do it, but I may look things up."

Expert



"I can handle anything you throw at me"

Beginner

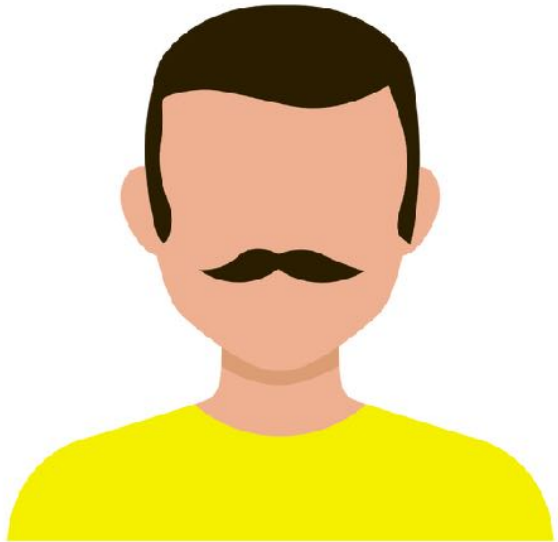


No
mental model

Competent
Practitioner

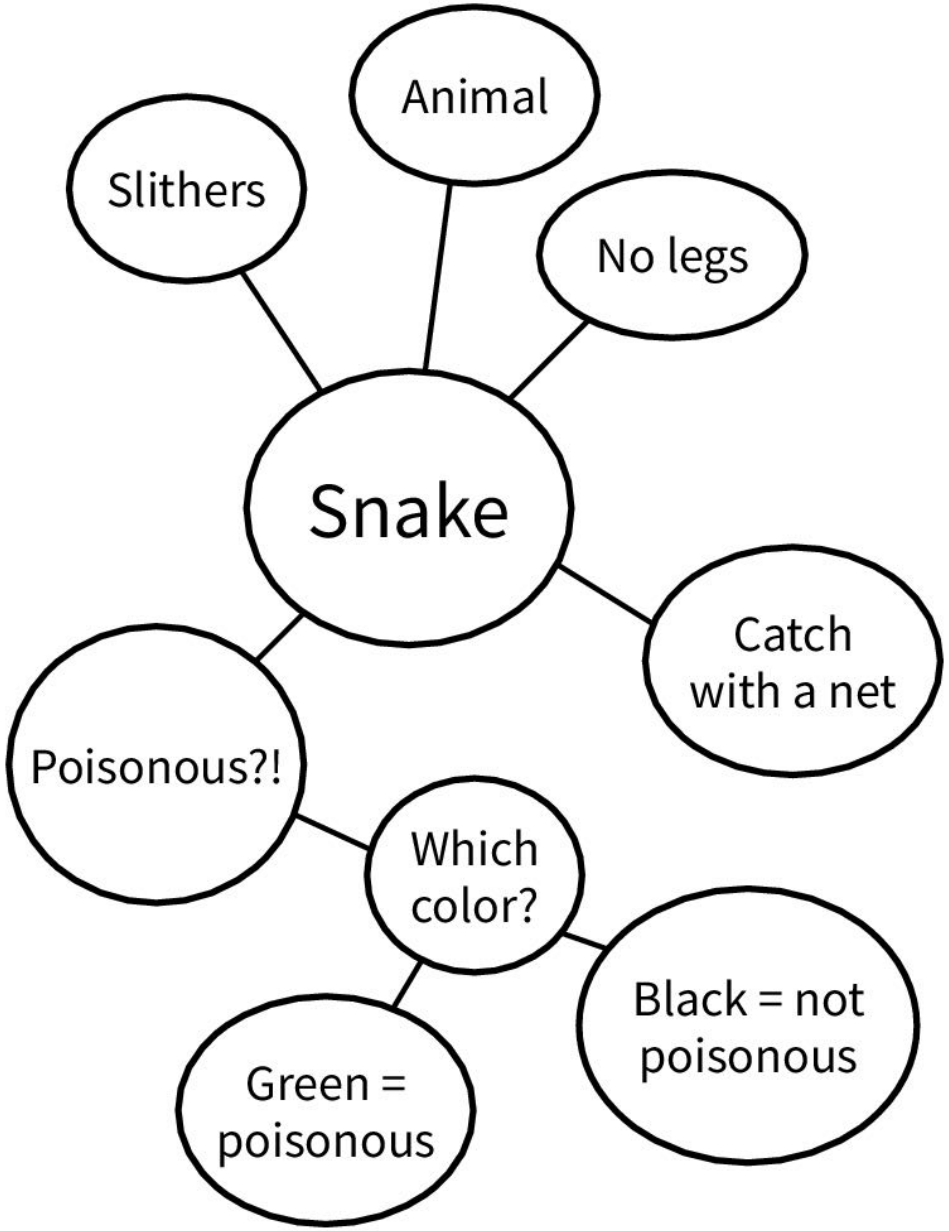
Useful
mental model

Expert



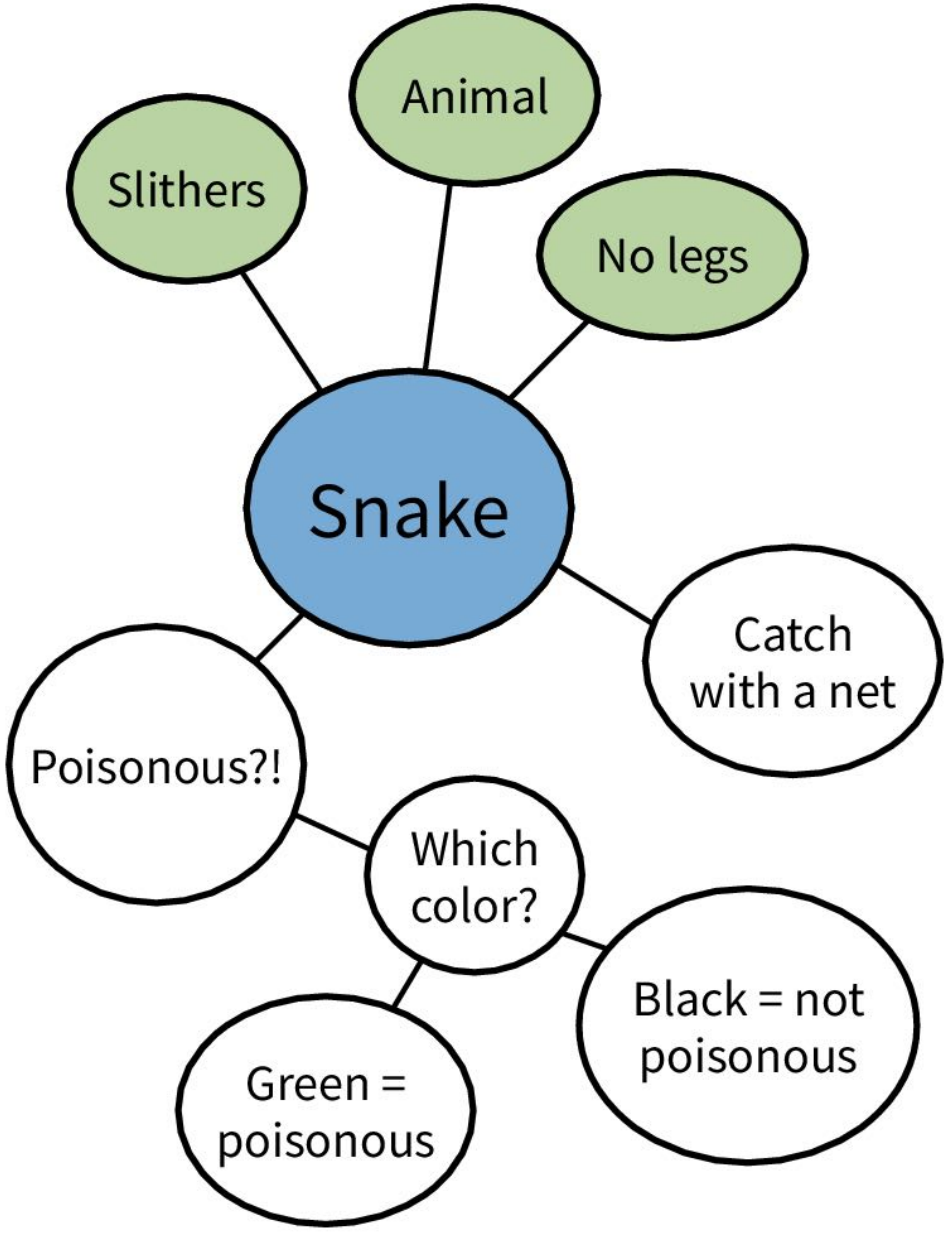
Elaborate
mental models

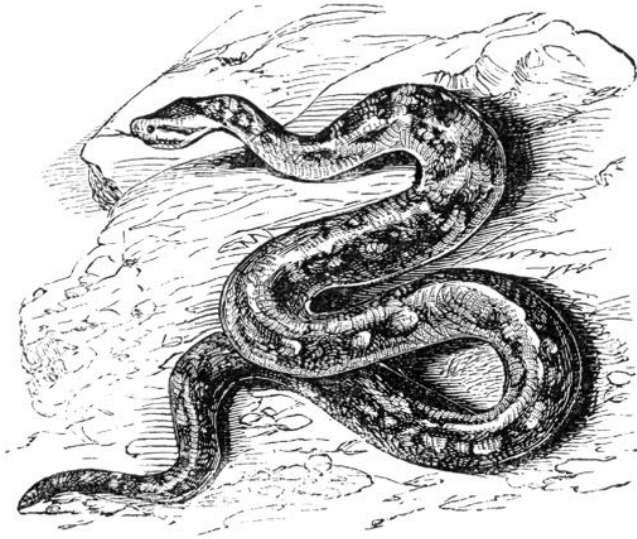
Mental Models

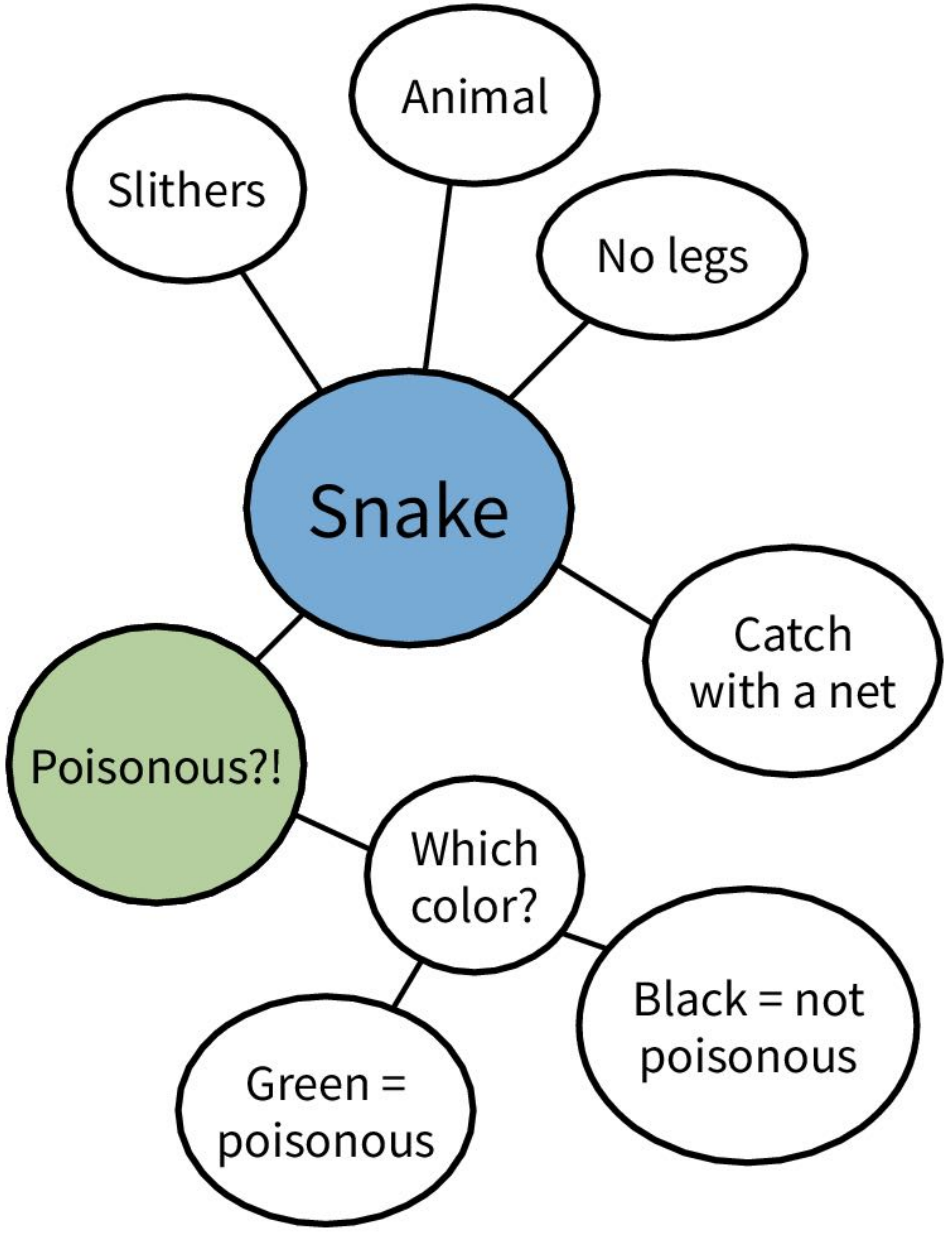


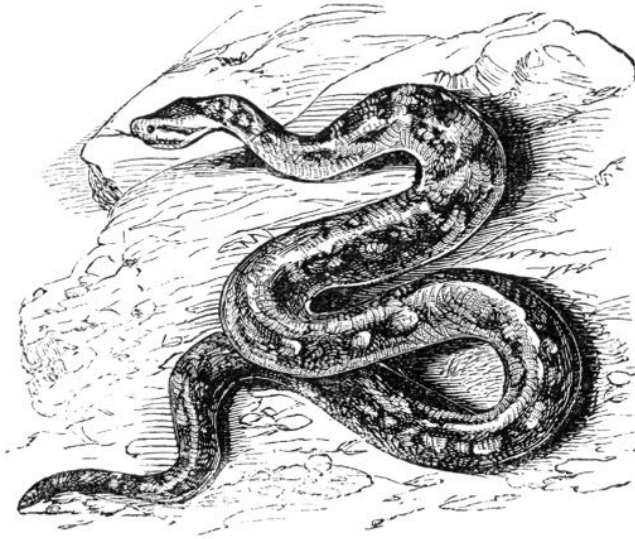
Mental Model

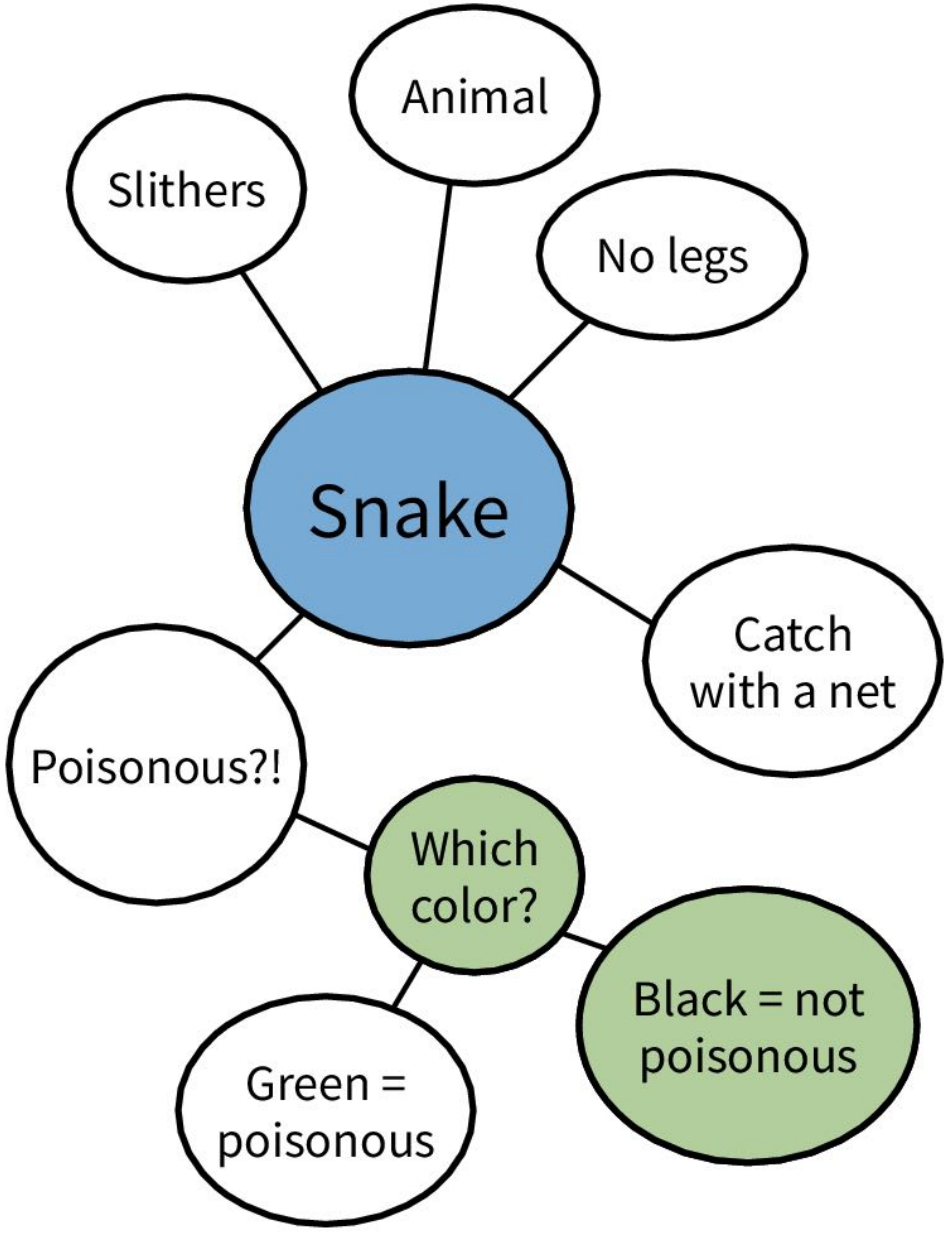
a structure that organizes facts according to their relationships

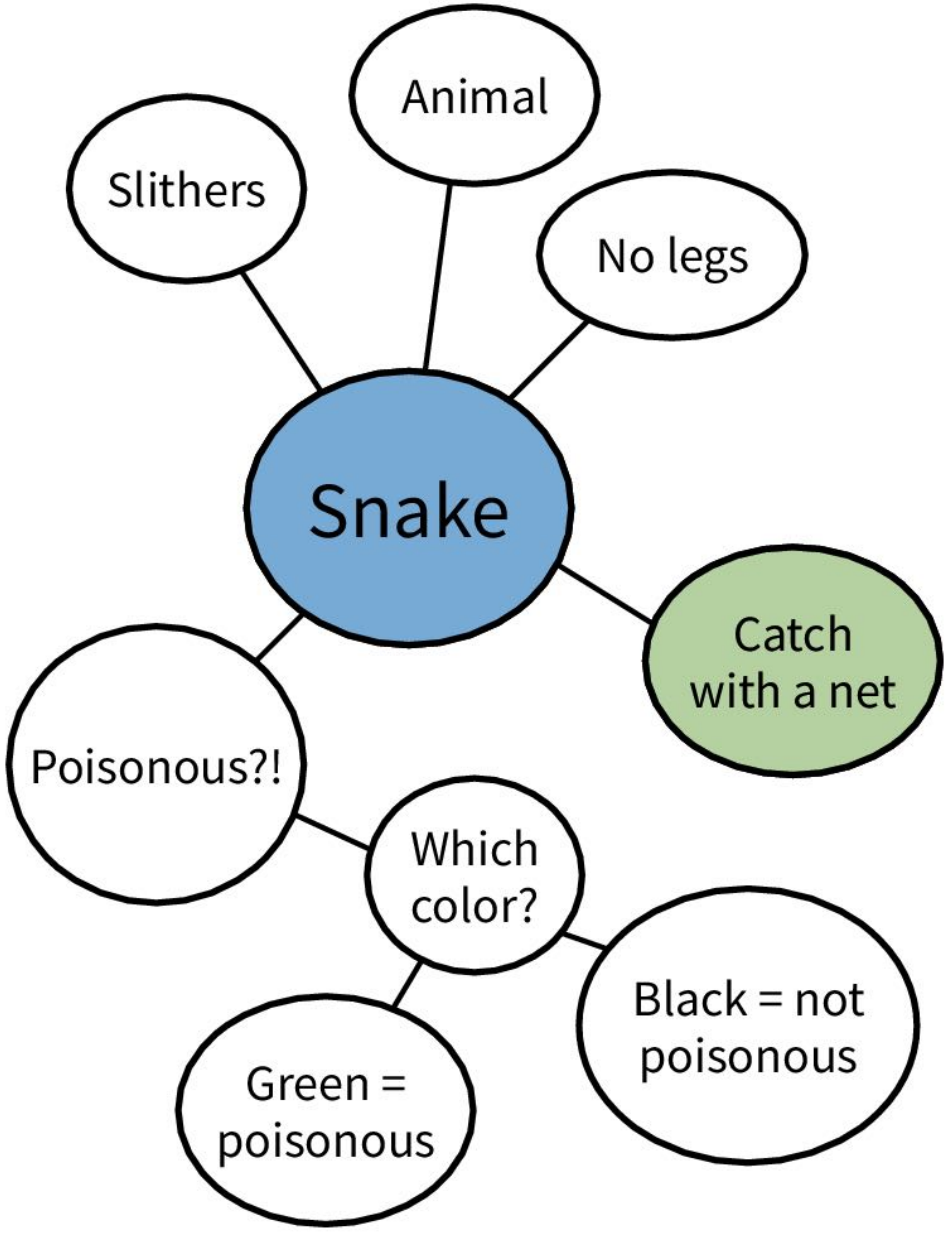


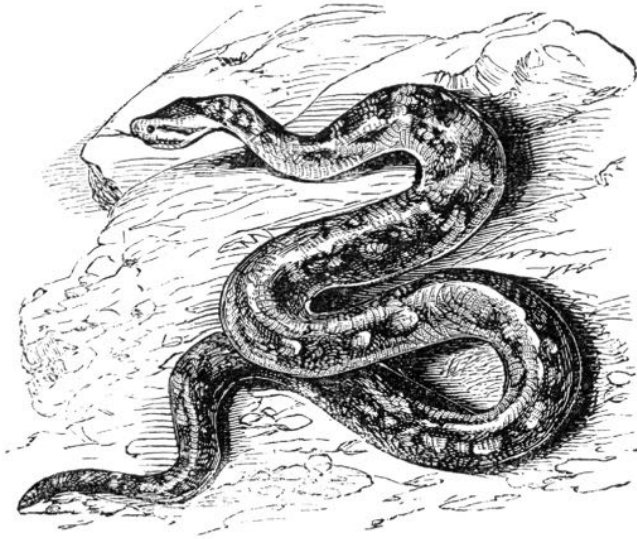












Mental Model

a structure that organizes facts according to their relationships

(demo)

recap

- Traffic Light System: Green = "good!" ; Red = "Help!"
- R + Rstudio
- Tower of babel analogy for writing R code
- Functions are _
- columns in data frames are accessed with _ ?
- packages are installed with _ ?
- packages are loaded with _ ?

- Why do we care about Reproducibility?
- Output + input of rmarkdown
- I have an assignment group
- I have made contact with my assignment group

The "pipe" operator - %>%

The symbol, %>% is referred to as the "pipe operator"

What you need to know:

- Read it as "then"
- It passes the output along to the next function

```
data %>%
  select(age, height, hair_colour) %>%
  filter(nationality == "australian")
```

" Use the data, THEN select the variables (columns), `age`, `height`, and `hair_colour` THEN filter so nationality is equal to "australian" "

That is all you need to know for the moment, but you can read more here

Problem solving (demo)

Some common questions you can ask yourself when something isn't working:

- Have I got my data?
- Does the thing exist? (Check environment)
- Have I run the code from the top down to where I am now?
- Did none of that work? (Now Restart R)
- Is the column I want there?
- Try using quotes "", or no quotes, or (last resort) backticks

Style guide

> *"Good coding style is like correct punctuation: you can manage without it, butitsuremakesthingseasiertoread." -- Hadley Wickham*

- Style guide for this course is based on the Tidyverse style guide: http://style.tidyverse.org/
- There's more to it than what we'll cover today, we'll mention more as we introduce more functionality, and do a recap later in the semester

File names and code chunk labels

- Do not use spaces in file names, use - or _ to separate words
- Use all lowercase letters

```
# Good
ucb-admit.csv

# Bad
UCB Admit.csv
```

Object names

- Use _ to separate words in object names
- Use informative but short object names
- Do not reuse object names within an analysis

```
# Good
acs_employed

# Bad
acs.employed
acs2
acs_subset
acs_subsetted_for_males
```

Spacing

- Put a space before and after all infix operators (=, +, -, <-, etc.), and when naming arguments in function calls.
- Always put a space after a comma, and never before (just like in regular English).

```
# Good
average <- mean(feet / 12 + inches, na.rm = TRUE)

# Bad
average<-mean(feet/12+inches,na.rm=TRUE)
```

ggplot

- Always end a line with +
- Always indent the next line

```
# Good
ggplot(diamonds, mapping = aes(x = price)) +
  geom_histogram()

# Bad
ggplot(diamonds,mapping=aes(x=price))+geom_histogram()
```

Long lines

- Limit your code to 80 characters per line. This fits comfortably on a printed page with a reasonably sized font.
- Take advantage of RStudio editor's auto formatting for indentation at line breaks.

Assignment

- Use <- not =

```r
# Good
x <- 2

# Bad
x = 2
```

Quotes

Use ", not ', for quoting text. The only exception is when the text already contains double quotes and no single quotes.

```
ggplot(diamonds, mapping = aes(x = price)) +
  geom_histogram() +
  # Good
  labs(title = "`Shine bright like a diamond`",
  # Good
       x = "Diamond prices",
  # Bad
       y = 'Frequency')
```

Source: Artwork by @allison_horst

Overview

- `filter()`
- `select()`
- `mutate()`
- `arrange()`
- `group_by()`
- `summarise()`
- `count()`

Artwork by @allison_horst

R Packages

```
avail_pkg <- available.packages()
dim(avail_pkg)
## [1] 15383    17
```

As of 2020-03-17 there are 15383 R packages available

Name clashes

```
library(tidyverse)
```

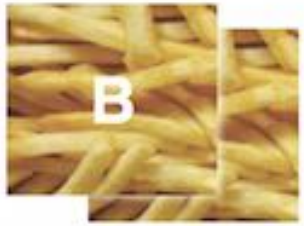
Many R packages

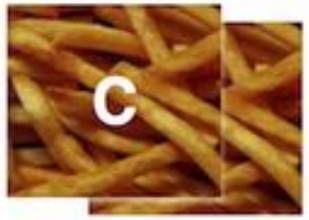
- A blessing & a curse!
- So many packages available, it can make it hard to choose!
- Many of the packages are designed to solve a specific problem
- The tidyverse is designed to work with many other packages following a consistent philosophy
- What this means is that you shouldn't notice it!

Let's talk about data

12 subjects

Three oils,
two batches





Five scales

RANCID

For 10 weeks

Example: french fries

- Experiment in Food Sciences at Iowa State University.
- Aim: find if cheaper oil could be used to make hot chips
- Question: Can people distinguish between chips fried in the new oils relative to those current market leader oil.
- 12 tasters recruited
- Each sampled two chips from each batch
- Over a period of ten weeks.

Same oil kept for a period of 10 weeks! May be a bit gross!

Example: french-fries - pivoting into long form

```
french_fries <- read_csv("data/french_fries.csv")
french_fries
```

```
## # A tibble: 6 x 9
##    time treatment subject   rep potato buttery grassy rancid painty
##   <dbl>     <dbl>   <dbl> <dbl>  <dbl>   <dbl>  <dbl>  <dbl>  <dbl>
## 1     1         1       3     1    2.9     0      0      0      5.5
## 2     1         1       3     2   14       0      0      1.1    0
## 3     1         1      10     1   11       6.4    0      0      0
## 4     1         1      10     2    9.9     5.9    2.9    2.2    0
## 5     1         1      15     1    1.2     0.1    0      1.1    5.1
## 6     1         1      15     2    8.8     3      3.6    1.5    2.3
```

This data set was brought to R by Hadley Wickham, and was one of the problems that inspired the thinking about tidy data and the tidyverse set of tools

Example: french-fries - pivoting into long form

```
fries_long <- french_fries %>%
  pivot_longer(cols = potato:painty,
               names_to = "type",
               values_to = "rating")
fries_long
```

```
## # A tibble: 3,480 x 6
##     time treatment subject   rep ty
##    <dbl>     <dbl>   <dbl> <dbl> <c
##  1     1         1       3     1 po
##  2     1         1       3     1 bu
##  3     1         1       3     1 gr
##  4     1         1       3     1 ra
##  5     1         1       3     1 pa
##  6     1         1       3     2 po
##  7     1         1       3     2 bu
##  8     1         1       3     2 gr
##  9     1         1       3     2 ra
## 10     1         1       3     2 pa
## # … with 3,470 more rows
```

Example: french-fries - pivoting back

```
fries_long
## # A tibble: 3,480 x 6
##     time treatment subject   rep typ
##    <dbl>     <dbl>   <dbl> <dbl> <c
##  1     1         1       3     1 pot
##  2     1         1       3     1 but
##  3     1         1       3     1 gra
##  4     1         1       3     1 ran
##  5     1         1       3     1 pas
##  6     1         1       3     2 pot
##  7     1         1       3     2 but
##  8     1         1       3     2 gra
##  9     1         1       3     2 ran
## 10     1         1       3     2 pas
## # … with 3,470 more rows
```

```
fries_long %>%
  pivot_wider(names_from = type,
              values_from = rating)
## # A tibble: 696 x 9
##     time treatment subject   rep pot
##    <dbl>     <dbl>   <dbl> <dbl>  <d
##  1     1         1       3     1
##  2     1         1       3     2
##  3     1         1      10     1
##  4     1         1      10     2
##  5     1         1      15     1
##  6     1         1      15     2
##  7     1         1      16     1
##  8     1         1      16     2
##  9     1         1      19     1
## 10     1         1      19     2
## # … with 686 more rows
```

`filter()`

choose observations from your data

filter(): example

```
fries_long %>%
  filter(subject == 10)
## # A tibble: 300 x 6
##     time treatment subject   rep type    rating
##    <dbl>     <dbl>   <dbl> <dbl> <chr>    <dbl>
##  1     1         1      10     1 potato    11
##  2     1         1      10     1 buttery    6.4
##  3     1         1      10     1 grassy     0
##  4     1         1      10     1 rancid     0
##  5     1         1      10     1 painty     0
##  6     1         1      10     2 potato     9.9
##  7     1         1      10     2 buttery    5.9
##  8     1         1      10     2 grassy     2.9
##  9     1         1      10     2 rancid     2.2
## 10     1         1      10     2 painty     0
## # … with 290 more rows
```

`filter()`: details

Filtering requires comparison to find the subset of observations of interest. What do you think the following mean?

- `subject != 10`
- `x > 10`
- `x >= 10`
- `class %in% c("A", "B")`
- `!is.na(y)`

03:00

`filter()`: details

```
subject != 10
```

Find rows corresponding to all subjects except subject 10

```
x > 10
```

find all rows where variable x has values bigger than 10

```
x >= 10
```

finds all rows variable x is greater than or equal to 10.

```
class %in% c("A", "B")
```

finds all rows where variable `class` is either A or B

```
!is.na(y)
```

finds all rows that *DO NOT* have a missing value for variable y

Your turn: open french-fries.Rmd

Filter the french fries data to have:

- only week 1
- oil type 1 (oil type is called treatment)
- oil types 1 and 3 but not 2
- weeks 1-4 only

French Fries Filter: only week 1

```
fries_long %>% filter(time == 1)
## # A tibble: 360 x 6
##     time treatment subject   rep type    rating
##    <dbl>     <dbl>   <dbl> <dbl> <chr>    <dbl>
##  1     1         1       3     1 potato     2.9
##  2     1         1       3     1 buttery    0
##  3     1         1       3     1 grassy     0
##  4     1         1       3     1 rancid     0
##  5     1         1       3     1 painty     5.5
##  6     1         1       3     2 potato    14
##  7     1         1       3     2 buttery    0
##  8     1         1       3     2 grassy     0
##  9     1         1       3     2 rancid     1.1
## 10     1         1       3     2 painty     0
## # … with 350 more rows
```

French Fries Filter: oil type 1

```
fries_long %>% filter(treatment == 1)
## # A tibble: 1,160 x 6
##     time treatment subject   rep type    rating
##    <dbl>     <dbl>   <dbl> <dbl> <chr>    <dbl>
##  1     1         1       3     1 potato     2.9
##  2     1         1       3     1 buttery    0
##  3     1         1       3     1 grassy     0
##  4     1         1       3     1 rancid     0
##  5     1         1       3     1 painty     5.5
##  6     1         1       3     2 potato    14
##  7     1         1       3     2 buttery    0
##  8     1         1       3     2 grassy     0
##  9     1         1       3     2 rancid     1.1
## 10     1         1       3     2 painty     0
## # … with 1,150 more rows
```

French Fries Filter: oil types 1 and 3 but not 2

```
fries_long %>% filter(treatment != 2)
## # A tibble: 2,320 x 6
##     time treatment subject   rep type    rating
##    <dbl>     <dbl>   <dbl> <dbl> <chr>    <dbl>
##  1     1         1       3     1 potato     2.9
##  2     1         1       3     1 buttery    0
##  3     1         1       3     1 grassy     0
##  4     1         1       3     1 rancid     0
##  5     1         1       3     1 painty     5.5
##  6     1         1       3     2 potato    14
##  7     1         1       3     2 buttery    0
##  8     1         1       3     2 grassy     0
##  9     1         1       3     2 rancid     1.1
## 10     1         1       3     2 painty     0
## # … with 2,310 more rows
```

French Fries Filter: weeks 1-4 only

```
fries_long %>% filter(time %in% c("1", "2", "3", "4"))
## # A tibble: 1,440 x 6
##     time treatment subject   rep type    rating
##    <dbl>     <dbl>   <dbl> <dbl> <chr>    <dbl>
##  1     1         1       3     1 potato     2.9
##  2     1         1       3     1 buttery    0
##  3     1         1       3     1 grassy     0
##  4     1         1       3     1 rancid     0
##  5     1         1       3     1 painty     5.5
##  6     1         1       3     2 potato    14
##  7     1         1       3     2 buttery    0
##  8     1         1       3     2 grassy     0
##  9     1         1       3     2 rancid     1.1
## 10     1         1       3     2 painty     0
## # … with 1,430 more rows
```

about `%in%`

[demo]

`select()`

- Chooses which variables to keep in the data set.
- Useful when there are many variables but you only need some of them for an analysis.

`select()`: a comma separated list of variables, by name.

```
french_fries %>%
  select(time,
         treatment,
         subject)
## # A tibble: 696 x 3
##     time treatment subject
##    <dbl>     <dbl>   <dbl>
##  1     1         1       3
##  2     1         1       3
##  3     1         1      10
##  4     1         1      10
##  5     1         1      15
##  6     1         1      15
##  7     1         1      16
##  8     1         1      16
##  9     1         1      19
## 10     1         1      19
## # … with 686 more rows
```

`select()`: drop selected variables by prefixing with `-`

```
french_fries %>%
  select(-time,
         -treatment,
         -subject)
```

```
## # A tibble: 696 x 6
##      rep potato buttery grassy rancid painty
##    <dbl>  <dbl>   <dbl>  <dbl>  <dbl>  <dbl>
##  1     1    2.9     0      0      0      5.5
##  2     2   14       0      0      1.1    0
##  3     1   11       6.4    0      0      0
##  4     2    9.9     5.9    2.9    2.2    0
##  5     1    1.2     0.1    0      1.1    5.1
##  6     2    8.8     3      3.6    1.5    2.3
##  7     1    9       2.6    0.4    0.1    0.2
##  8     2    8.2     4.4    0.3    1.4    4
##  9     1    7       3.2    0      4.9    3.2
## 10     2   13       0      3.1    4.3   10.3
## # … with 686 more rows
```

select()

Inside `select()` you can use text-matching of the names like `starts_with()`, `ends_with()`, `contains()`, `matches()`, or `everything()`

```
french_fries %>%
  select(contains("e"))
## # A tibble: 696 x 5
##     time treatment subject   rep buttery
##    <dbl>     <dbl>   <dbl> <dbl>   <dbl>
##  1     1         1       3     1     0
##  2     1         1       3     2     0
##  3     1         1      10     1     6.4
##  4     1         1      10     2     5.9
##  5     1         1      15     1     0.1
##  6     1         1      15     2     3
##  7     1         1      16     1     2.6
##  8     1         1      16     2     4.4
##  9     1         1      19     1     3.2
## 10     1         1      19     2     0
## # … with 686 more rows
```

`select()`: Using it

You can use the colon, `:`, to choose variables in order of the columns

```
french_fries %>%
  select(time:subject)
## # A tibble: 696 x 3
##     time treatment subject
##    <dbl>     <dbl>   <dbl>
##  1     1         1       3
##  2     1         1       3
##  3     1         1      10
##  4     1         1      10
##  5     1         1      15
##  6     1         1      15
##  7     1         1      16
##  8     1         1      16
##  9     1         1      19
## 10     1         1      19
## # … with 686 more rows
```

Your turn: back to the french fries data

- `select()` time, treatment and rep
- `select()` subject through to rating
- drop subject

03:00

Artwork by @allison_horst

`mutate()`: create a new variable; keep existing ones

```
french_fries
## # A tibble: 696 x 9
##     time treatment subject   rep potato buttery grassy rancid painty
##    <dbl>     <dbl>   <dbl> <dbl>  <dbl>   <dbl>  <dbl>  <dbl>  <dbl>
##  1     1         1       3     1    2.9     0      0      0      5.5
##  2     1         1       3     2   14       0      0      1.1    0
##  3     1         1      10     1   11       6.4    0      0      0
##  4     1         1      10     2    9.9     5.9    2.9    2.2    0
##  5     1         1      15     1    1.2     0.1    0      1.1    5.1
##  6     1         1      15     2    8.8     3      3.6    1.5    2.3
##  7     1         1      16     1    9       2.6    0.4    0.1    0.2
##  8     1         1      16     2    8.2     4.4    0.3    1.4    4
##  9     1         1      19     1    7       3.2    0      4.9    3.2
## 10     1         1      19     2   13       0      3.1    4.3   10.3
## # … with 686 more rows
```

`mutate()`: create a new variable; keep existing ones

```
french_fries %>%
  mutate(rainty = rancid + painty)
```

```
## # A tibble: 696 x 10
##     time treatment subject   rep potato buttery grassy rancid painty rainty
##    <dbl>     <dbl>   <dbl> <dbl>  <dbl>   <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
##  1     1         1       3     1    2.9       0      0      0    5.5    5.5
##  2     1         1       3     2   14         0      0      1.1  0      1.1
##  3     1         1      10     1   11         6.4    0      0    0      0
##  4     1         1      10     2    9.9       5.9    2.9    2.2  0      2.2
##  5     1         1      15     1    1.2       0.1    0      1.1  5.1    6.20
##  6     1         1      15     2    8.8       3      3.6    1.5  2.3    3.8
##  7     1         1      16     1    9         2.6    0.4    0.1  0.2    0.3
##  8     1         1      16     2    8.2       4.4    0.3    1.4  4      5.4
##  9     1         1      19     1    7         3.2    0      4.9  3.2    8.1
## 10     1         1      19     2   13         0      3.1    4.3  10.3   14.6
## # … with 686 more rows
```

Your turn: french fries

Compute a new variable called `lrating` by taking a log of the rating

02:00

`summarise()`: boil data down to one row observation

```
fries_long
```

```
## # A tibble: 6 x 6
##    time treatment subject   rep type    rating
##   <dbl>     <dbl>   <dbl> <dbl> <chr>    <dbl>
## 1     1         1       3     1 potato     2.9
## 2     1         1       3     1 buttery    0
## 3     1         1       3     1 grassy     0
## 4     1         1       3     1 rancid     0
## 5     1         1       3     1 painty     5.5
## 6     1         1       3     2 potato    14
```

```
fries_long %>%
  summarise(rating = mean(rating, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   rating
##    <dbl>
## 1   3.16
```

What if we want a summary for each `type`?

use `group_by()`

Using `summarise()` + `group_by()`

Produce summaries for every group:

```
fries_long %>%
  group_by(type) %>%
  summarise(rating = mean(rating, na.rm=TRUE))
## # A tibble: 5 x 2
##   type    rating
##   <chr>    <dbl>
## 1 buttery  1.82
## 2 grassy   0.664
## 3 painty   2.52
## 4 potato   6.95
## 5 rancid   3.85
```

Your turn: Back to french-fries.Rmd

- Compute the average rating by subject
- Compute the average rancid rating per week

03:00

french fries answers

```
fries_long %>%
  group_by(subject) %>%
  summarise(rating = mean(rating, na.rm=TRUE))
## # A tibble: 12 x 2
##    subject rating
##      <dbl>  <dbl>
##  1       3   2.46
##  2      10   4.24
##  3      15   2.16
##  4      16   3.00
##  5      19   4.54
##  6      31   4.00
##  7      51   4.39
##  8      52   2.72
##  9      63   3.48
## 10      78   1.94
## 11      79   1.94
## 12      86   2.94
```

french fries answers

```
fries_long %>%
  filter(type == "rancid") %>%
  group_by(time) %>%
  summarise(rating = mean(rating, na.rm=TRUE))
## # A tibble: 10 x 2
##     time rating
##    <dbl>  <dbl>
##  1     1   2.36
##  2     2   2.85
##  3     3   3.72
##  4     4   3.60
##  5     5   3.53
##  6     6   4.08
##  7     7   3.89
##  8     8   4.27
##  9     9   4.67
## 10    10   6.07
```

`arrange()`: orders data by a given variable.

Useful for display of results (but there are other uses!)

```
fries_long %>%
  group_by(type) %>%
  summarise(rating = mean(rating, na.rm=TRUE))
## # A tibble: 5 x 2
##   type    rating
##   <chr>    <dbl>
## 1 buttery  1.82
## 2 grassy   0.664
## 3 painty   2.52
## 4 potato   6.95
## 5 rancid   3.85
```

arrange()

```
fries_long %>%
  group_by(type) %>%
  summarise(rating = mean(rating, na.rm=TRUE)) %>%
  arrange(rating)
## # A tibble: 5 x 2
##   type    rating
##   <chr>    <dbl>
## 1 grassy   0.664
## 2 buttery  1.82
## 3 painty   2.52
## 4 rancid   3.85
## 5 potato   6.95
```

Your turn: french-fries.Rmd - arrange

- Arrange the average rating by type in decreasing order
- Arrange the average subject rating in order lowest to highest.

02:00

arrange() answers

```
fries_long %>%
  group_by(type) %>%
  summarise(rating = mean(rating, na.rm=TRUE)) %>%
  arrange(desc(rating))
## # A tibble: 5 x 2
##   type    rating
##   <chr>    <dbl>
## 1 potato   6.95
## 2 rancid   3.85
## 3 painty   2.52
## 4 buttery  1.82
## 5 grassy   0.664
```

arrange() answers

```
fries_long %>%
  group_by(subject) %>%
  summarise(rating = mean(rating, na.rm=TRUE)) %>%
  arrange(rating)
## # A tibble: 12 x 2
##    subject rating
##      <dbl>  <dbl>
##  1      78   1.94
##  2      79   1.94
##  3      15   2.16
##  4       3   2.46
##  5      52   2.72
##  6      86   2.94
##  7      16   3.00
##  8      63   3.48
##  9      31   4.00
## 10      10   4.24
## 11      51   4.39
## 12      19   4.54
```

`count()` the number of things in a given column

```
fries_long %>%
  count(type, sort = TRUE)
## # A tibble: 5 x 2
##   type        n
##   <chr>   <int>
## 1 buttery   696
## 2 grassy    696
## 3 painty    696
## 4 potato    696
## 5 rancid    696
```

Your turn: `count()`

- count the number of subjects
- count the number of types

02:00

French Fries: Putting it together to problem solve

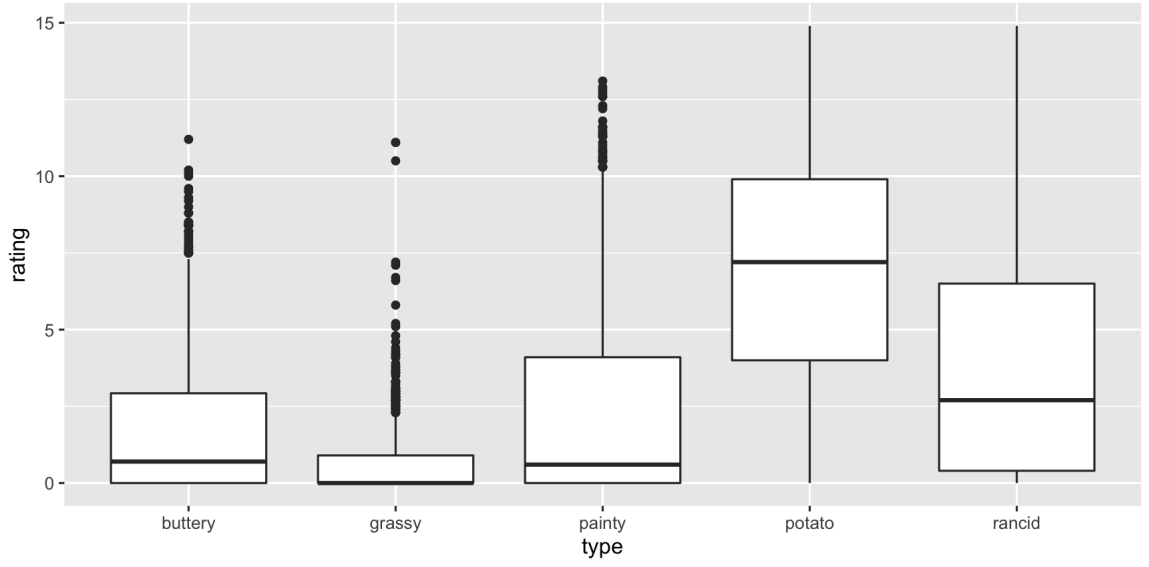
French Fries: Are ratings similar?

```
fries_long %>%
  group_by(type) %>%
  summarise(
    m = mean(rating,
             na.rm = TRUE),
    sd = sd(rating,
            na.rm = TRUE)) %>%
  arrange(-m)
## # A tibble: 5 x 3
##   type        m    sd
##   <chr>   <dbl> <dbl>
## 1 potato  6.95   3.58
## 2 rancid  3.85   3.78
## 3 painty  2.52   3.39
## 4 buttery 1.82   2.41
## 5 grassy  0.664  1.32
```

The scales of the ratings are quite different. Mostly the chips are rated highly on potato'y, but low on grassy.

French Fries: Are ratings similar?

```
ggplot(fries_long,
       aes(x = type,
           y = rating)) +
  geom_boxplot()
```



French Fries: Are reps like each other?

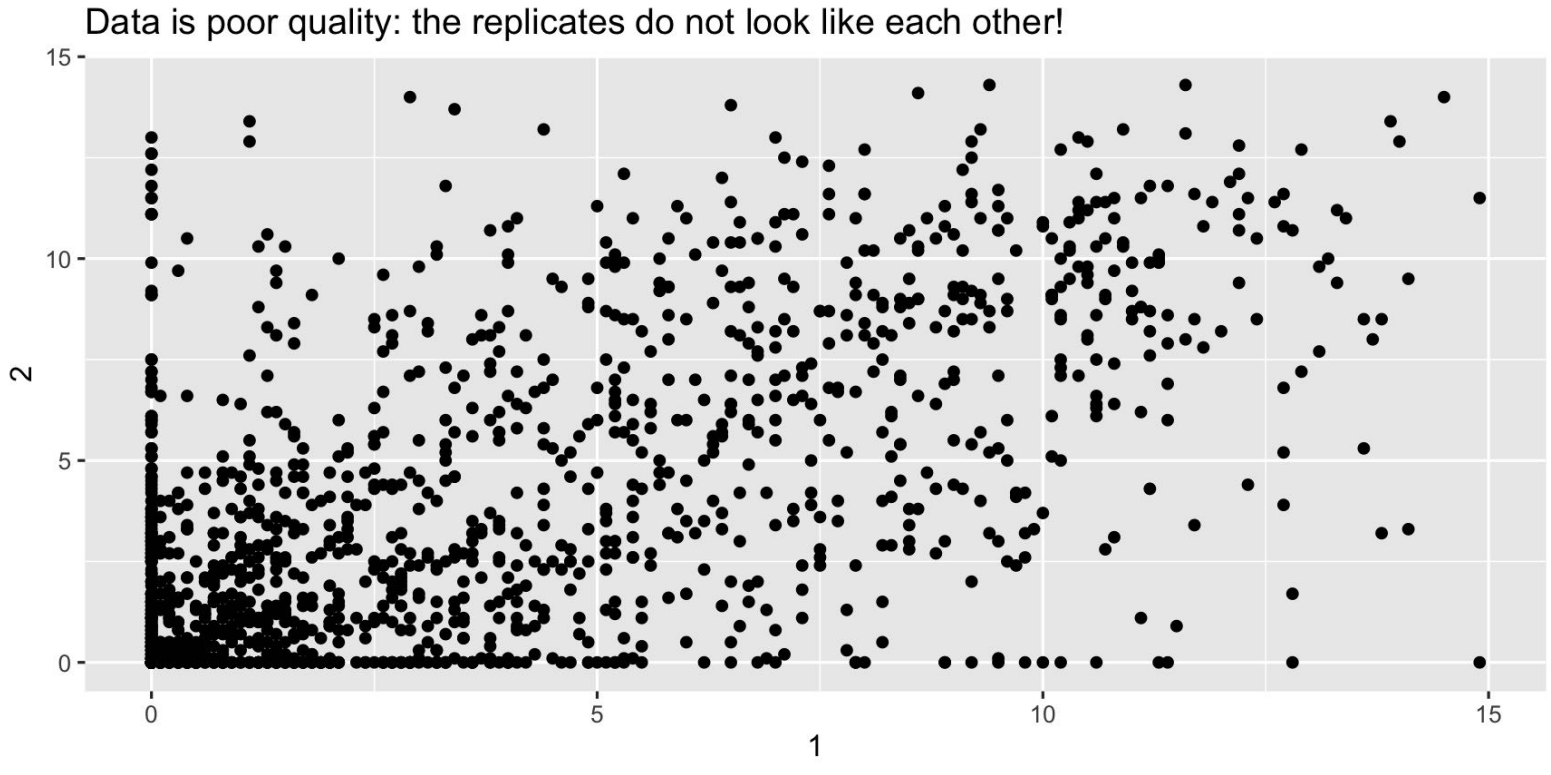
```
fries_spread <- fries_long %>%
  pivot_wider(names_from = rep,
              values_from = rating)

fries_spread
## # A tibble: 1,740 x 6
##     time treatment subject type      `1`   `2`
##    <dbl>     <dbl>   <dbl> <chr>   <dbl> <dbl>
##  1     1         1       3 potato    2.9  14
##  2     1         1       3 buttery   0     0
##  3     1         1       3 grassy    0     0
##  4     1         1       3 rancid    0     1.1
##  5     1         1       3 painty    5.5   0
##  6     1         1      10 potato   11     9.9
##  7     1         1      10 buttery   6.4   5.9
##  8     1         1      10 grassy    0     2.9
##  9     1         1      10 rancid    0     2.2
## 10     1         1      10 painty    0     0
## # … with 1,730 more rows
```

French Fries: Are reps like each other?

```
summarise(fries_spread,
          r = cor(`1`, `2`, use = "complete.obs"))
## # A tibble: 1 x 1
##       r
##   <dbl>
## 1 0.668
```

French Fries:

```
ggplot(fries_spread,
       aes(x = `1`,
           y = `2`)) +
geom_point() +
labs(title = "Data is poor quality: the replicates do not look like each other!")
```

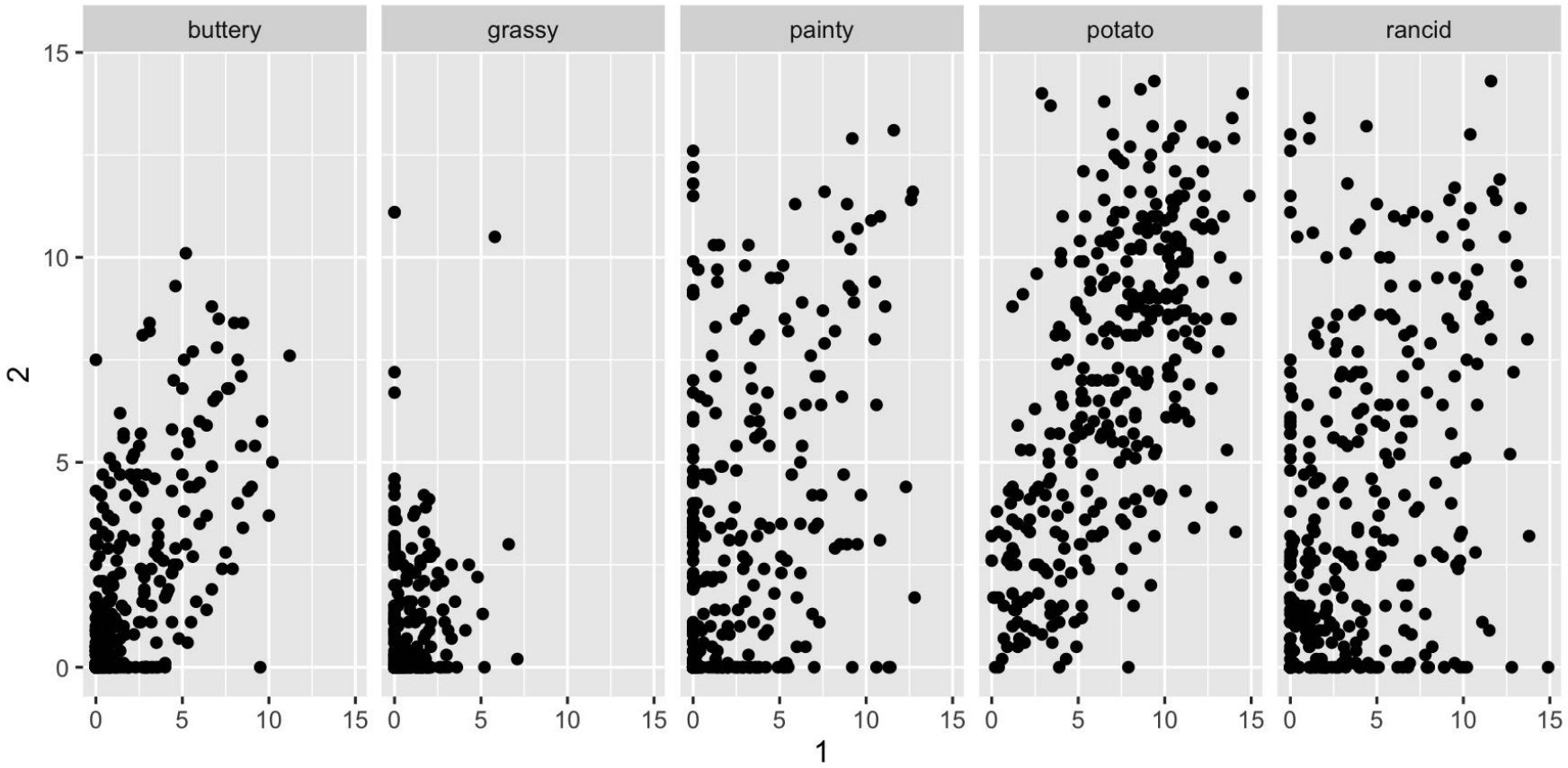


French Fries: Replicates by rating type

```
fries_spread %>%
  group_by(type) %>%
  summarise(r = cor(x = `1`,
                    y = `2`,
                    use = "complete.obs"))
## # A tibble: 5 x 2
##   type        r
##   <chr>   <dbl>
## 1 buttery 0.650
## 2 grassy  0.239
## 3 painty  0.479
## 4 potato  0.616
## 5 rancid  0.391
```

French Fries: Replicates by rating type

```
ggplot(fries_spread, aes(x=`1`, y=`2`)) +
  geom_point() + facet_wrap(~type, ncol = 5)
```



When to use quotes? " ' , nothing, or backtick?

When to use quotes? " ' , nothing, or backtick?

- Use no quotes (bare variable names) when the variable exists
- Otherwise use strings

Example:

```
fries_long %>%
  pivot_wider(names_from = type,
              values_from = rating)
```

vs

```
french_fries %>%
  pivot_longer(cols = potato:painty,
               names_to = "type",
               values_to = "rating")
```

When to use quotes? " ' , nothing, or backtick?

Variables with unusual names (starting with numbers, spaces, or containing special characters like !@#$%^&*( ) – need to be referenced with backticks:

```
data %>% select(`name with spaces`)
```

Lab exercise: Exploring data PISA data

Open `pisa.Rmd` on rstudio cloud.

Lab Quiz

Time to take the lab quiz.