# Introduction to R

## PSY517 Quantitative Analysis III

Derek Powell

August 2, 2021



Introduction to R

2021-08-24

Introduction to R
PSY517 Quantitative Analysis III
Derek Powell
August 2, 2021

# Why R? (or, why am I doing this to you?)





Why R? (or, why am I doing this to you?)

poll class

- how many, when you heard class would use R, felt excited?
  - how many felt worried?
- how many have used R? what have you done with it?
- How was the LSR reading?

# Why R? (or, why am I doing this to you?)

- R is a powerful tool for statistics
- R is a powerful tool for data management
  - Scripts and automation reduce errors, reduce tedium, and make analyses reproducible
- Programming is where the $$$ is at
  - Basic programming knowledge is becoming a necessity in industry and academia
- R has a great community, tons of support, and is really not that hard to learn
- We will substitute programming in place of math to understand statistics





Why R? (or, why am I doing this to you?)
- R is a powerful tool for statistics
- R is a powerful tool for data management
  - Scripts and automation reduce errors, reduce tedium, and make analyses reproducible
- Programming is where the $$$ is at
  - Basic programming knowledge is becoming a necessity in industry and academia
- R has a great community, tons of support, and is really not that hard to learn
- We will substitute programming in place of math to understand statistics

# Kicking ass

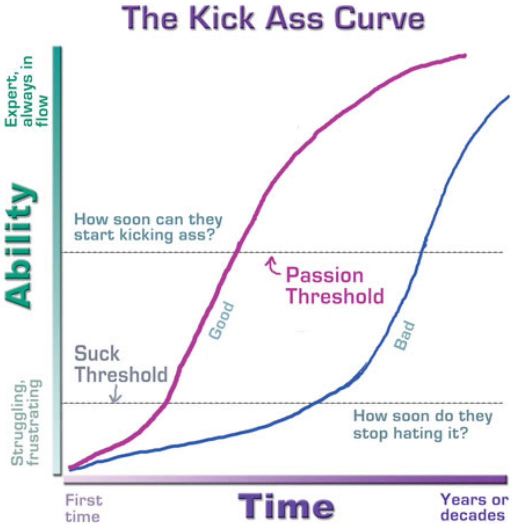


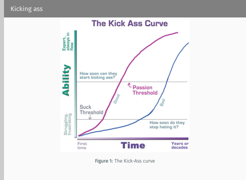

Figure 1: The Kick-Ass curve





2021-08-24

└Kicking ass



# Let's jump in

I'll show you ...

- scripts
- notebooks
- how to load data
- how to manipulate data
- how to plot data
- how to conduct a statistical test









Let's jump in

I'll show you ...

- scripts
- notebooks
- how to load data
- how to manipulate data
- how to plot data
- how to conduct a statistical test

# Rstudio tour

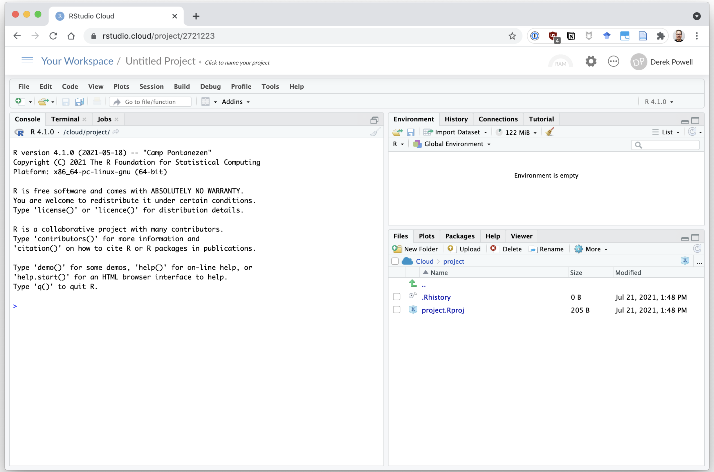



Figure 2: The Rstudio interface





Rstudio tour

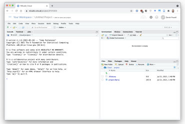


Figure 2: The Rstudio interface

# Interactive coding demo



Interactive coding demo

# Interactive coding demo

Let’s jump over to Rstudio, open a script, and run some code.





Interactive coding demo

Let’s jump over to Rstudio, open a script, and run some code.

R “packages” are collections of code, functions, and data that have been packaged together. Typically, packages support conducting certain types of analyses, making plots, etc.

The most important set of packages we will learn in this course are the `tidyverse` packages. We can load all of them at once with the command below:

```
library(tidyverse)
```





Loading packages

R “packages” are collections of code, functions, and data that have been packaged together. Typically, packages support conducting certain types of analyses, making plots, etc.

The most important set of packages we will learn in this course are the tidyverse packages. We can load all of them at once with the command below:

library(tidyverse)

## Loading data

```r
df <- read_csv("stroop-2014.csv")
```

- Load data from Daniel Lakens' introductory psychology course.
- Students performed the classic "Stroop" task: name ink color that congruent or incongruent words were printed in
  - (e.g. The word "Green" printed in green versus yellow ink).
- Reading interferes with color naming, so it takes longer to name the ink color when the word it is printing is "incongruent"





Loading data
df <- read_csv("stroop-2014.csv")
Load data from Daniel Lakens' introductory psychology course.
Students performed the classic "Stroop" task: name ink color that congruent or incongruent words were printed in
(e.g. The word "Green" printed in green versus yellow ink).
Reading interferes with color naming, so it takes longer to name the ink color when the word it is printing is "incongruent"

# Inspecting data with **head()**

We can inspect our data with the **head()** function.

```r
head(df)
```

```
## # A tibble: 6 x 3
##   subj_num congruent incongruent
##      <dbl>     <dbl>       <dbl>
## 1        1     13.7         22.7
## 2        2     14.8         25.9
## 3        3     16.8         25.7
## 4        4     11.9         21.2
## 5        5     10.5         22.6
## 6        6      9.44        20.7
```

# Indexing from dataframes

Our `df` object is a dataframe that consists of multiple variables. We can “index” or extract those variables using the `$` operator. Let’s pull out the congruent trial data.

```
df$congruent
```

```
##  [1] 13.741 14.788 16.819 11.888 10.516  9.436 13.256 18.643 15.827 13.000
## [11] 17.600 19.065 16.387 19.765 10.281 16.702 17.465 19.040 22.678 15.767
## [21] 14.445 16.888 10.034 10.091 16.460 15.721 10.900 17.196 21.392 17.725
## [31] 23.204 15.926 16.618 14.850 13.003 15.247  9.396 11.998 15.615 22.891
## [41] 10.825 14.690 21.000 18.363  9.733 14.563 12.162 13.076
```

# Using and defining functions

We can compute the mean and standard deviation of our congruent trials with:

```
mean(df$congruent)
```

```
## [1] 15.34742
```

```
sd(df$congruent)
```

```
## [1] 3.700337
```

Note that `mean()` and `sd()` are *functions* that perform some computation on whatever inputs they are given.

R includes many built-in functions, but sometimes we would like to do something new and unique. In those cases, we can define our own functions.





Using and defining functions
We can compute the mean and standard deviation of our congruent trials with:
mean(df$congruent)
## [1] 15.34742
sd(df$congruent)
## [1] 3.700337
Note that mean() and sd() are functions that perform some computation on whatever inputs they are given.
R includes many built-in functions, but sometimes we would like to do something new and unique. In those cases, we can define our own functions.

# Defining functions

One major omission in R is a built-in function to compute the standard error of a variable. R has a function `sd()` to compute the standard deviation, but nothing for the standard error. Recall, the formula for the standard error is:

$$SE = \frac{\sigma}{\sqrt{n}}$$

Let's make our own standard error function in R, named `se()`.

```
se <- function(x){
  sd(x) / sqrt(length(x))
  }
```

To learn more about this function, try typing `?sqrt` and `?length` into the R console.





Defining functions

One major omission in R is a built-in function to compute the standard error of a variable. R has a function sd() to compute the standard deviation, but nothing for the standard error. Recall, the formula for the standard error is:

SE = σ/√n

Let's make our own standard error function in R, named se().

se <- function(x){ sd(x) / sqrt(length(x)) }

To learn more about this function, try typing ?sqrt and ?length into the R console.

Try computing each of the components of this function, sd, sqrt, and length

# Computing standard error

Now we can use our function to calculate the standard error.

Using our function

```r
se(df$congruent)
```

```
## [1] 0.5340977
```

Checking each piece

```r
sd(df$congruent)
```

```
## [1] 3.700337
```

```r
length(df$congruent)
```

```
## [1] 48
```

```r
sqrt(48)
```

```
## [1] 6.928203
```

```r
3.700337 / 6.928203
```

```
## [1] 0.5340977
```





Computing standard error
Now we can use our function to calculate the standard error.
Checking each piece
sd(df$congruent)
## [1] 3.700337
length(df$congruent)
Using our function
se(df$congruent)
## [1] 48
## [1] 0.5340977
sqrt(48)
## [1] 6.928203
3.700337 / 6.928203
## [1] 0.5340977

# Hands-on: Making your own function

Let's make a function to standardize the `congruent` and `incongruent` variables.

Recall our standard error equation and corresponding R function.

$$SE = \frac{\sigma}{\sqrt{n}}$$

```
se <- function(x){
  sd(x) / sqrt(length(x))
  }
```

How would you create a function to standardize a variable based on the equation?

$$Z = \frac{x - \mu}{\sigma}$$





Hands-on: Making your own function
Let's make a function to standardize the congruent and incongruent variables.
Recall our standard error equation and corresponding R function.
How would you create a function to standardize a variable based on the equation?

# Working with data



Working with data

# Checking for missing values

First let's check if we are missing any values.

```
any(is.na(df))
```

```
## [1] FALSE
```

It looks like we aren't missing any data.

1. I passed our `df` object to the `is.na()` function, which will gave a matrix of `TRUE/FALSE` values indicating whether the value was missing or not.
2. Then I passed the output of `is.na()` into the `any()` function, which tells us if any of the values we give it are `TRUE`.





Checking for missing values
First let's check if we are missing any values.
any(is.na(df))
## [1] FALSE
It looks like we aren't missing any data.
1. I passed our df object to the is.na() function, which will gave a matrix of TRUE/FALSE values indicating whether the value was missing or not.
2. Then I passed the output of is.na() into the any() function, which tells us if any of the values we give it are TRUE.

# Removing cases with `filter()`

Suppose participant #3 called us up with a confession: they were really distracted the day of the experiment and could hardly pay any attention. Let's remove them from our dataset.

We will use the "pipe" operator `%>%` to pipe our data (`df`) to the `filter()` function. We can compare two variables with `==`, `!=`, `>`, `<`, `>=`, and `<=`.

```
df %>%
  filter(subj_num != 3)
```

```
## # A tibble: 47 x 3
##   subj_num congruent incongruent
##      <dbl>     <dbl>       <dbl>
## 1        1      13.7        22.7
## 2        2      14.8        25.9
## 3        4      11.9        21.2
## 4        5      10.5        22.6
## # ... with 43 more rows
```





Removing cases with filter()
Suppose participant #3 called us up with a confession: they were really distracted the day of the experiment and could hardly pay any attention. Let's remove them from our dataset.
We will use the "pipe" operator %>% to pipe our data (df) to the filter() function. We can compare two variables with ==, !=, >, <, >=, and <=.
df %>%
  filter(subj_num != 3)
## # A tibble: 47 x 3
##   subj_num congruent incongruent
##      <dbl>     <dbl>       <dbl>
## 1        1      13.7        22.7
## 2        2      14.8        25.9
## 3        4      11.9        21.2
## 4        5      10.5        22.6
## # ... with 43 more rows

# Creating new variables with `mutate()`

Things might feel more scientific if we store our response time measures as milliseconds (such precision!). We can compute new variables with the `mutate()` function.

```
df %>%
  mutate(
    congruent_ms = congruent*1000,
    incongruent_ms = incongruent*1000
  )
```

```
## # A tibble: 48 x 5
##   subj_num congruent incongruent congruent_ms incongruent_ms
##      <dbl>     <dbl>       <dbl>        <dbl>          <dbl>
## 1        1     13.7         22.7        13741          22715
## 2        2     14.8         25.9        14788          25916
## 3        3     16.8         25.7        16819          25677
## 4        4     11.9         21.2        11888          21213
## 5        5     10.5         22.6        10516          22556
## 6        6      9.44        20.7         9436          20715
## # ... with 42 more rows
```

# Renaming variables with rename()

Maybe we'd like to capitalize our condition name variables.

```
df %>%
  rename(Incongruent = incongruent, Congruent = congruent)
```

```
## # A tibble: 48 x 3
##   subj_num Congruent Incongruent
##      <dbl>     <dbl>       <dbl>
## 1        1     13.7         22.7
## 2        2     14.8         25.9
## 3        3     16.8         25.7
## 4        4     11.9         21.2
## 5        5     10.5         22.6
## 6        6      9.44        20.7
## # ... with 42 more rows
```





Renaming variables with rename()

Maybe we'd like to capitalize our condition name variables.

df %>%
  rename(Incongruent = incongruent, Congruent = congruent)

## # A tibble: 48 x 3
##   subj_num Congruent Incongruent
##      <dbl>     <dbl>       <dbl>
## 1        1     13.7         22.7
## 2        2     14.8         25.9
## 3        3     16.8         25.7
## 4        4     11.9         21.2
## 5        5     10.5         22.6
## 6        6      9.44        20.7
## # ... with 42 more rows

# Stringing together actions with %>%

- The `%>%` operator pipes the output of one function into the next function as its first argument.
- For example,

```
any(is.na(df))
```

is the same as

```
df %>% is.na() %>% any()
```

- This lets us string commands together, as shown here

Example

```
df %>%
  filter(subj_num != 3) %>%
  mutate(
    congruent_ms = congruent*1000,
    incongruent_ms = incongruent*1000
  ) %>%
  rename(
    Incongruent = incongruent,
    Congruent = congruent
    )
```





Stringing together actions with %>%
The %>% operator pipes the output of one function into the next function as its first argument.
For example,
any(is.na(df))
is the same as
df %>% is.na() %>% any()
This lets us string commands together, as shown here
Example

The same data can be stored in different formats.

- **Wide format**: each row of a data table is a case with many variables.
- **Long format**: each row stores the value for one variable of one case.

### Wide format

| case | x | y |
|---|---|---|
| a | 2 | 4 |
| b | 3 | 6 |

### Long format

| case | variable | value |
|---|---|---|
| a | x | 2 |
| a | y | 4 |
| b | x | 2 |
| b | y | 6 |





Tidying data

Wide format

The same data can be stored in different formats.
Wide format: each row of a data table is a case with many variables.
Long format: each row stores the value for one variable of one case.

Long format

# From wide to long with `gather()`

Let's convert our data from wide to long format using the `gather()` function.

```
df_long <- df %>%
  gather(trial_type, rt, congruent, incongruent)
```

```
## # A tibble: 96 x 3
##   subj_num trial_type     rt
##      <dbl> <chr>       <dbl>
## 1        1 congruent    13.7
## 2        1 incongruent  22.7
## 3        2 congruent    14.8
## 4        2 incongruent  25.9
## 5        3 congruent    16.8
## 6        3 incongruent  25.7
## # ... with 90 more rows
```



From wide to long with gather()
Let's convert our data from wide to long format using the gather() function.
df_long <- df %>%
  gather(trial_type, rt, congruent, incongruent)
## # A tibble: 96 x 3
##   subj_num trial_type     rt
##      <dbl> <chr>       <dbl>
## 1        1 congruent    13.7
## 2        1 incongruent  22.7
## 3        2 congruent    14.8
## 4        2 incongruent  25.9
## 5        3 congruent    16.8
## 6        3 incongruent  25.7
## # ... with 90 more rows

# From long to wide with spread()

Sometimes wide data is what we want. We can convert long data to wide data with the spread() function.

```
df_wide <- df_long %>%
  spread(trial_type, rt)
```

```
## # A tibble: 48 x 3
##   subj_num congruent incongruent
##      <dbl>     <dbl>       <dbl>
## 1        1     13.7         22.7
## 2        2     14.8         25.9
## 3        3     16.8         25.7
## 4        4     11.9         21.2
## 5        5     10.5         22.6
## 6        6      9.44        20.7
## # ... with 42 more rows
```





From long to wide with spread()
Sometimes wide data is what we want. We can convert long data to wide data with the spread() function.
df_wide <- df_long %>%
spread(trial_type, rt)
## # A tibble: 48 x 3
## subj_num congruent incongruent
## <dbl> <dbl> <dbl>
## 1 1 13.7 22.7
## 2 2 14.8 25.9
## 3 3 16.8 25.7
## 4 4 11.9 21.2
## 5 5 10.5 22.6
## 6 6 9.44 20.7
## # ... with 42 more rows

# Pivot functions

A recent update to the `tidyverse` is the introduction of the `pivot_longer()` and `pivot_wider()` functions.

- `pivot_longer()` = fancier but more complicated `gather()`
- `pivot_wider()` = fancier but more complicated `spread()`

I don't expect we will benefit from the extra complexity of `pivot_wider()` and `pivot_longer()` for most of what we do in this class.





Pivot functions
A recent update to the tidyverse is the introduction of the pivot_longer() and pivot_wider() functions.
pivot_longer() = fancier but more complicated gather()
pivot_wider() = fancier but more complicated spread()
I don't expect we will benefit from the extra complexity of pivot_wider() and pivot_longer() for most of what we do in this class.

# Summarizing data

- `group_by()` breaks the data into groups
- `summarize()` can be used with functions that take a vector input (from each group) and output a single number

```
df_long %>%
  group_by(trial_type) %>%
  summarize(mean_rt = mean(rt), se_rt = se(rt))
```

```
## # A tibble: 2 x 3
##   trial_type  mean_rt se_rt
##   <chr>         <dbl> <dbl>
## 1 congruent      15.3 0.534
## 2 incongruent    23.5 0.715
```





Summarizing data
- group_by() breaks the data into groups
- summarize() can be used with functions that take a vector input (from each group) and output a single number
df_long %>%
  group_by(trial_type) %>%
  summarize(mean_rt = mean(rt), se_rt = se(rt))
## # A tibble: 2 x 3
##   trial_type  mean_rt se_rt
##   <chr>         <dbl> <dbl>
## 1 congruent      15.3 0.534
## 2 incongruent    23.5 0.715

# Plotting data

Introduction to R

Plotting data

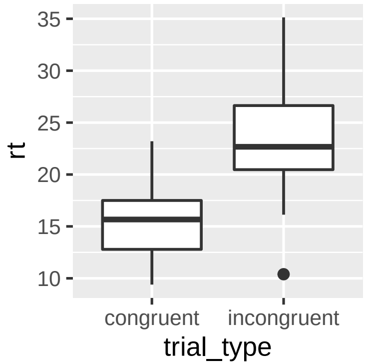
2021-08-24

Plotting data

```r
df_long %>%
  ggplot(aes(x=trial_type, y=rt)) +
  geom_boxplot()
```

Key **`ggplot()`** functions

- `ggplot()`: create a plot object
- `aes()`: specify the mapping of variables to plot elements
- `geom_boxplot()` (and other `geom_`s): draw something on the plot
- Note that ggplot elements are added to one another with `+`



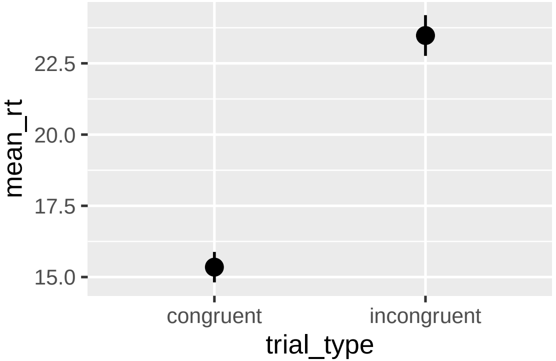





Basic plotting
df_long %>%
ggplot(aes(x=trial_type, y=rt)) +
geom_boxplot()
Key ggplot() functions
ggplot(): create a plot object
aes(): specify the mapping of variables to plot elements
geom_boxplot() (and other geom_s): draw something on the plot
Note that ggplot elements are added to one another with +

# Plotting summarized data

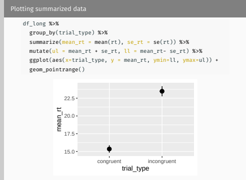
```
df_long %>%
  group_by(trial_type) %>%
  summarize(mean_rt = mean(rt), se_rt = se(rt)) %>%
  mutate(ul = mean_rt + se_rt, ll = mean_rt- se_rt) %>%
  ggplot(aes(x=trial_type, y = mean_rt, ymin=ll, ymax=ul)) +
  geom_pointrange()
```








2021-08-24



# Creating publication-quality figures (1)

```
df_long %>%
  group_by(trial_type) %>%
  summarize(
    mean_rt = mean(rt),
    se_rt = se(rt)
  ) %>%
  mutate(
    ul = mean_rt + 1.96 * se_rt,
    ll = mean_rt - 1.96 * se_rt
  ) %>%
  ggplot(
    aes(
      x = trial_type, y = mean_rt,
      ymin = ll, ymax = ul
    )
  ) +
  geom_bar(
    stat = "identity", width = .5, fill = "grey"
    ) +
  geom_errorbar(width = .1) +
  theme_bw(base_size = 28) +
  theme(panel.grid = element_blank()) +
  labs(
    x = "Trial Type",
    y = "Response time (s)"
  )
```

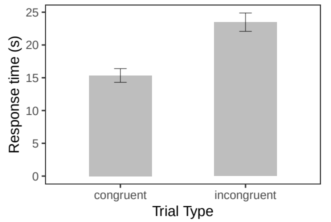
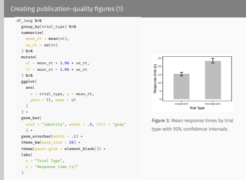



Figure 3: Mean response times by trial type with 95% confidence intervals





2021-08-24



```r
corr_val <- cor(df$congruent, df$incongruent)

df %>%
ggplot(aes(x = congruent, y = incongruent)) +
  geom_smooth(method="lm", alpha=.2) +
  geom_point() +
  annotate(
    "text",
    x = 20, y = 13,
    label=paste("r =", round(corr_val,3)),
    size = 6) +
  theme_bw(base_size=16) +
  theme(
    aspect.ratio=1,
    panel.grid = element_blank()
    ) +
  labs(
    x = "Congruent RT (s)",
    y = "Incongruent RT (s)"
    )
```

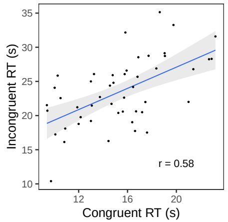
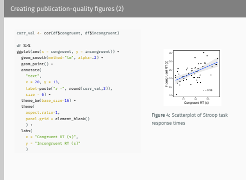



Figure 4: Scatterplot of Stroop task response times







# Doing statistics





Doing statistics

# Performing a t-test

```r
t.test(df$congruent, df$incongruent, paired=TRUE)
```

```
##
##  Paired t-test
##
## data:  df$congruent and df$incongruent
## t = -13.681, df = 47, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -9.324809 -6.933983
## sample estimates:
## mean of the differences
##               -8.129396
```





Performing a t-test

t.test(df$congruent, df$incongruent, paired=TRUE)

##
##  Paired t-test
##
## data:  df$congruent and df$incongruent
## t = -13.681, df = 47, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -9.324809 -6.933983
## sample estimates:
## mean of the differences
##               -8.129396

# Basic linear regression

```r
fit <- lm(incongruent ~ congruent, data=df)
summary(fit)
```

```
##
## Call:
## lm(formula = incongruent ~ congruent, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.7229 -2.5923  0.1305  2.6973  9.1001
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  11.5641     2.5353   4.561 3.77e-05 ***
## congruent     0.7762     0.1607   4.831 1.55e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.076 on 46 degrees of freedom
## Multiple R-squared:  0.3365, Adjusted R-squared:  0.3221
## F-statistic: 23.33 on 1 and 46 DF,  p-value: 1.549e-05
```

# Hands-on exercise: Data in the wild



Hands-on exercise: Data in the wild

# Hands-on exercise: Data in the wild

"How information about what is 'healthy' versus 'unhealthy' impacts children's consumption of otherwise identical foods" (DeJesus et al., 2019).

In Experiment 1 of this study:

- Researchers presented children with two food options: one "healthy" and one "unhealthy"
- Then they left children alone with the food and allowed them to eat if they wanted
- A research assistant observed through a camera and recorded the number of bites the children took
- After each child finished, researchers weighed the remaining food in the dish to calculate how many grams of food the children ate





Hands-on exercise: Data in the wild

"How information about what is 'healthy' versus 'unhealthy' impacts children's consumption of otherwise identical foods" (DeJesus et al., 2019).

In Experiment 1 of this study:

- Researchers presented children with two food options: one "healthy" and one "unhealthy"
- Then they left children alone with the food and allowed them to eat if they wanted
- A research assistant observed through a camera and recorded the number of bites the children took
- After each child finished, researchers weighed the remaining food in the dish to calculate how many grams of food the children ate

## Hands-on: Loading the data

**Load data from this paper yourself**

The data is stored in the file `dejesus-example.csv`.

**Data processing goals**

Let's create a new tibble called `dejesus_cleaned` where we

- Include only the second trials for each child from Experiment 1
- Rename variables to be a bit clearer
- Remove some unnecessary variables





Hands-on: Loading the data
Load data from this paper yourself
The data is stored in the file dejesus-example.csv.
Data processing goals
Let's create a new tibble called dejesus_cleaned where we
· Include only the second trials for each child from Experiment 1
· Rename variables to be a bit clearer
· Remove some unnecessary variables

# Hands-on: Cleaning things up

Variables we'd like

- `child_id`: participant number for each child
- `condition`: whether the child was told the food was healthy or unhealthy
- `bites`: how many bites the child took
- `grams`: how much of the food they ate (g)
- `age`: age of the children in years
- `gender`: gender of the children

Instructions

1. Look at the data with `head()`, `glimpse()` or `View()`
2. Filter for the 2nd trial from Exp. 1 using `filter()` (Exp. 1 is called "1_healthy_unh")
3. Rename variables to be a bit clearer using `rename()`
4. Remove unnecessary variables with `select()`

I didn't show you `select()`, take a look at the help with `?select` to learn about it.





Hands-on: Cleaning things up
Variables we'd like
- child_id: participant number for each child
- condition: whether the child was told the food was healthy or unhealthy
- bites: how many bites the child took
- grams: how much of the food they ate (g)
- age: age of the children in years
- gender: gender of the children
Instructions
1. Look at the data with head(), glimpse() or View()
2. Filter for the 2nd trial from Exp. 1 using filter() (Exp. 1 is called "1_healthy_unh")
3. Rename variables to be a bit clearer using rename()
4. Remove unnecessary variables with select()
I didn't show you select(), take a look at the help with ?select to learn about it.

- why bother renaming things? did you know what food_gen meant?
- presumably the authors do, but it could be hard to remember

# Hands-on: Inspecting the data

Now use the tools you've just learned to do some data sleuthing.

**Ask yourself**

- Are we missing data for any observations?
- Do all the observed values make sense?
- Can we fix or clean up these data? (how?)

**Remember**

- `head()` and `View()`
- `is.na()` and `any()`
- `filter()`
- plotting with `geom_point()`





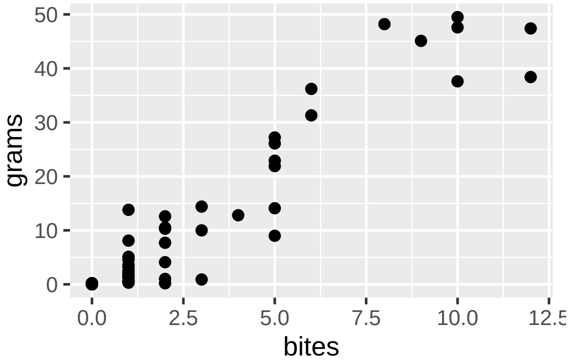
Hands-on: Inspecting the data
Now use the tools you've just learned to do some data sleuthing.
Ask yourself
- Are we missing data for any observations?
- Do all the observed values make sense?
- Can we fix or clean up these data? (how?)
Remember
- head() and View()
- is.na() and any()
- filter()
- plotting with geom_point()

Want them to: - check for NA - notice impossible values - maybe fix them - make a plot - do a test

# Hands-on: Inspecting with plotting

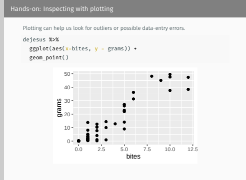
Plotting can help us look for outliers or possible data-entry errors.

```
dejesus %>%
  ggplot(aes(x=bites, y = grams)) +
  geom_point()
```







- A number of trials where children took zero bites show non-zero amounts of food being eaten according to grams
- These are most likely measurement errors, due to the inaccuracies of scales for small weights.
- If we like, we can fix these measurement errors using the `if_else()` function. The function takes 3 arguments, like so:

```
if_else(condition, value_if_true, value_if_false)
```

```
dejesus_cleaned <- dejesus %>%
  mutate(
    grams = if_else(bites==0, 0, grams)
    )
```





Hands-on: Fixing the dataset
- A number of trials where children took zero bites show non-zero amounts of food being eaten according to grams
- These are most likely measurement errors, due to the inaccuracies of scales for small weights.
- If we like, we can fix these measurement errors using the if_else() function. The function takes 3 arguments, like so:
if_else(condition, value_if_true, value_if_false)
dejesus_cleaned <- dejesus %>%
mutate(
grams = if_else(bites==0, 0, grams)
)

- scale is supposed to have .1g accuracy, which means 1sd to measurements. .1 to .2 is within 1-2sd. if we think a .5g bite is possible, then .3g observation could be 2sd off a .5g bite, rather than 3sd off a 0g bite. But it's hard to know without knowing the accuracy of the "bite" observers.
- We could treat this as a statistical inference problem, but lets save ourselves for more intresting issues. We can probably safely recode the .1 and .2 on zero bites as zeros. It's not as clear what to do with the .3 on 1 bite.

## Hands-on: Descriptives

- Calculate the gender and age breakdown of the children





Hands-on: Descriptives
• Calculate the gender and age breakdown of the children

```
summary(dejesus$age)
dejesus %>% count(gender)
```

# Hands-on: Plotting the data

- Plot the bites and/or grams eaten data by condition
- Try a few different approaches: `geom_boxplot()`, `geom_violin()`, `geom_jitter()`
- Try plotting mean with standard error bars





Hands-on: Plotting the data
- Plot the bites and/or grams eaten data by condition
- Try a few different approaches: geom_boxplot(), geom_violin(), geom_jitter()
- Try plotting mean with standard error bars

```
ggplot(dejesus, aes(x=grams)) + geom_histogram()
ggplot(dejesus) %>% aes(x=condition, y = bites) +
geom_jitter(width=.2)
```

# Hands-on: A basic linear model

- Use `lm()` to fit a simple linear regression model predicting grams eaten from number of bites the children took.
- From the output of the regression, what is the expected size of a bite in grams?
- Does the model's intercept make sense?





Hands-on: A basic linear model
- Use `lm()` to fit a simple linear regression model predicting grams eaten from number of bites the children took.
- From the output of the regression, what is the expected size of a bite in grams?
- Does the model's intercept make sense?

# Hands-on: A statistical test

- I already showed you a t-test, so try using a non-parametric test with `wilcox.test()` to compare the groups in terms of grams or bites





Hands-on: A statistical test
- I already showed you a t-test, so try using a non-parametric test with `wilcox.test()` to compare the groups in terms of grams or bites

# The path to kicking ass

- You have now seen all the basic workflow steps for doing statistics with R
  - loading data
  - processing data
  - plotting data
  - making models and performing tests





The path to kicking ass

- You have now seen all the basic workflow steps for doing statistics with R
  - loading data
  - processing data
  - plotting data
  - making models and performing tests

- discuss homework submission

# Extra slides



2021-08-24

Extra slides

The three commands below make the same changes.

```
df_long %>%
  mutate(trial_type = ifelse(trial_type=="congruent", "Congrent", "Incongruent"))

df_long %>%
  mutate(
    trial_type = fct_recode(
      trial_type,
      "Congruent"="congruent",
      "Incongruent"="incongruent")
    )

df_long %>%
  mutate(
    trial_type = case_when(
      trial_type=="congruent" ~ "Congruent",
      trial_type=="incongruent" ~ "Incongruent"
    )
  )
```