Day 5, Part 1: Report Data

Brennan Terhune-Cotter and Matt Dye

https://github.com/brennangitsit/2023_IAM3_R

Agenda

1. Summarizing data
2. Conducting common statistical tests
3. Reporting results
4. Tips for writing good code

Summarizing data

Summarizing a Dataset

```
??summarise
```

- The functions `summarise()` (or `summarize` across the pond) are great for summarizing data!
- If you want to summarize across groups, you use `group_by` first to group the data.
- Let's use a dataset called `asl_signs`, which has information about ASL signs and their frequency, iconicity, movement, handshape, etc.

asl_signs

entry_id	sign_frequency	iconicity	iconicity_type	lexical_class	handshape	selected_fin
tree	5.143	4.232	Perceptual	Noun	5	imrp
night	6.032	1.919	Arbitrary	Noun	flat_b	imrp
hamburger	4.429	3.714	Arbitrary	Noun	c	imrp
nephew	2.621	1.108	Arbitrary	Noun	flat_n	im
castle	1.579	3.540	Arbitrary	Noun	curved_v	im
humble	3.200	1.846	Arbitrary	Adjective	1	i
cup	5.742	2.897	Arbitrary	Noun	c	imrp
english	4.645	1.026	Arbitrary	Noun	c	imrp
dentist	2.677	3.923	Arbitrary	Noun	s	imrp
sandwich	3.677	2.538	Arbitrary	Noun	flat_b	imrp
monkey	2.619	6.014	Pantomimic	Noun	curved_5	imrp
chair	5.714	1.979	Arbitrary	Noun	h	im
candy_1	4.419	1.897	Arbitrary	Noun	1	i
wander	3.548	3.487	Arbitrary	Verb	1	i
scientist	3.516	1.410	Arbitrary	Noun	a	t
read	6.387	4.571	Perceptual	Verb	v	im

cat	5.097	4.618	Both	Noun	f	i
room	5.742	4.154	Perceptual	Noun	open_b	imrp
island	3.161	1.718	Arbitrary	Noun	i	p
paper	6.484	3.051	Arbitrary	Noun	5	imrp

Summarizing asl_signs

Let's summarize the *iconicity* variable, which is a score on a Likert scale of 1-7 (already summarized across respondents).

```
asl_signs %>%
  summarise(
    n = n(),
    mean_iconicity = mean(iconicity),
    stdev_iconicity = sd(iconicity),
    min_iconicity = min(iconicity),
    max_iconicity = max(iconicity)
  ) %>%
  kable()
```

n	mean_iconicity	stdev_iconicity	min_iconici
1768	NA	NA	N

Everything except n is NA! This is because we forgot to take care of NA values in our data.

Summarizing asl_signs

Let's summarize the *iconicity* variable, which is a score on a Likert scale of 1-7 (already summarized across respondents).

```
asl_signs %>%
  summarise(
    n = n(),
    mean_iconicity = mean(iconicity, na.rm =
    stdev_iconicity = sd(iconicity, na.rm = T
    min_iconicity = min(iconicity, na.rm = T)
    max_iconicity = max(iconicity, na.rm = T)
  ) %>%
  kable()
```

n	mean_iconicity	stdev_iconicity	min_iconicit
1768	2.948419	1.459429	

This data isn't very interesting unless we have a grouping factor of interest.

Summarizing asl_signs: Group cases

```
asl_signs %>%
  group_by(lexical_class) %>%
  summarise(
    n = n(),
    mean_iconicity = mean(iconicity, na.rm =
    stdev_iconicity = sd(iconicity, na.rm = T
    min_iconicity = min(iconicity, na.rm = T)
    max_iconicity = max(iconicity, na.rm = T)
  ) %>%
  kable()
```

lexical_class	n	mean_iconicity	stdev_iconicity
Adjective	274	2.554081	1.132531
Noun	912	2.748597	1.446377
Verb	582	3.449101	1.486408

We use `group_by()` to group a dataframe using a variable.

Summarizing asl_signs: Group cases

```
asl_signs %>%
  group_by(iconicity_type) %>%
  summarise(
    n = n(),
    mean_iconicity = mean(iconicity, na.rm =
    stdev_iconicity = sd(iconicity, na.rm = T
    min_iconicity = min(iconicity, na.rm = T)
    max_iconicity = max(iconicity, na.rm = T)
  ) %>%
  kable()
```

iconicity_type	n	mean_iconicity	stdev_iconici
Arbitrary	1415	2.344436	0.813015
Both	96	5.097688	0.791273
Pantomimic	145	5.698214	0.823368
Perceptual	112	5.142523	0.787789

We can group by different variables.

Summarizing asl_signs: Group cases

```
asl_signs %>%
  group_by(lexical_class, iconicity_type) %>%
  summarise(
    n = n(),
    mean_iconicity = mean(iconicity, na.rm =
    stdev_iconicity = sd(iconicity, na.rm = T
    min_iconicity = min(iconicity, na.rm = T)
    max_iconicity = max(iconicity, na.rm = T)
  ) %>%
  kable()
```

We can even group by two variables at once.

lexical_class	iconicity_type	n	mean_iconicity
Adjective	Arbitrary	245	2.261521
Adjective	Both	13	4.848923
Adjective	Pantomimic	8	5.157000
Adjective	Perceptual	8	5.145125
Noun	Arbitrary	752	2.198046
Noun	Both	51	4.992882
Noun	Pantomimic	62	5.793323
Noun	Perceptual	47	5.047106
Verb	Arbitrary	418	2.657444
Verb	Both	32	5.365781
Verb	Pantomimic	75	5.677320
Verb	Perceptual	57	5.222232

Summarizing asl_signs: Assign name

```r
asl_signs_summ <- asl_signs %>%
  group_by(lexical_class, iconicity_type) %>%
  summarise(
    n = n(),
    mean_iconicity = mean(iconicity, na.rm =
    stdev_iconicity = sd(iconicity, na.rm = T
    min_iconicity = min(iconicity, na.rm = T)
    max_iconicity = max(iconicity, na.rm = T)
  )

kable(asl_signs_summ)
```

To save the summarized data as an object, we assign it to a new object with the name `asl_signs_summ`.

lexical_class	iconicity_type	n	mean_iconicity
Adjective	Arbitrary	245	2.261521
Adjective	Both	13	4.848923
Adjective	Pantomimic	8	5.157000
Adjective	Perceptual	8	5.145125
Noun	Arbitrary	752	2.198046
Noun	Both	51	4.992882
Noun	Pantomimic	62	5.793323
Noun	Perceptual	47	5.047106
Verb	Arbitrary	418	2.657444
Verb	Both	32	5.365781
Verb	Pantomimic	75	5.677320
Verb	Perceptual	57	5.222232

Plotting Summarized Data

Raw vs. Summary geom_ Functions

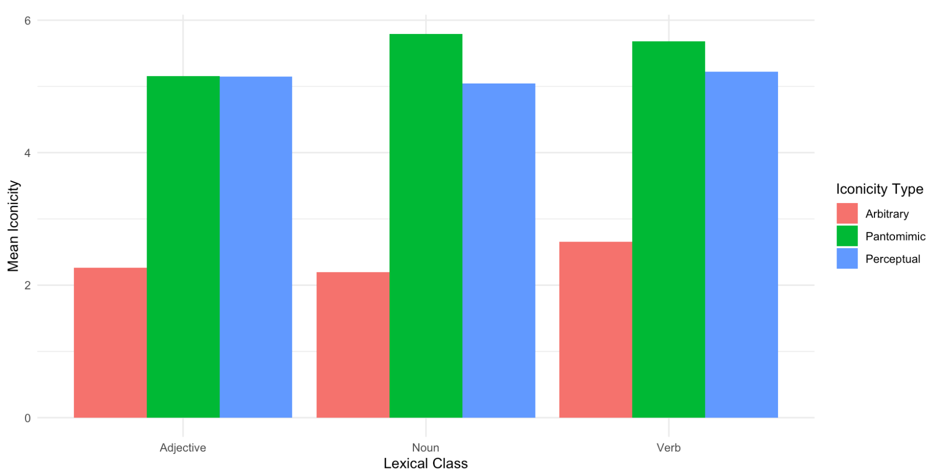
> **remember that ggplots are made by:**
>
> 1. specifying the dataset
> 2. specifying the aesthetic mappings
> 3. adding layers, especially geometric objects (**geom_...**) which display the data

- Some geometric objects display the **raw data** and require you to summarize it manually (`geom_col`, `geom_line`)
- Some geometric objects **summarize the data** for you (`geom_violin`, `geom_histogram`)
- Other special cases:
  - `geom_point()` displays the **raw data**
  - `geom_bar()` displays the **count** of **categorical data**

Plotting with Raw geom_ Functions

- Plotting summarized data with **raw geoms** is simple if you've made a summary dataset
- You just make a ggplot like we have been doing with raw data, but give it the **summary dataset**

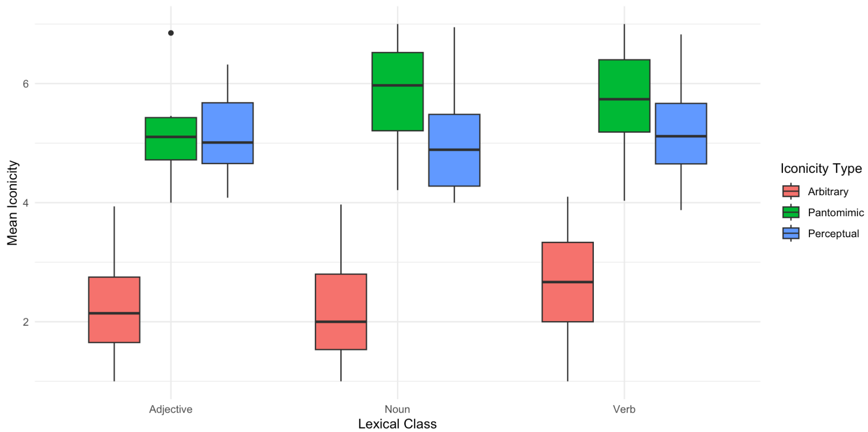
```
library(ggplot2)
asl_signs_summ %>%
  filter(iconicity_type != "Both") %>%
  ggplot(aes(x = lexical_class, y = mean_iconicity, fill = iconicity_type)) +
    geom_col(position="dodge") + # position = "dodge" gives me clustered barplots
    labs(x = "Lexical Class", y = "Mean Iconicity", fill = "Iconicity Type") +
    theme_minimal()
```



Plotting with Summary geom_ Functions

- Plotting summarized data with **summary geoms** is even simpler - make a ggplot with the raw dataset!
- The geom object summarizes the data for you. This is usually the case for geom objects that show **distribution**.

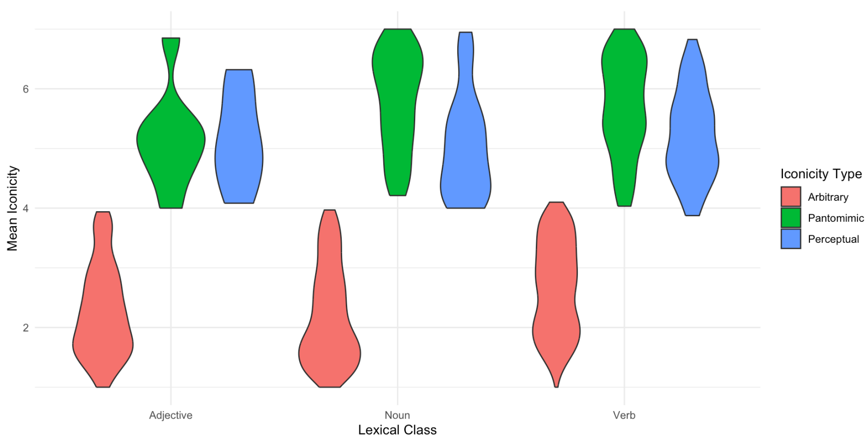
```r
asl_signs %>%
  filter(iconicity_type != "Both") %>%
  ggplot(aes(x = lexical_class, y = iconicity, fill = iconicity_type)) +
    geom_boxplot() +
    labs(x = "Lexical Class", y = "Mean Iconicity", fill = "Iconicity Type") +
    theme_minimal()
```



Plotting with Summary geom_ Functions

- Plotting summarized data with **summary geoms** is even simpler - make a ggplot with the raw dataset!
- The geom object summarizes the data for you. This is usually the case for geom objects that show **distribution**.

```
asl_signs %>%
  filter(iconicity_type != "Both") %>%
  ggplot(aes(x = lexical_class, y = iconicity, fill = iconicity_type)) +
    geom_violin() +
    labs(x = "Lexical Class", y = "Mean Iconicity", fill = "Iconicity Type") +
    theme_minimal()
```



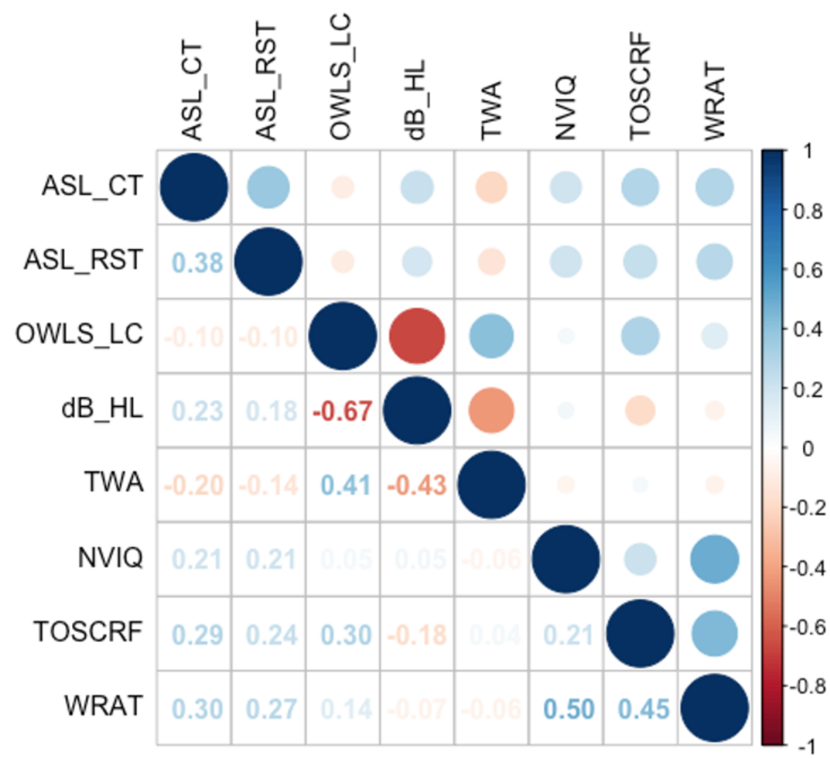
A note about viz...

> ggplot is not the only package that can create visualizations!
>
> for example, I created the correlations plot I showed you in **viz** with a package `corrplot::`

```r
# Impute missing values into correlations dat
library(mice) # For imputing missing values
filter <- stats::filter # fixes masked functi
md.pattern(corrs_df)
corr_imputation <- mice(corrs_df, m=5, maxit
complete_corrs_df <- complete(corr_imputation

# Calculate correlations
correlations <- cor(complete_corrs_df, method
correlations_sig <- cor.mtest(correlations, c

# Correlation plot
corrplot.mixed(correlations,
               tl.pos = 'lt',
               diag = 'u',
               tl.col = "black",
               p.mat = correlations_sig$p,
               sig_level = 0.50)
```



Statistical Testing

Statistical Tests & Models in R

- In the R ecosystem:
  - Statistical tests are *functions*, usually in specialized packages
  - They create *objects*, which are usually lists of lists of lists.
- You cannot view these objects directly; instead, you use *other* functions which look inside these objects and give you the output you like to see
  - These functions are usually called `summary()` or similar

```
my_anova <- aov(formula,
                data = my_data)
summary(my_anova)
```

⚠ these functions do not work with pipe (`%>%`) because the first argument is <u>not</u> the dataset!

Common Statistical Tests

1. **T-Tests and ANOVAs (Comparing Means):**
   - `t.test()`: Conducts a Student's t-test (two-samples and paired), which compares the means of two groups.
   - `aov()`: Conducts a one-way or multi-way ANOVA, used to compare the means of two or more groups.
2. **Regression and Correlation:**
   - `lm()` fits linear regression models; `glm()` fts generalized linear models.
   - `cor.test()`: Tests for correlation between two variables.
   - `lmer()` and `glmer()` (from the `lme4` package): Fit linear mixed-effects models, which are commonly used in linguistic research to account for random effects such as participant and item variability.
3. **Chi-Square Test:**
   - `chisq.test()`: Conducts a chi-square test of independence, used to examine the relationship between two categorical variables.
4. **Factor Analysis:**
   - `factanal()`: Performs a factor analysis, used in psychological research to identify underlying latent variables.

Running Statistical Tests in Functions

- Statistical tests in R are another step where redundancy can be an issue
- You may have to run the same test, with the same settings, multiple times
- This is risky in point-and-click programs and better, but annoying, in R
- You can use **functions** to streamline your testing scripts!

```
# Function to conduct an ANOVA within FLAD or
single_study_anova <- function(studyname, eff
  if(effect == "presence") {
    outputdf <- component %>%
      filter(flankers != "S" & study == study
      do(tidy(aov(value ~ group * flankers *
      mutate(sig = case_when(
        p.value < .001 ~ "***",
        p.value < .05 ~ "**",
        p.value < .1 ~ "*"
      )) %>%
      filter(term != "Residuals")
    return(outputdf)
  } else if (effect == "identity") {
    outputdf <- component %>%
      filter(flankers != "N" & study == study
      do(tidy(aov(value ~ group * flankers *
      mutate(sig = case_when(
        p.value < .001 ~ "***",
        p.value < .05 ~ "**",
        p.value < .1 ~ "*"
      )) %>%
      filter(term != "Residuals")
    return(outputdf)
```

Running a Paired T-Test

If we wanted to compare performance on test1 and times 1 and 2 (to see if scores change) from the tidy climate data we created yesterday, then we should run a "paired" t-test that takes into account the fact that the scores at time 1 and time 2 were obtained from the same individuals:

```
tidy_lang_data_complex <- readRDS("../../data/tidy_lang_data_complex.rds")
ttest_test1 <- t.test(test1 ~ time, data = tidy_lang_data_complex, paired = TRUE)
ttest_test1
```

```

	Paired t-test

data:  test1 by time
t = -3.9739, df = 34, p-value = 0.000349
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 -2.0295925 -0.6561218
sample estimates:
mean difference 
      -1.342857 
```

Reporting Data

R Markdown

- Researchers commonly use R Markdown or R Notebook to write reports
- Because the data and plots in these reports are from code, they will automatically update with new data every time you knit or render them 😎
  - No more rewriting results tables or remaking plots every time!
  - (You still have to rewrite your discussion and conclusions … for now 😉)

Creating Tables

- Packages exist for creating publication-ready tables
- I've been using `kable()` throughout this presentation to make pretty tables.
- There is also a package called `kableExtra`. From the author Zao Hu:

> The goal of `kableExtra` is to help you build common complex tables and manipulate table styles. It imports the pipe `%>%` symbol from `magrittr` and verbalize all the functions, so basically you can add "layers" to a kable output in a way that is similar with `ggplot2` and `plotly`.

How to Write Good Code

Commenting

- Commenting is your *inline documentation* of your code and analysis
- Especially as a beginning coder, there is no such thing as too little commenting
- Comments should:
  1. explain what your code is doing
  2. explain decisions you made and why
  3. not repeat the code, but clarify & contextualize it

Naming Variables and Objects

How you name variables and objects can make life much easier for you.

- Use long & descriptive variable or object names if you have to.
  - Text is cheap, brain capacity is not.
  - Which dataframe name is clearer?

    ```
    df3
    average_EEG_response_times
    ```

- Variables and objects should never have spaces or hyphens; use underscores instead.
  - Names with spaces or hyphens must be surrounded by `` ` ` `` every time you call them, which is super annoying.

Coding Style

- Don't use run-on code lines; most functions should start on a new line.
- Use blank lines often to separate code blocks! You can't have too many blank lines.
- Add spaces around operators: `+ - == < != <-` etc.
- Add spaces after comments like in English.

```
new_df <- mutate(df,
                 new_name = old_name,
                 new_name2 = old_name2,
                 new_name3 = old_name3)
newer_df <- filter(new_df,
                   group = "deaf")
```

Debugging

Other tips for coding in R

- Delete old objects you no longer need with `rm()`. This helps keep your environment clean.
- If you need to quote something, highlight it and press `"` or `'`. This also works with `(`.
- Use **sectioning comments** (`# Section title -----`) which allow you to "minimize" sections.

Now What?

- Your only **real** OYOLab!
- If you've imported your data into R, look at your data in R
  - Figure out what you want to do with it.
  - Write out some goals you have for your data.
  - Write pseudocode to figure out your goals.
  - Try writing code to work with your data!
  - Try writing visualizations to explore the data.
- If you don't have data, play around with ours!
  - Try select(), filter(), mutate()
  - Can you make some simple visualizations to explore the data?