Starting to Work in R

GLOCAL Workshop

Noah Vanderhoeven M.A.

PhD Student

University of Western Ontario

Introduction

The goals for this session are for participants to be able to...

- **Develop an understanding of how to explore and describe a dataset.**
- **Be able to understand the mathematical and logical language of R and make use of these operators in R Studio.**
- **Develop an understanding of tidy data practices that will allow them to manage data and prepare their dataframe for future work.**
- **Work with the dataframe to create new variables, manage missingness and create visualizations**

Loading our packages

- It seems to be best practice to load your packages at the start of your script, to help keep your script organized.
- Sometimes you will find that you need a package later on and you did not load it when you first started your script. Not a problem! You can load a package at any point.
- Once we have installed a package, it will be loaded under the packages window forever (or until we uninstall it or something else major happens to our RStudio).
- You will notice that we have included the `install.packages()` specifying "rio" but with a hashtag in front of it. This "comments out" the code.
- In other words, this is a way to write code that won't run when we execute our code. It is also a way to comment on our code - which is an important step for writing good code.

```
# install.packages("rio")
library(rio)
# We LOAD the rio() package which we use to import our data into R
```

Setting our working directory

- Your working directory is what R can access (e.g. what datasets it can load for you to work on). It follows the path of folders your computer stores things in.
- If you load a script with R studio closed beforehand your working directory will be the same as the file path where you retrieved the script from.
- If you want set a different working directory, to access other files without having to shut down R studio and lose all of your saved objects in your global environment, there are multiple ways to set our working directory.
- The point and click method involves clicking "Session" in the toolbar at the top of your screen and then clicking "Set working directory" and there are multiple options in that menu for setting it. Eg. set it "to source file location" meaning to the same location where this R Script is saved. "Choose Directory..." where you can navigate to this folder (similar to when you do "Save As" in Word etc.)

- Or you can also use the functions in the code chunk below

```
rm(list = ls()) # removes everything from memory

getwd() # will return your current working directory

setwd("C:/Users/noahv/OneDrive - The University of Western Ontario/Documents/P
# will set you working directory to be
# whatever you tell it to within the quotation marks
```

Please take a moment to run `getwd()` in your script to check that everything is in order.

- You want to make sure the R script you are working in and the sample dataset we have shared with you are stored in the same place. If they are not you won't be able to load the data and work with it. If you need to change your working directory try one of the methods below.

- The point and click method involves clicking "Session" in the toolbar at the top of your screen and then clicking "Set working directory" and there are multiple options in that menu for setting it. Eg. set it "to source file location" meaning to the same location where this R Script is saved. "Choose Directory..." where you can navigate to this folder (similar to when you do "Save As" in Word etc.)

```
rm(list = ls()) # removes everything from memory

getwd() # will return your current working directory

setwd("C:/Users/noahv/OneDrive - The University of Western Ontario/Documents/P
# will set you working directory to be
#  whatever you tell it to within the quotation marks
```

- Do not be hesitant to ask for assistance if something is amiss!

Loading our data

- We will now use the `import()` function from the **rio** package or `read.csv()` from base R package **utils** to import our data. This is the same data that we used earlier.

```
ces <- import(file = "ces_for_intro_to_r.csv")
# or
ces <- read.csv(file = "ces_for_intro_to_r.csv")
```

- Either way we load the data, if we don't "assign" (using the assign arrow <-) what we are doing to a name (object), R will not "store" or "save" our data for use. You should see "ces" listed under your global environment in the top right corner of your screen.

```r
# Remember that we can use the `View()` function to look at
# our data. Here I have # out the function so that it doesn't
# run. If you want it to run, remove the # in front of View


#View(ces)
```

Starting to explore and describe our data

`dim()` function for matrices and dataframes

We can use `dim()` to check the dimensions of a dataframe or matrix.

```
dim(ces)
```

```
## [1] 749  13
```

```
# The dimensions of this dataframe are:
# (remember, always rows x columns)
# 749 rows (observations) and 13 columns (variables).
```

`length()` function for vectors and the $, the selection operator

- Here we would like to learn about the length of the dob variable.
- We can select the dob variable using the $ operator.
- This is a common task: selecting a particular variable from a dataframe or matrix using the $ operator.

```
length(ces$dob)
```

```
## [1] 749
```

Types of variables

- There are four main types of variables you will deal with in R (we will talk about the fourth when we talk about logical/boolean values).
  - First is a numeric variable which deals with all real numbers (both those with and without decimals).
  - Second, there are integers. Integers are real numbers that do not have decimal points. The suffix L is used to specify integer data.
  - Character values are used to represent character or string values. Characters are generally considered single letters whereas a string is a set of letters.

What type of variable would you say the date of birth, dob, variable is?

`class()` function

- We can use the `class()` function to check class of objects in our global environment. This function, and many of the other functions we will talk about in this part of the workshop, are included in base R meaning we don't need to install a package in order to use these functions.
    - (Eg. the basic "Calculator" app on your phone allows you to add/subtract/multiply and perform other "functions").

```
class(ces)
```

```
## [1] "data.frame"
```

```
class(ces$dob)
```

```
## [1] "integer"
```

```
class(ces$educ)
```

```
## [1] "character"
```

`head()` function

We can use the `head()` function to look at the first 6 rows of the ces dataset, or the first 6 values of the DOB variable.

```
head(ces)
```

```
##     X  dob gender province      educ    relig                               mostimp
## 1  63 1998 Female       ON  Some uni    Other climate change/ carbon emmissions
## 2  93 1978   Male       QC  Some uni     None                              Economie
## 3 103 1972   Male       ON Bachelors Catholic                               ECONOMY
## 4 117 1954   Male       SK  Some uni Catholic                tax breaks for seniors
## 5 143 1972   Male       NS  Postgrad     None                                 Taxes
## 6 189 1954   Male       ON  Some uni Catholic          The economy and immgration
##   natret leftright feel_trudeau         vote        changefptp
## 1 Better         4           85      Liberal Strongly disagree
## 2  Worse         7           12        Green           Neutral
## 3   Same         7            0 Conservative           Neutral
## 4   Same         3           79      Liberal    Somewhat agree
## 5  Worse         7           21 Conservative    Somewhat agree
## 6  Worse         8           41 Conservative    Strongly agree
##        keepcarbontax
## 1     Strongly agree
## 2     Somewhat agree
## 3 Strongly disagree
## 4     Somewhat agree
## 5 Strongly disagree
## 6 Somewhat disagree
```

`unique()` function

The unique function allows us to look at the unique values in a variable.

```
unique(ces$educ)
```

```
## [1] "Some uni"   "Bachelors"  "Postgrad"   "HS or less"
```

Using indexing to select

Now, let's try selecting a specific variable from our dataframe so that we can look at it.

```r
# selecting the 5th variable from our dataset, education level
ces[5]
ces[[5]]
```

```
## [1] "Some uni"  "Some uni"  "Bachelors" "Some uni"  "Postgrad"  "Some uni"
```

- Since the ces object is a multi-dimensional object (dataframe with rows x columns), we can index (select) rows by columns to pull out a specific cell from our dataframe.

```
# for example, we will select the 5th value in the second column
ces[5,2]
```

```
## [1] 1972
```

- We could also look at this person's gender.

```
# Remember we are looking at the 5th observation.
# So let's select the 5th observation in the third column.
ces[5,3]
```

```
## [1] "Male"
```

Logical or boolean values

- Boolean values in R tell us whether a given expression is TRUE or FALSE.
- Here we are asking if there are any NAs and what this returns is a boolean value for each obs.
- FALSE meaning the value does not equal NA and TRUE if the value equals NA.

```
is.na(ces$gender)
```

- I wrapped this in head() to see only the first five NA obs.

```
head(is.na(ces$gender))
```

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

Starting to work with our data

The `factor()` function

- When we have categorical variables, we store them as a variable class called 'factor' in R.
- A factor cannot have decimal points. It can be either a character (string, think letters) or integer (numeric, as long as there are no decimal points).
- What differentiates a factor from an integer variable, for instance, is that a factor has "levels" or order.
- We don't know the distance between these levels, however. Whereas the numbers 1 and 2 have a quantifiable difference between them (1 point), a factor does not (eg. male, female).
- We COULD store the factor as numbers. Eg. we could code male as 1 and female as 2, but that does not mean that there is 1 point difference between them, we are simply using the numbers to store the categories.

By wrapping ces$gender in the `factor()` function, we are asking R to store the gender variable from the ces dataframe as a factor. We can see there are two unique categories or "levels" (male and female).

```
factor(ces$gender)
```

We could also wrap this in head() to see only the first 5 obs and the levels of the factor.

```
head(factor(ces$gender))
```

```
## [1] Female Male   Male   Male   Male   Male
## Levels: Female Male
```

```
head(factor(ces$vote))
```

```
## [1] Liberal      Green        Conservative Liberal      Conservative
## [6] Conservative
## Levels: Bloc Quebecois Conservative Green Liberal NDP Other PPC
```

Assignment operator <-

(Alt + - on Windows, Option + - for Mac)

We can use the assignment operator to assign values to an object. First, let's start by assigning some numbers to an object.

```
one <- 1
# we assigned the number 1 to an object called "one".
# Look in your global environment to see it saved.

numbers <- c(1,2,3,4,5) # here we create a vector of numbers 1-5.

numbers <- c(1:5)
# we could do the same thing above using a simpler method
# where the colon tells R "through", in this case, 1 through 5.
```

Assignment operator <-

We can also use the assignment operator to assign a variable from a df (a vector) to a new object name.

```
party <- factor(ces$vote)
#Let's check the class...it is a factor as we expected!
class(party)
```

```
## [1] "factor"
```

Arithmetic operators

```r
min(ces$feel_trudeau)
```

```
## [1] 0
```

```r
max(ces$feel_trudeau)
```

```
## [1] 100
```

```r
median(ces$feel_trudeau) # middle value
```

```
## [1] 56
```

```r
mean(ces$feel_trudeau)
```

```
## [1] 47.89453
```

```r
# another way to calculate the mean
sum(ces$feel_trudeau)/nrow(ces)
```

```
## [1] 47.89453
```

Arithmetic operators

```
3+2 #addition
```

```
## [1] 5
```

```
3*2 # multiplication
```

```
## [1] 6
```

```
3/2 # division
```

```
## [1] 1.5
```

```
(10+10)*2 # addition and multiplication
```

```
## [1] 40
```

Summary: Arithmetic operators

- Addition +
- Subtraction -
- Multiplication *
- Division /
- Exponent ^

Relational operators

```
class(ces$leftright)
```

```
## [1] "integer"
```

```
obs_two <- ces[2,10]
# use the assign operator to save this single cell value as a new object.

obs_three <- ces[3,10]

obs_two > obs_three
```

```
## [1] TRUE
```

```
# returns a boolean (true/false)
# telling us whether observation 2 is greater than observation 3
# on specifically the age variable (dob).

obs_five <- ces[5,15]
obs_five >= obs_three # greater than or equal to?
```

```
## logical(0)
```

Summary: Relational operators

- Less than <
- Greater than >
- Less than or equal to <=
- Greater than or equal to >=
- Equal to ==
- Not equal to !=

Missingness

```
ces$gendre
```

```
## NULL
```

```
# here it returns NULL because this doesnt exist.
# We've misspelled gender and so nothing returns.
```

```
ces[750,2]# NA = Not available
```

```
## [1] NA
```

```
0/0
```

```
## [1] NaN
```

```
# zero divided by zero: not a number/ impossible value
```

Summary: Types of missingness

- NA Not available
- NULL None
- NaN Not a number/impossible value.

A brief introduction to the tidyverse

- **Filter**
- **Arrange**
- **Select**
- **ggplot**

```r
# install.packages("tidyverse")
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──
## ✓ ggplot2 3.4.0      ✓ purrr   0.3.5
## ✓ tibble  3.1.8      ✓ dplyr   1.0.10
## ✓ tidyr   1.2.1      ✓ stringr 1.5.0
## ✓ readr   2.1.2      ✓ forcats 0.5.2

## Warning: package 'ggplot2' was built under R version 4.2.2

## Warning: package 'tidyr' was built under R version 4.2.2

## Warning: package 'purrr' was built under R version 4.2.2
```

Filter

- `filter()` includes all rows that fit into the rule applied to a specific column or columns.

```
ces1 <- ces %>%
  filter(leftright > 5)
summary(ces1$leftright)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   6.000   7.000   7.000   7.492   8.000  10.000
```

```
# Checking the number of rows or columns in the data versus the filtered data,
# filtering will usually reduce the number of rows
nrow(ces) - nrow(ces1)
```

```
## [1] 418
```

Binding

- We can also bind dataframes by columns if the rows are alike using `cbind()` or by the rows using `rbind()` if the columns are matching perfectly.

```
ces2 <- ces %>%
  filter(leftright <= 5)
# binding the two filtered dataframes by row
ces3 <- rbind(ces1, ces2)
# Checking the number of rows or columns in the original data versus
# the merged filtered dataframes
nrow(ces) - nrow(ces3)
```

```
## [1] 0
```

- We will see an example of using `cbind()` when we discuss `select()`

Arrange

- By default, `arrange()` will sort the vector on ascending order.

```
ces1 <- ces1 %>%
arrange(leftright)
head(ces1$leftright)
```

```
## [1] 6 6 6 6 6 6
```

- Descending order

```
ces1 <- ces1 %>%
  arrange(desc(leftright))
head(ces1$leftright)
```

```
## [1] 10 10 10 10 10 10
```

Select

- Using `select()`, you will reduce your original number of columns into a shorter total by picking out certain variables.

```
ces4 <- ces %>%
  dplyr::select(gender, dob, province, leftright)
ncol(ces) - ncol(ces4)
```

```
## [1] 9
```

- We can use `cbind()` to add a variable from our original dataset to this more selective dataset.

```
# selecting the vote variable from our original dataset
ces5 <- ces %>%
  dplyr::select(vote)
# using cbind to add this variable to our more selective dataset
ces6 <- cbind(ces4, ces5)
ncol(ces6) - ncol(ces4)
```

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

Tidy data practices

- Ignoring the commands, the tidyverse comes down to the concept of Tidy Data.
- The goal is your data are arranged so each row contains an observation, and each column contains a variable about that observation.

```
head(ces6)
```

```
##   gender  dob province leftright         vote
## 1 Female 1998       ON         4      Liberal
## 2   Male 1978       QC         7        Green
## 3   Male 1972       ON         7 Conservative
## 4   Male 1954       SK         3      Liberal
## 5   Male 1972       NS         7 Conservative
## 6   Male 1954       ON         8 Conservative
```

- Data organized this way are considered to be wide data, which is useful because the pivot_() family of functions can be used to transform the data from long to wide data and vice versa. This will be quite useful as you add more skills to your R toolkit and learn to make graphs and perform statistical analysis.

Creating a new variable

- We can use base R to create a new variable using our existing data.
- For this we will use the square brackets [] to set the values we are looking for in our data to create the new variable. Here we will create a new variable called ROC (for Rest of Canada) that takes on a value of 1 when a province is not Quebec and a value of NA for the province of Quebec. To do this we will use the existing province variable that covers where a participant lives.

```
# creating a vector for our new variable in our dataframe
ces$ROC <- NA
# using base R to create a Rest of Canada Variable
ces$ROC[ces$province %in% c("AB", "BC", "MB", "NB", "NL",
                            "NS", "ON", "PE", "SK")] <- 1
# coding Quebec to be equal to an NA character
ces$ROC[ces$province == "QC"] <- NA
# using table to check that we created the variable correctly
table(ces$ROC)
table(ces$province)
```

Filtering out NAs

- Now we will use the ROC variable to practice filtering out missing values or NAs.
- NAs can disrupt our calculations and dealing with them in a reasonable way is important for ethical and effective data visualization. If data is missing at random we can remove it without too much concern that this will bias our results. If it is not missing at random than we have to think more about what removing all misssing values may bias our results.

```
# filtering out all NAs from the entire dataframe by row
ces_1 <- na.omit(ces)
# filtering out the NAs by row based on a single column
ces_2 <- ces %>%
  filter(ROC != is.na(ROC))
```

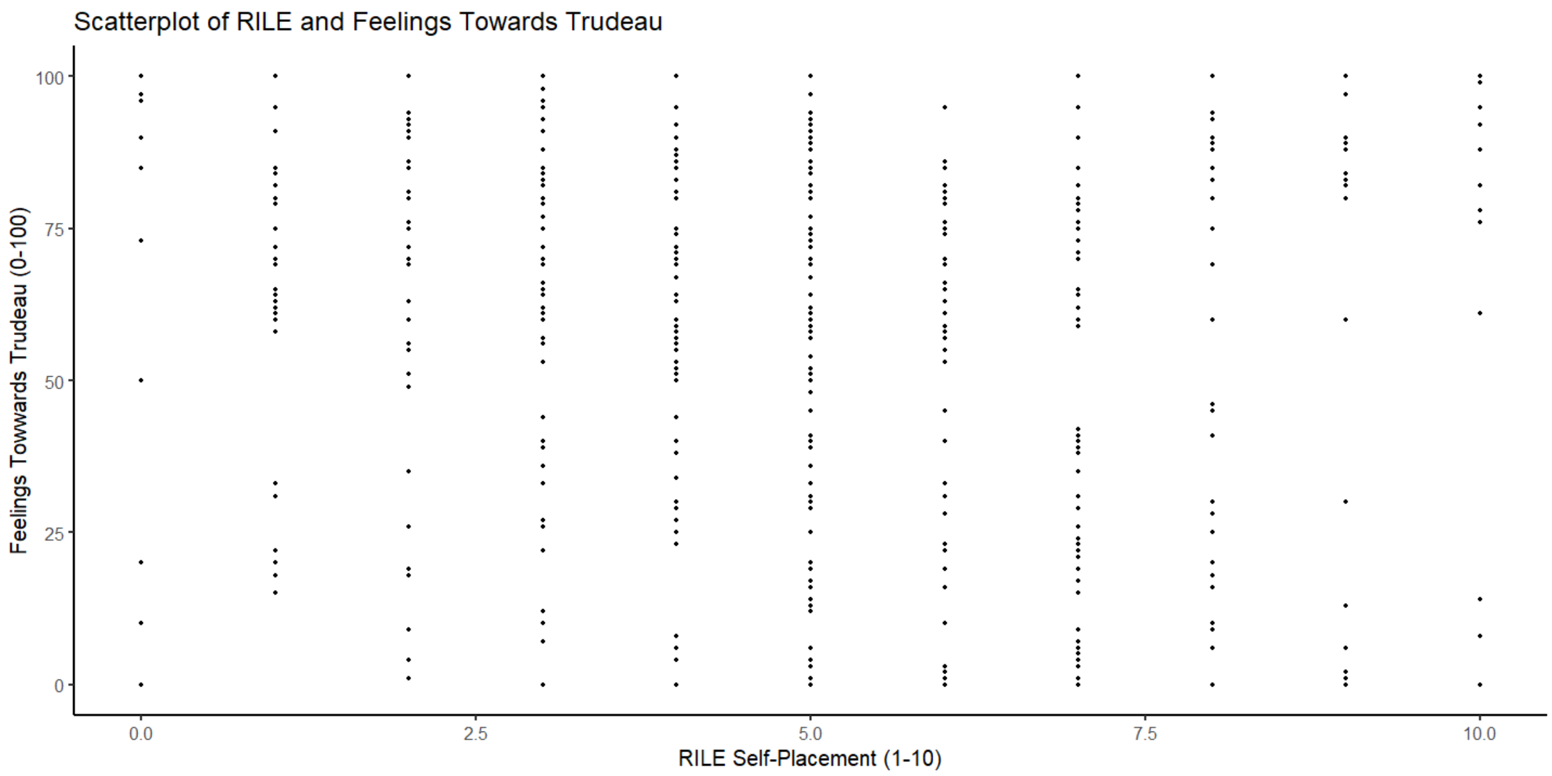
- These two methods of removing NAs end up with the same dataframe but they do so in different ways that have their uses for specific problems or tasks.

Making a scatterplot with ggplot2

- `ggplot()` is a package that can allow us to make engaging data visualizations. Here we can make a scatter plot of the effect someone's Left-Right ideological placement has on their feelings towards Prime Minister Justin Trudeau before the 2021 federal election.
- leftright is a uni-dimensional measure of political ideology, where 1 represents someone who is very left-leaning and 10 represents someone who is very right-leaning. feel_trudeau is a feeling thermometer where 0 indicates someone with very negative feelings about PM Trudeau and 100 represents someone who has very positive feelings about PM Trudeau.

```
ggplot(ces_2, aes(leftright, feel_trudeau)) +
  # adding our data, and then x and y variables
  # using the point geometry to create a scatterplot
  geom_point() +
  # adding labels and a title
  labs(title =  "Scatterplot of RILE and Feelings Towards Trudeau",
       x = "RILE Self-Placement (1-10)",
       y = "Feelings Towwards Trudeau (0-100)") +
  # using a theme to make our graph visually clear to present our data
  theme_classic(base_size = 20)
```

Making a scatterplot with ggplot2



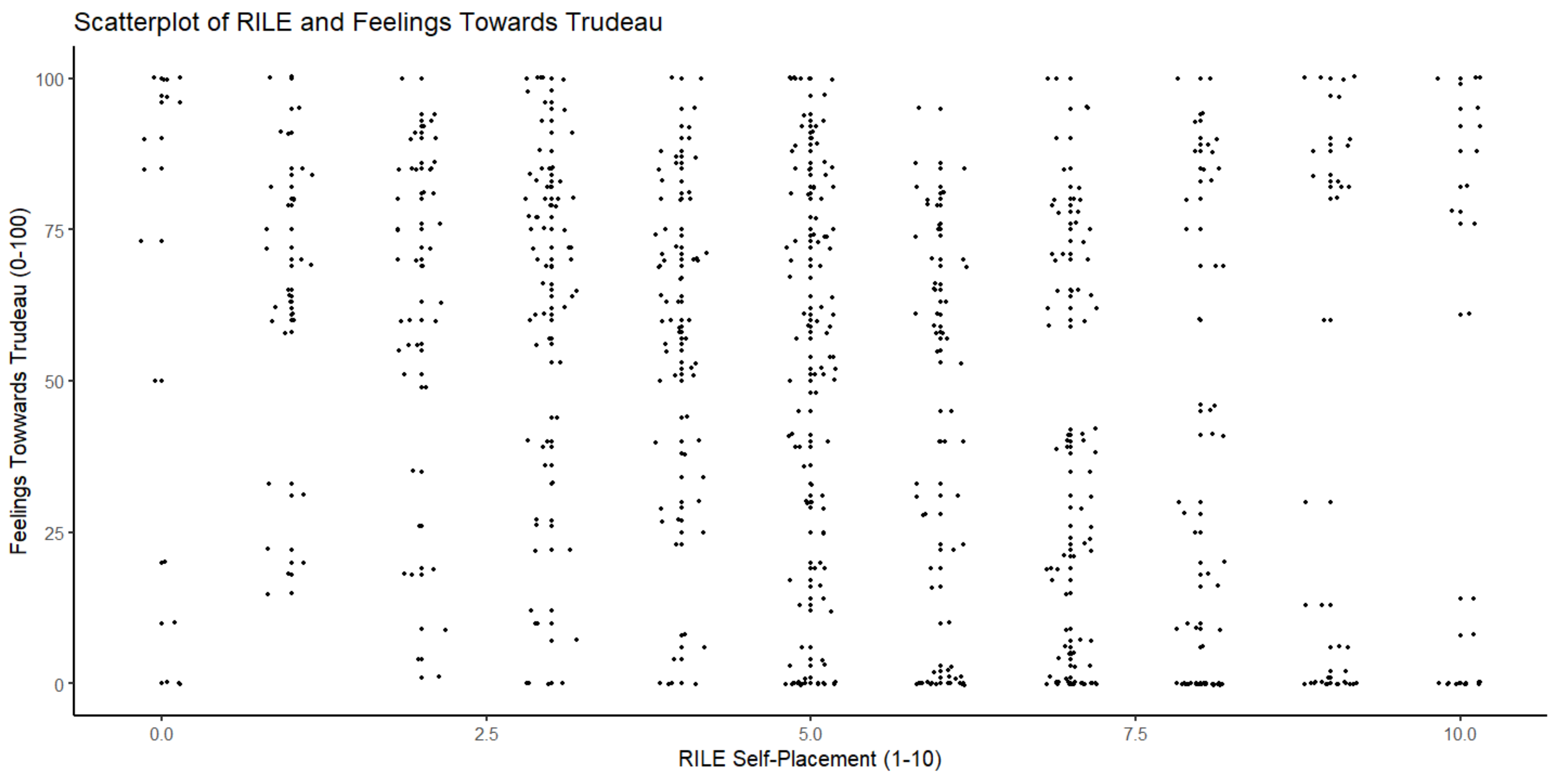
- What do we think the relationship is?

Using Jitter to make our data visualization easier to read

- `geom_jitter()` is a geom object that will slightly adjust how our data is presented so that we can see overlapping points easier and get a better sense for the density of our variables on our plot. From the previous graph it may have been hard to say what this relationship looked like.

```
ggplot(ces_2, aes(leftright, feel_trudeau)) +
  geom_point() +
  # using jitter to shift our overlapping points to understand the density
  geom_jitter(width = 0.2, height = 0.2) +
  labs(title =  "Scatterplot of RILE and Feelings Towards Trudeau",
       x = "RILE Self-Placement (1-10)",
       y = "Feelings Towwards Trudeau (0-100)") +
  theme_classic(base_size = 20)
```

Using Jitter to make our data visualization easier to read



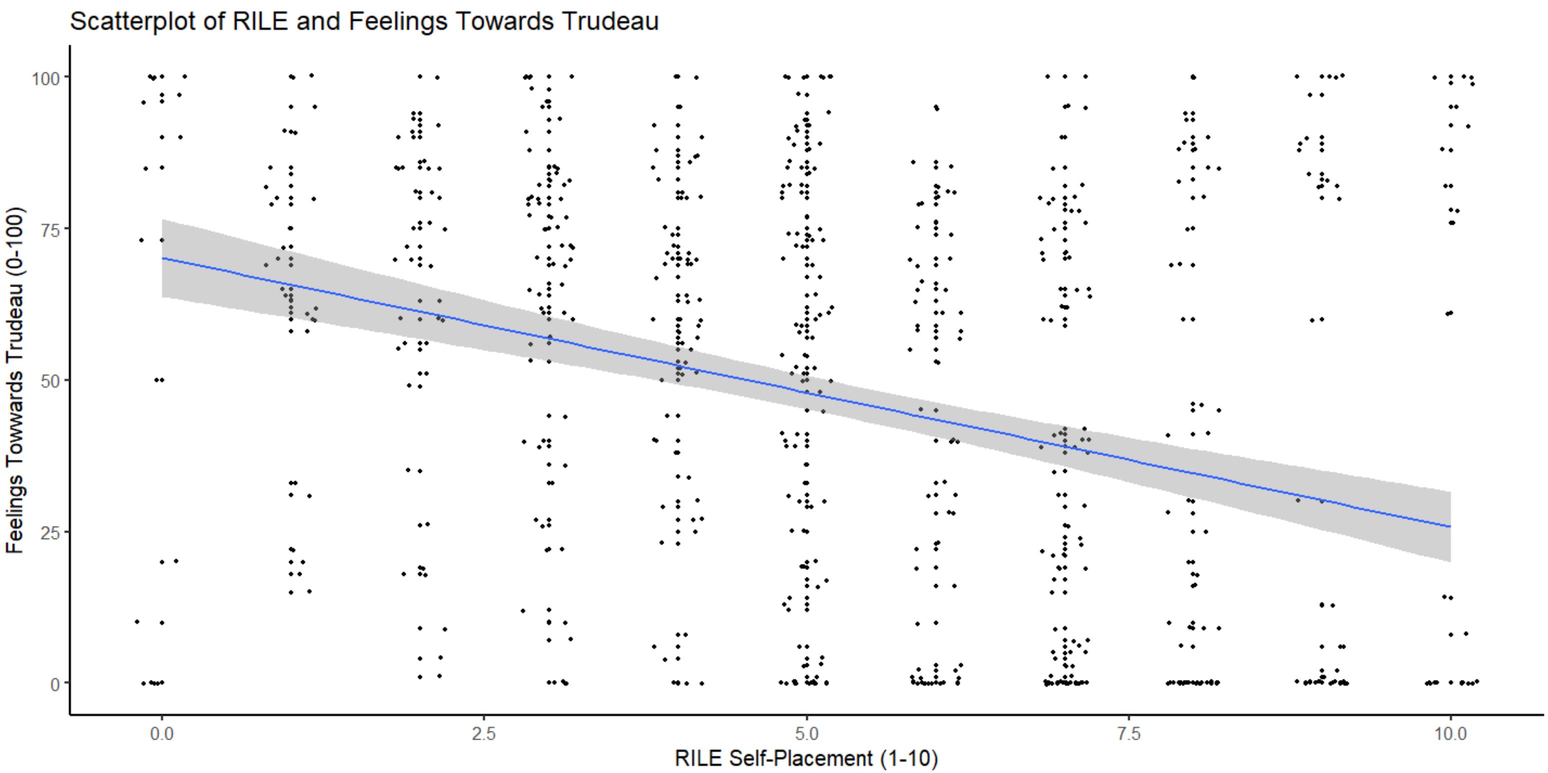
- What do we think now?

Adding a regression line to visualize the relationship between these variables

- We can add a regression line using `geom_smooth()`, to show the average effect of Left-Right ideological placement on feelings towards PM Trudeau before the 2021 federal election.

```
ggplot(ces_2, aes(leftright, feel_trudeau)) +
  geom_point() +
  geom_jitter(width = 0.2, height = 0.2) +
  labs(title =  "Scatterplot of RILE and Feelings Towards Trudeau",
       x = "RILE Self-Placement (1-10)",
       y = "Feelings Towwards Trudeau (0-100)") +
  # using geom_smooth to add a regression line with a confidence interval
  geom_smooth(method = "lm", se = TRUE) +
  theme_classic(base_size = 20)
```

Adding a regression line to visualize the relationship between these variables



- This makes it clear that there seems to be a negative relationship between these two variables. As people living outside of Quebec move further along the Left-Right ideological scale they have more negative feelings towards PM Trudeau.

Running a simple linear regression

- We can use the `lm()` function to run a simple linear regression model to assess the relationship between these two variables and get a better sense of the average effect of Left-Right ideology on feelings towards PM Trudeau outside of Quebec.

```
# saving our model as an object to then print using summary()
model_1 <- lm(feel_trudeau ~ leftright, ces_2)
# printing the results of our model using summary()
summary(model_1)
```

Running a simple linear regression

```
##
## Call:
## lm(formula = feel_trudeau ~ leftright, data = ces_2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -70.241 -31.253   1.774  26.819  74.236
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  70.2409     3.2802  21.414  < 2e-16 ***
## leftright    -4.4476     0.5606  -7.934 1.09e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 33.81 on 582 degrees of freedom
## Multiple R-squared:  0.09761,    Adjusted R-squared:  0.09606
## F-statistic: 62.95 on 1 and 582 DF,  p-value: 1.091e-14
```

- For every one-unit increase on our 1-10 scale of Left-Right ideology people become on average 4.44 points more negative about PM Justin Trudeau. This relationship is statistically significant as the p-value is less than 0.05.

Thank you for listening!

Questions or comments?

nvande@uwo.ca