Introduction to R workshop

James Uanhoro

May 16-17, 2017

Installing *R* & RStudio

R is an open source (so free) programming language & environment for statistical computing and graphics.

Its development is dependent on a core team and volunteer contributors.

Download links for R: Windows; Mac OS X

RStudio is a design environment for *R* which makes *R* easier to use. Its point and click features make programming in *R* less of a pain :). This document was created using *R* & RStudio.

Download links for RStudio: Windows; Mac OS X

Using *R* as a calculator (and saving variables)

R can be used for arithmetic calculations.

```
2 + 2
```

```
## [1] 4
```

```
2 * 2
```

```
## [1] 4
```

```
5 * 2
```

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

```
5000 / 5003
```

```
## [1] 0.9994004
```

```
# In R, anything after a "hashtag", is not executed by R, it's returned verbatim!
# I'm going to comment the following code to know why I wrote it.
# Calculating the number of seconds in a year, not sure why.
60 * 60 * 24 * 365.25
```

```
## [1] 31557600
```

```
# If someone lived for 84 years, how many seconds did they live?
60 * 60 * 24 * 365.25 * 84  # It's cumbersome typing to calculate the seconds in a year again.
```

```
## [1] 2650838400
```

```
# We can save values
num.secs.in.year <- 60 * 60 * 24 * 365.25
# Math folks would call this new value a "scalar" variable - useful to know online-forum-nerd-term (ofnt)
num.secs.in.year  # Returns the value saved into the variable
```

```
## [1] 31557600
```

```
# We can answer the same question again this time with:
num.secs.in.year * 84
```

```
## [1] 2650838400
```

```
# Variables can also be text
my.full.name <- "James Ohisei Uanhoro"
```

My name and numbers

- My name is wrapped in quotes. This is important because text in *R* may denote certain special words (keywords). For *R* to recognize it as plain text, we need to wrap it in quotes. If we don't, *R* will throw a fit.
- Numbers are numbers are numbers, no need for wrapping. Variables like `num.secs.in.year` are keywords, *R* will recognize it without wrapping in quotes.

A few brief notes about `<-`

- In *R*, when we save/store anything, we prefer to use the `<-` or `back arrow` sign for *variable assignment*.
- An artifact of history
- It's easier to use `=`, it would have the same effect almost every single time
- Use `<-` so your code looks familiar to other *R* programmers
- If you are sharing your code, it's a good thing to follow the styles used by others

A few notes about `.` and naming conventions

- In *R*, we prefer to use nouns to save variable names: `num.secs.in.year`, `my.lucky.number`, `my.best.food`, ...
- So we can read our code, we need to separate the words
- We prefer to use `.` to separate words in variable names

A note about styles

- As in real life, they are just preferences
- A styling convention is called a **"style guide"**
- Most commonly cited *R* style guide is Google's R Style Guide

Lists, tables, data frames and data i/o

R scripts (detour)

These are simple text files that you can type and save your code into. This way, you can simply run the script from scratch and recreate your previous work.

In RStudio:

- `File -> New File -> R Script`
- Save it into a directory of your choosing with `Ctrl + s`

Lists

In *R*, we can create lists (`ofnt: vector`), which are just what they are in plain English.

```
ages.of.nieces <- c(2, 2, 4, 6, 8)  # The list of ages (in years) of my nieces
midwest.states <- c("Michigan", "Ohio", "Illinois", "Minnessota")  # A few off the top of my head
midwest.states
```

```
## [1] "Michigan"   "Ohio"       "Illinois"   "Minnessota"
```

What is c?

- `c()` is our first *function* in *R*.
- A *function* is essentially a keyword that *R* interprets as a defined set of instructions to follow
- We can "call" a function in *R* by typing the function name, followed by parantheses
- The `c()` or combine function will take whatever you place inside the parentheses, separated by commas, and create a new list.
- We can also pass in variables, or even other lists into lists

```
obama.cab.officials <- c("Clinton", "Holder", "Gates", "Duncan", "Emanuel")
trump.cab.officials <- c("Tillerson", "Sessions", "Mattis", "DeVos", "Priebus")
(all.cab.officials <- c(obama.cab.officials, trump.cab.officials))
```

```
##  [1] "Clinton"   "Holder"    "Gates"     "Duncan"    "Emanuel"
##  [6] "Tillerson" "Sessions"  "Mattis"    "DeVos"     "Priebus"
```

We can access particular items (`ofnt: elements`) inside lists by typing the name of the list and brackets with the position of the item we want inside the brackets.

```
# Report the number one state in the Midwest
midwest.states[1]
```

```
## [1] "Michigan"
```

```
# Save it in some new variable
midwest.number.one <- midwest.states[1]
midwest.states[2]  # Should be Ohio
```

```
## [1] "Ohio"
```

```
midwest.states[-1]  # Should return everything except the first item, we don't really care for Michigan
```

```
## [1] "Ohio"       "Illinois"   "Minnessota"
```

```
midwest.states[2:4]  # Should also return everything except the first item, we don't really care for Michigan
```

```
## [1] "Ohio"       "Illinois"   "Minnessota"
```

Let's create a few more lists

```
obama.cab.ages <- c(61, 58, 65, 44, 49)
trump.cab.ages <- c(65, 70, 66, 59, 45)
obama.cab.gender <- c("F", "M", "M", "M", "M")
trump.cab.gender <- c("M", "M", "M", "F", "M")
```

Some common functions on lists

```
length(obama.cab.officials)  # How many Obama cabinet officials?
```

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

```
min(trump.cab.ages)  # The age of the young cabinet level Trump official?
```

```
## [1] 45
```

```
max(obama.cab.ages)  # The age of the oldest cabinet level Obama official?
```

```
## [1] 65
```

```r
mean(trump.cab.ages)  # The average age of the five Trump officials?
```

```
## [1] 61
```

We've just seen four common functions in *R*. `length`: number of items in a list; `min`: minimum item in a list; `max`: maximum item in a list; `mean`: average item in a list

---

Tables

We can also create tables (`ofnt: matrix`), which are a little more like the data we work with.

```r
(obama.table <- matrix(c(obama.cab.officials, obama.cab.ages, obama.cab.gender), nrow = 5))
```

```
##      [,1]      [,2] [,3]
## [1,] "Clinton" "61" "F"
## [2,] "Holder"  "58" "M"
## [3,] "Gates"   "65" "M"
## [4,] "Duncan"  "44" "M"
## [5,] "Emanuel" "49" "M"
```

```r
(trump.table <- matrix(c(trump.cab.officials, trump.cab.ages, trump.cab.gender), ncol = 3))
```

```
##      [,1]        [,2] [,3]
## [1,] "Tillerson" "65" "M"
## [2,] "Sessions"  "70" "M"
## [3,] "Mattis"    "66" "M"
## [4,] "DeVos"     "59" "F"
## [5,] "Priebus"   "45" "M"
```

Accessing table rows and columns

```r
trump.table[, 3]  # Pulling out column 3 data from Trump
```

```
## [1] "M" "M" "M" "F" "M"
```

```r
obama.table[2, ]  # Pulling out row 2 data from Obama
```

```
## [1] "Holder" "58"     "M"
```

Data frames

Data frames are what we most often work with in *R*. Think of them as tables in Excel.

```
# We can convert the Trump table into a dataframe
(trump.df <- as.data.frame(trump.table))
```

```
##           V1 V2 V3
## 1 Tillerson 65  M
## 2  Sessions 70  M
## 3    Mattis 66  M
## 4     DeVos 59  F
## 5   Priebus 45  M
```

```
# But this would be nicer
(obama.df <- data.frame(name = obama.cab.officials, age = obama.cab.ages, gender = obama.cab.gender))
```

```
##      name age gender
## 1 Clinton  61      F
## 2  Holder  58      M
## 3   Gates  65      M
## 4  Duncan  44      M
## 5 Emanuel  49      M
```

Let's update Trump's

```
names(trump.df) <- c("name", "age", "gender")
trump.df  # Now with column names
```

```
##        name age gender
## 1 Tillerson  65      M
## 2  Sessions  70      M
## 3    Mattis  66      M
## 4     DeVos  59      F
## 5   Priebus  45      M
```

Columns names are nice in *R* because they make data easy to access.

```
# What are the names of Trump officials?
trump.df$name
```

```
## [1] Tillerson Sessions  Mattis    DeVos     Priebus
## Levels: DeVos Mattis Priebus Sessions Tillerson
```

```
# What are their ages?
trump.df$age
```

```
## [1] 65 70 66 59 45
## Levels: 45 59 65 66 70
```

Important: You can access a column in a data frame by typing the name, a dollar sign, then the column name.

Some common functions on data frames

Data frames always have a structure. Some columns are numbers, others are text, . . . Whenever you create or open a dataframe, the `str()` function should be your first go-to. Here, we know what the columns mean as we created them, but if you are importing data into *R*, it can help you understand what the columns/variables in your data mean.

```
str(trump.df)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ name  : Factor w/ 5 levels "DeVos","Mattis",..: 5 4 2 1 3
##  $ age   : Factor w/ 5 levels "45","59","65",..: 3 5 4 2 1
##  $ gender: Factor w/ 2 levels "F","M": 2 2 2 1 2
```

Important: Open a dataset in* R*, check its structure using* str()*!

```
head(trump.df)  # See the first six rows of data
```

```
##        name age gender
## 1 Tillerson  65      M
## 2  Sessions  70      M
## 3    Mattis  66      M
## 4     DeVos  59      F
## 5   Priebus  45      M
```

You can also try the `View()` function if the dataframe is not so large.

Data input/output

The easiest way to import data into *R* is from text files. The preferred type is CSV (comma-separated values).

The function we use most is `read.csv()`. And to make things simple, it is best to have the file in the same folder as your R Script.

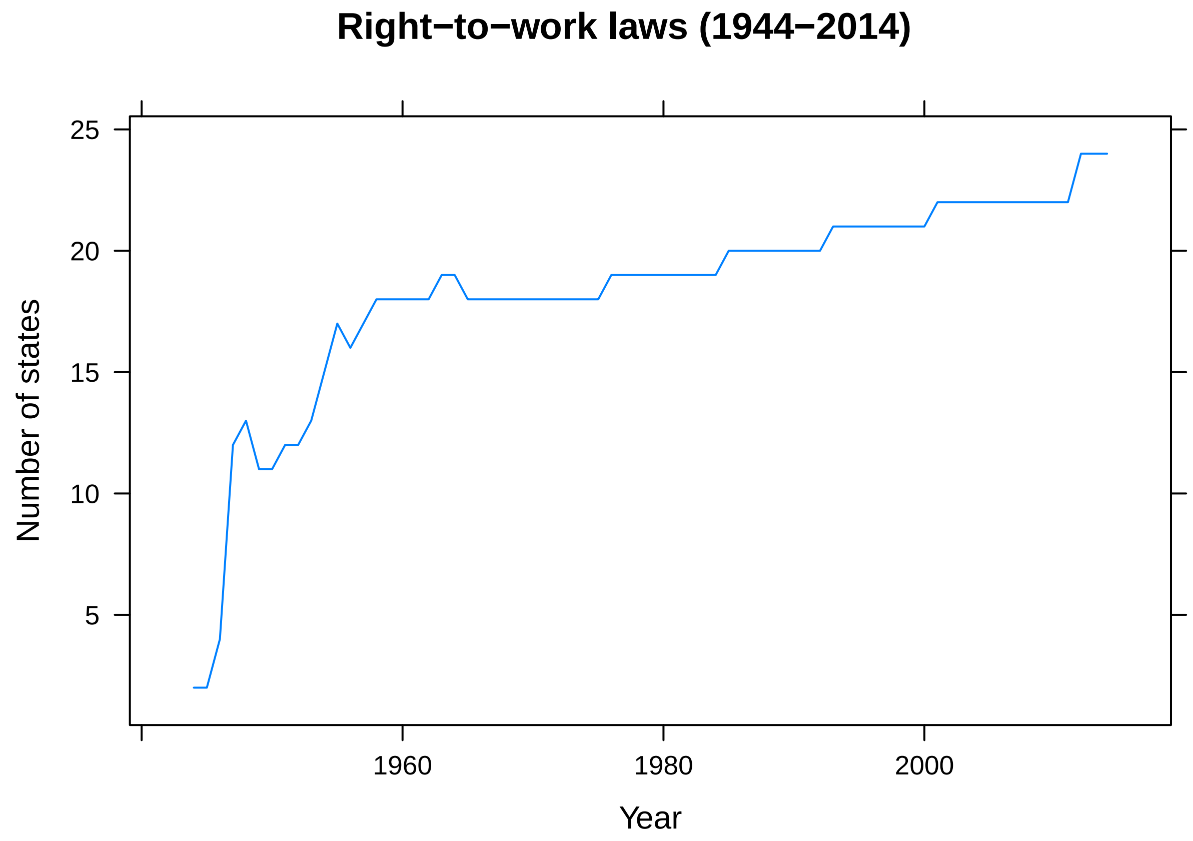
The easiest way to load in a file is to use syntax that takes the form:

```
rtw.data <- read.csv(file.choose())
```

Then navigate and select the CSV file you want to load. Use `csp_policies_1944_2014_rtw.csv`.

This is my preferred way of doing the same thing:

```
rtw.data <- read.csv("./csp_policies_1944_2014_rtw.csv")
```



This saves the data in the file as a data frame named `rtw.data`. We can work with it just as we worked with `trump.df` and `obama.df`.

As recommended earlier, the first thing to do is:

```r
str(rtw.data)
```

```
## 'data.frame':    3621 obs. of  13 variables:
##  $ year          : int  1944 1944 1944 1944 1944 1944 1944 1944 1944 1944 ...
##  $ st            : Factor w/ 51 levels "AK","AL","AR",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ stateno       : num  2 1 4 3 5 6 7 8.5 8 9 ...
##  $ state         : Factor w/ 51 levels "Alabama","Alaska",..: 2 1 4 3 5 6 7 9 8 10 ...
##  $ state_fips    : int  2 1 5 4 6 8 9 11 10 12 ...
##  $ state_icpsr   : int  81 41 42 61 71 62 1 55 11 43 ...
##  $ variable      : Factor w/ 1 level "labor_right_to_work": 1 1 1 1 1 1 1 1 1 1 ...
##  $ value         : int  0 0 1 0 0 0 0 0 0 1 ...
##  $ var_desc      : Factor w/ 1 level "Right to Work law": 1 1 1 1 1 1 1 1 1 1 ...
##  $ dates         : Factor w/ 1 level "1944-2014": 1 1 1 1 1 1 1 1 1 1 ...
##  $ var_long_desc: Factor w/ 1 level "Is the state a right-to-work state?": 1 1 1 1 1 1 1 1 1 1 ...
##  $ sources_notes: Factor w/ 1 level "Caughey, Devin, and Christopher Warshaw. 2015. \"The Dynamics of State Policy Liberalism, 1936-201
##  $ topic         : Factor w/ 1 level "I. POLICY OUTPUTS": 1 1 1 1 1 1 1 1 1 1 ...
```

This dataset has 3621 observations or rows, and 13 variables or columns.

- `st` & `state`: There are 51 (levels of) sts and states
- `variable`, `var_desc`, `dates`, `var_long_desc`, `source_notes`, `topic`: These variables have only one level in them, the same value repeats over and over. For `topic`, every row contains the text: “I. POLICY OUTPUTS”
- `value` & `year`: integers or numbers

We can delete variables that do not contain useful information

```r
rtw.data <- subset(rtw.data, select = c("year", "st", "state", "value"))
str(rtw.data)
```

```
## 'data.frame':    3621 obs. of  4 variables:
##  $ year : int  1944 1944 1944 1944 1944 1944 1944 1944 1944 1944 ...
##  $ st   : Factor w/ 51 levels "AK","AL","AR",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ state: Factor w/ 51 levels "Alabama","Alaska",..: 2 1 4 3 5 6 7 9 8 10 ...
##  $ value: int  0 0 1 0 0 0 0 0 0 1 ...
```

We can also choose to retain only rows that meet certain criteria. How about data only for 2009?

```r
rtw.data.2009 <- subset(rtw.data, subset = year == 2009)
str(rtw.data.2009)  # Now we have 51 rows from the 50 states + DC
```

```
## 'data.frame':    51 obs. of  4 variables:
```

```
##  $ year : int  2009 2009 2009 2009 2009 2009 2009 2009 2009 2009 ...
##  $ st   : Factor w/ 51 levels "AK","AL","AR",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ state: Factor w/ 51 levels "Alabama","Alaska",..: 2 1 4 3 5 6 7 9 8 10 ...
##  $ value: int  0 1 1 1 0 0 0 0 0 1 ...
```

A few brief notes about `==`

- This is a test of equality.
- *R* contains several of these tests: `==`, `!=`, `>`, `<`, `>=`, `<=`, ...

Alternatively, we could select data from 2000 onwards

```r
rtw.data.2000.onward <- subset(rtw.data, subset = year >= 2000)
min(rtw.data.2000.onward$year)  # Mininum year should be 2000
```

```
## [1] 2000
```

A few brief notes about `subset`

- The first thing we pass in is our data frame.
- We have a few options we can pass in. To see them, type `?subset`. We can do the same for any *R* function if we forget its options (`ofnt: arguments`)
- If we want to select columns/variables, we use the `select` option and pass a list of wanted variable names
- If we want to select specific rows, we use the `subset` option and tell it the test of equality

Another useful function is `summary`. It can summarize a data frame for us.

```r
summary(rtw.data.2000.onward)
```

```
##       year            st              state          value       
##  Min.   :2000   AK     : 15   Alabama   : 15   Min.   :0.0000  
##  1st Qu.:2003   AL     : 15   Alaska    : 15   1st Qu.:0.0000  
##  Median :2007   AR     : 15   Arizona   : 15   Median :0.0000  
##  Mean   :2007   AZ     : 15   Arkansas  : 15   Mean   :0.4379  
##  3rd Qu.:2011   CA     : 15   California: 15   3rd Qu.:1.0000  
##  Max.   :2014   CO     : 15   Colorado  : 15   Max.   :1.0000  
##                 (Other):675   (Other)   :675
```

We can export this new data frame we have created into a CSV file for use in another program.

```r
write.csv(rtw.data.2000.onward, file = "right_to_work_2000_onward.csv", row.names = FALSE)
```

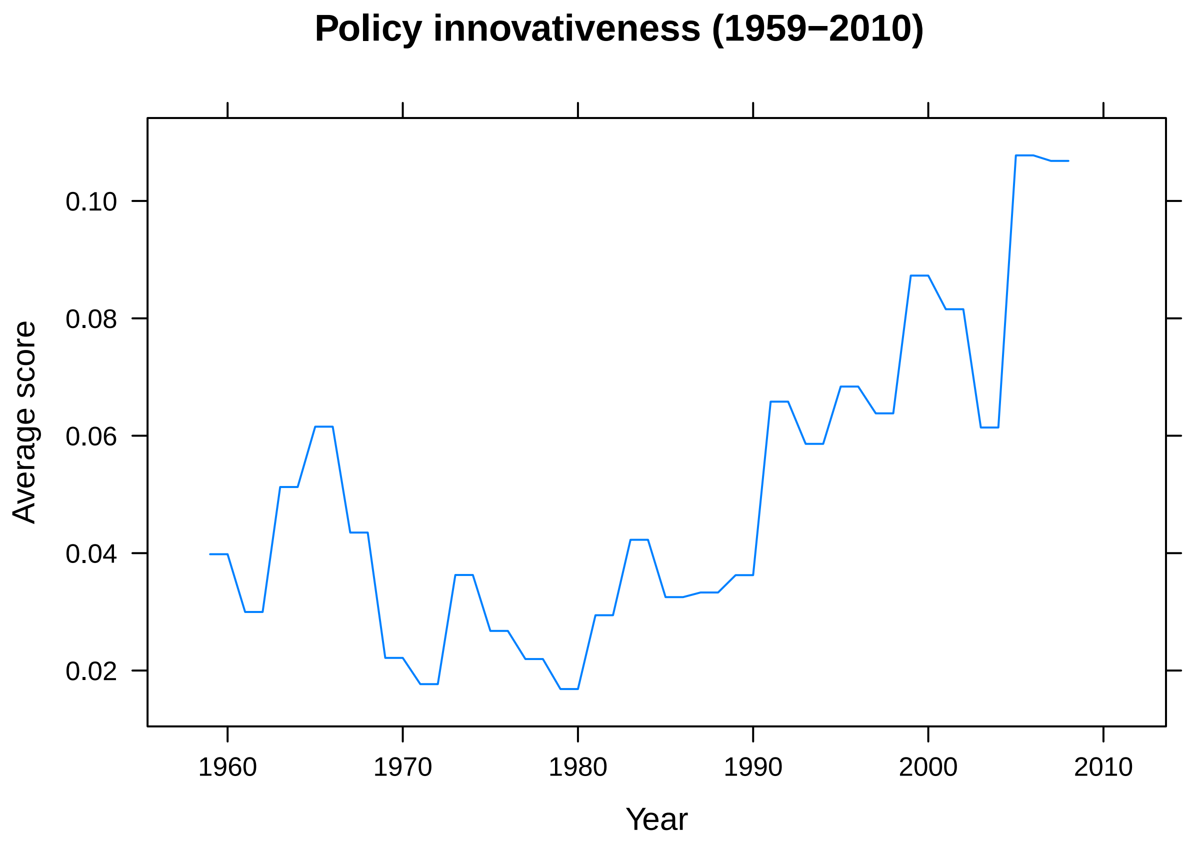
Your data frame will be saved into a file the same folder

Pop question:

How many states had right-to-work signed into law in 2009? Use the `rtw.data.2009` data frame to answer the question. *Hint*: Use the function `sum`

Introduction to functions - basic descriptive statistics

Import the `csp_policies_1959_2010_policies.csv` dataset, and save it as a variable named `policies.data`



What is the first thing to do?

```
## 'data.frame':    2600 obs. of  9 variables:
##  $ year                          : int  1959 1959 1959 1959 1959 1959 1959 1959 1959 1959 ...
##  $ st                            : Factor w/ 50 levels "AK","AL","AR",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ stateno                       : int  2 1 4 3 5 6 7 8 9 10 ...
##  $ state                         : Factor w/ 50 levels "Alabama","Alaska",..: 2 1 4 3 5 6 7 8 9 10 ...
##  $ value.pollib_median           : num  0.193 -1.183 -0.874 -0.45 1.193 ...
##  $ value.pollib_lower            : num  -0.127 -1.688 -1.336 -0.897 0.878 ...
##  $ value.pollib_upper               : num  0.5237 -0.7021 -0.4491 -0.0275 1.5234 ...
##  $ value.pollib_sd                  : num  0.199 0.295 0.266 0.26 0.204 ...
##  $ value.innovatescore_boehmkeskinner: num  0.0957 0 0.078 0.02 0.2105 ...
```

```
##   year st stateno      state value.pollib_median value.pollib_lower
## 1 1959 AK       2     Alaska            0.193347          -0.127334
## 2 1959 AL       1    Alabama           -1.182550          -1.687840
## 3 1959 AR       4   Arkansas           -0.874343          -1.336310
## 4 1959 AZ       3    Arizona           -0.450413          -0.896608
## 5 1959 CA       5 California            1.193480           0.878236
## 6 1959 CO       6   Colorado            1.067440           0.770442
##   value.pollib_upper value.pollib_sd value.innovatescore_boehmkeskinner
## 1           0.523733        0.199320                           0.095726
## 2          -0.702068        0.294711                           0.000000
## 3          -0.449064        0.266077                           0.078046
## 4          -0.027465        0.259855                           0.020029
## 5           1.523410        0.204003                           0.210526
## 6           1.409950        0.200795                           0.163658
```

Some useful descriptive statistics

First, let's get Ohio specific data

```r
policies.data.ohio <- subset(policies.data, subset = state == "Ohio")
str(policies.data.ohio)
```

```
## 'data.frame':    52 obs. of  9 variables:
##  $ year                             : int  1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 ...
##  $ st                               : Factor w/ 50 levels "AK","AL","AR",..: 35 35 35 35 35 35 35 35 35 35 ...
##  $ stateno                          : int  35 35 35 35 35 35 35 35 35 35 ...
##  $ state                            : Factor w/ 50 levels "Alabama","Alaska",..: 35 35 35 35 35 35 35 35 35 35 ...
##  $ value.pollib_median              : num  0.326 0.313 0.528 0.461 0.391 ...
##  $ value.pollib_lower               : num  0.0166 -0.0275 0.2292 0.1467 0.0775 ...
##  $ value.pollib_upper               : num  0.639 0.669 0.857 0.771 0.682 ...
##  $ value.pollib_sd                  : num  0.19 0.214 0.194 0.185 0.192 ...
##  $ value.innovatescore_boehmkeskinner: num  0.1095 0.1095 0.0327 0.0327 0.0637 ...
```

We can compare average political liberalism across all states across time against Ohio since 1959

```r
mean(policies.data$value.pollib_median)
```

```
## [1] 0.03705148
```

```r
mean(policies.data.ohio$value.pollib_median)
```

```
## [1] 0.1045266
```

On average, Ohio has been higher than average.

Joining data frames

Let's get policy data, and right-to-work data from 2008

```r
policies.data.2008 <- subset(policies.data, subset = year == 2008)
str(policies.data.2008)
```

```
## 'data.frame':    50 obs. of  9 variables:
##  $ year                            : int  2008 2008 2008 2008 2008 2008 2008 2008 2008 2008 ...
##  $ st                              : Factor w/ 50 levels "AK","AL","AR",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ stateno                         : int  2 1 4 3 5 6 7 8 9 10 ...
##  $ state                           : Factor w/ 50 levels "Alabama","Alaska",..: 2 1 4 3 5 6 7 8 9 10 ...
##  $ value.pollib_median             : num  0.224 -1.906 -1.693 -0.641 2.433 ...
##  $ value.pollib_lower              : num  -0.233 -2.382 -2.157 -1.034 2.002 ...
##  $ value.pollib_upper              : num  0.621 -1.454 -1.388 -0.208 2.909 ...
##  $ value.pollib_sd                 : num  0.269 0.279 0.238 0.265 0.273 ...
##  $ value.innovatescore_boehmkeskinner: num  0.1096 0.0524 0.0717 0.2212 0 ...
```

```r
rtw.data.2008 <- subset(rtw.data, subset = year == 2008)
str(rtw.data.2008)
```

```
## 'data.frame':    51 obs. of  4 variables:
##  $ year : int  2008 2008 2008 2008 2008 2008 2008 2008 2008 2008 ...
##  $ st   : Factor w/ 51 levels "AK","AL","AR",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ state: Factor w/ 51 levels "Alabama","Alaska",..: 2 1 4 3 5 6 7 9 8 10 ...
##  $ value: int  0 1 1 1 0 0 0 0 0 1 ...
```

Policy data from 2008 has 50 rows. Right-to-work data from 2008 has 51 rows. What's the difference? We can `View()` both data frames to see why the discrepancy exists.

```r
View(policies.data.2008)
View(rtw.data.2008)
```

The difference is DC. The long way is create a new right-to-work data frame without DC.

```
rtw.data.2008.no.dc <- subset(rtw.data.2008, subset = st != "DC")  # Subset of states that are not called DC
str(rtw.data.2008.no.dc)  # We now have 50 rows
```

```
## 'data.frame':    50 obs. of  4 variables:
##  $ year : int  2008 2008 2008 2008 2008 2008 2008 2008 2008 2008 ...
##  $ st   : Factor w/ 51 levels "AK","AL","AR",..: 1 2 3 4 5 6 7 9 10 11 ...
##  $ state: Factor w/ 51 levels "Alabama","Alaska",..: 2 1 4 3 5 6 7 8 10 11 ...
##  $ value: int  0 1 1 1 0 0 0 0 1 1 ...
```

We can now join right-to-work data with policy liberalism data. We can save the policy liberalism score into a new column in `rtw.data.2008.no.dc`

```
rtw.data.2008.no.dc$lib.score <- policies.data.2008$value.pollib_median
str(rtw.data.2008.no.dc)
```

```
## 'data.frame':    50 obs. of  5 variables:
##  $ year     : int  2008 2008 2008 2008 2008 2008 2008 2008 2008 2008 ...
##  $ st       : Factor w/ 51 levels "AK","AL","AR",..: 1 2 3 4 5 6 7 9 10 11 ...
##  $ state    : Factor w/ 51 levels "Alabama","Alaska",..: 2 1 4 3 5 6 7 8 10 11 ...
##  $ value    : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ lib.score: num  0.224 -1.906 -1.693 -0.641 2.433 ...
```

The smarter thing to do is to merge both data frames.

```
str(policies.data.2008)
```

```
## 'data.frame':    50 obs. of  9 variables:
##  $ year                            : int  2008 2008 2008 2008 2008 2008 2008 2008 2008 2008 ...
##  $ st                              : Factor w/ 50 levels "AK","AL","AR",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ stateno                         : int  2 1 4 3 5 6 7 8 9 10 ...
##  $ state                           : Factor w/ 50 levels "Alabama","Alaska",..: 2 1 4 3 5 6 7 8 9 10 ...
##  $ value.pollib_median             : num  0.224 -1.906 -1.693 -0.641 2.433 ...
##  $ value.pollib_lower              : num  -0.233 -2.382 -2.157 -1.034 2.002 ...
##  $ value.pollib_upper              : num  0.621 -1.454 -1.388 -0.208 2.909 ...
##  $ value.pollib_sd                 : num  0.269 0.279 0.238 0.265 0.273 ...
##  $ value.innovatescore_boehmkeskinner: num  0.1096 0.0524 0.0717 0.2212 0 ...
```

```
str(rtw.data.2008)
```

```
## 'data.frame':    51 obs. of  4 variables:
##  $ year : int  2008 2008 2008 2008 2008 2008 2008 2008 2008 2008 ...
##  $ st   : Factor w/ 51 levels "AK","AL","AR",..: 1 2 3 4 5 6 7 8 9 10 ...
## $ state: Factor w/ 51 levels "Alabama","Alaska",..: 2 1 4 3 5 6 7 9 8 10 ...
## $ value: int  0 1 1 1 0 0 0 0 0 1 ...
```

```
new.data.2008 <- merge(policies.data.2008, rtw.data.2008, by = c("year", "st", "state"))
str(new.data.2008)  # It dropped DC automatically for us. Be careful, there are options which can be used to control for this.
```

```
## 'data.frame':    50 obs. of  10 variables:
##  $ year                           : int  2008 2008 2008 2008 2008 2008 2008 2008 2008 2008 ...
##  $ st                             : Factor w/ 50 levels "AK","AL","AR",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ state                          : Factor w/ 50 levels "Alabama","Alaska",..: 2 1 4 3 5 6 7 8 9 10 ...
##  $ stateno                        : int  2 1 4 3 5 6 7 8 9 10 ...
##  $ value.pollib_median            : num  0.224 -1.906 -1.693 -0.641 2.433 ...
##  $ value.pollib_lower             : num  -0.233 -2.382 -2.157 -1.034 2.002 ...
##  $ value.pollib_upper             : num  0.621 -1.454 -1.388 -0.208 2.909 ...
##  $ value.pollib_sd                : num  0.269 0.279 0.238 0.265 0.273 ...
##  $ value.innovatescore_boehmkeskinner: num  0.1096 0.0524 0.0717 0.2212 0 ...
##  $ value                          : int  0 1 1 1 0 0 0 0 1 1 ...
```

Let's rename the `value` column as `value.rtw`

```
# We can replace the last name, which is the 10th column
names(new.data.2008)  # Produces a list of variable names. We can edit the 10th item in the list as seen below
```

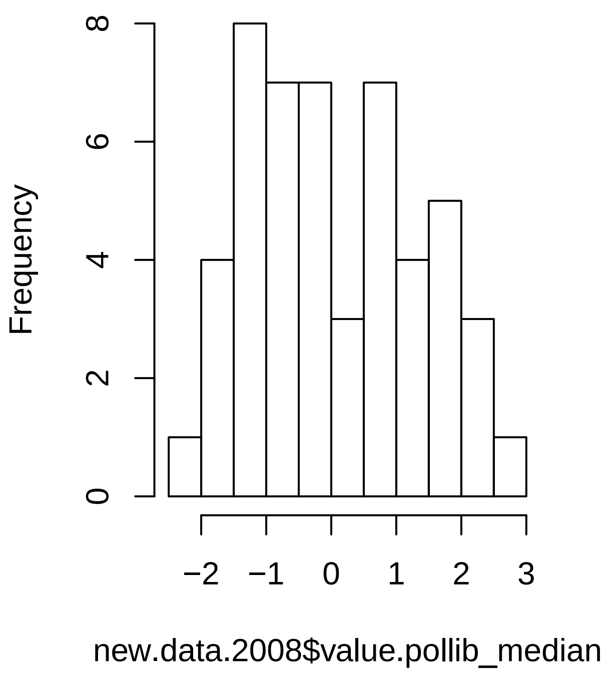
```
##  [1] "year"
##  [2] "st"
##  [3] "state"
##  [4] "stateno"
##  [5] "value.pollib_median"
##  [6] "value.pollib_lower"
##  [7] "value.pollib_upper"
##  [8] "value.pollib_sd"
##  [9] "value.innovatescore_boehmkeskinner"
## [10] "value"
```

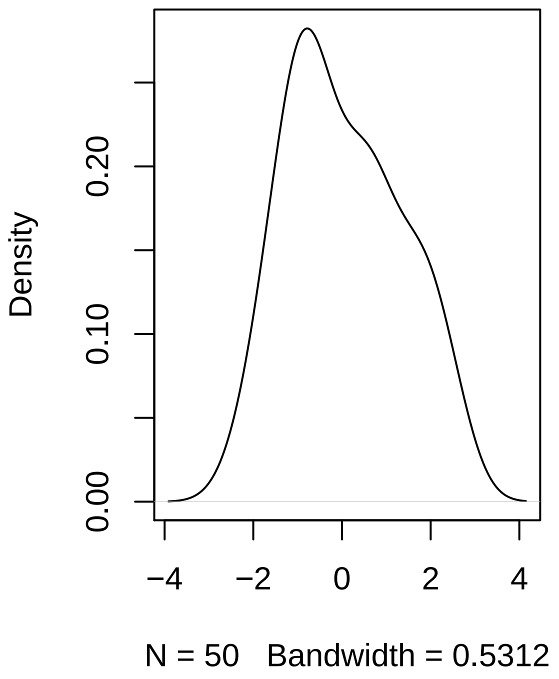
```
names(new.data.2008)[10] <- "value.rtw"
str(new.data.2008)  # Done!
```

```
## 'data.frame':    50 obs. of  10 variables:
##  $ year                           : int  2008 2008 2008 2008 2008 2008 2008 2008 2008 2008 ...
##  $ st                             : Factor w/ 50 levels "AK","AL","AR",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ state                          : Factor w/ 50 levels "Alabama","Alaska",..: 2 1 4 3 5 6 7 8 9 10 ...
##  $ stateno                        : int  2 1 4 3 5 6 7 8 9 10 ...
##  $ value.pollib_median               : num  0.224 -1.906 -1.693 -0.641 2.433 ...
##  $ value.pollib_lower                : num  -0.233 -2.382 -2.157 -1.034 2.002 ...
##  $ value.pollib_upper                : num  0.621 -1.454 -1.388 -0.208 2.909 ...
##  $ value.pollib_sd                   : num  0.269 0.279 0.238 0.265 0.273 ...
##  $ value.innovatescore_boehmkeskinner: num  0.1096 0.0524 0.0717 0.2212 0 ...
##  $ value.rtw                         : int  0 1 1 1 0 0 0 0 1 1 ...
```

A few useful plots

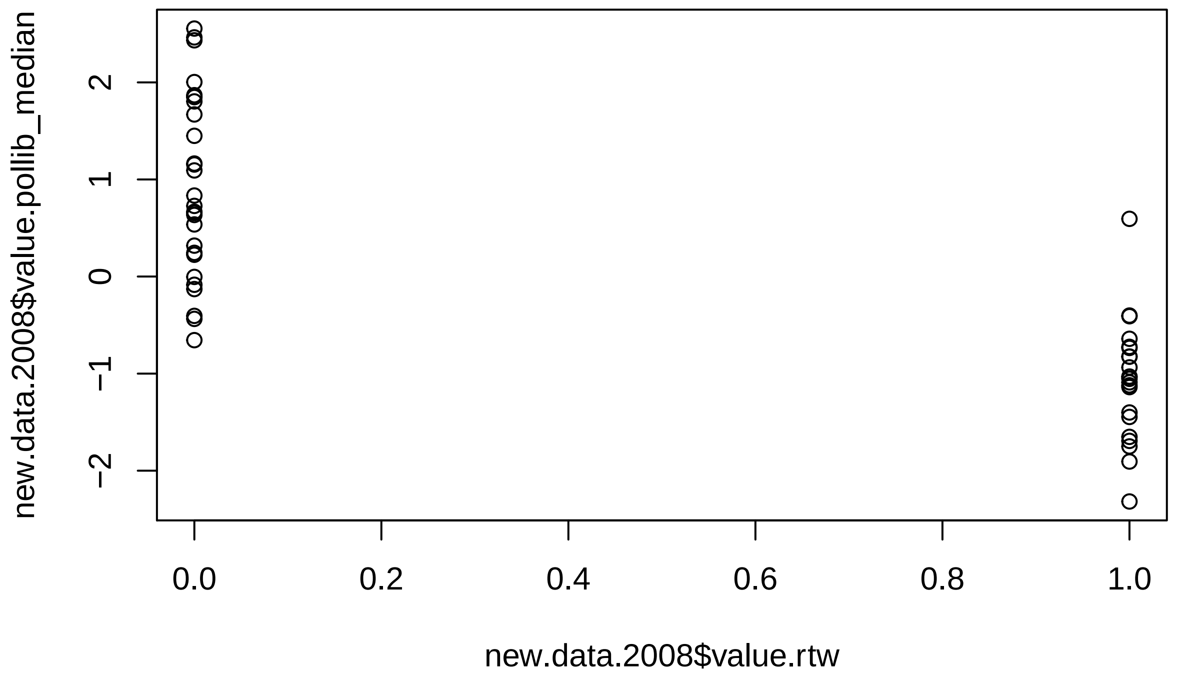
```
hist(new.data.2008$value.pollib_median)  # A histogram, useful for continuous data
plot(density(new.data.2008$value.pollib_median))  # A density, also nice for continuous data
```





We can also have a plot which allows us to see the relationship between right-to-work and liberalism

```
plot(x = new.data.2008$value.rtw, y = new.data.2008$value.pollib_median)
```



A simple descriptive statistic can tell us about the relationship between liberalism score and whether or not a state had right-to-work enacted in 2008. It is called the *correlation*. It ranges from -1 to +1. -1 is a negative relationship, as one goes up, the other goes down. +1 means both variables change in the same direction. A zero correlation means no relation.

What would we expect for the correlation between whether right-to-work is enacted and liberalism score?

```
cor(new.data.2008$value.rtw, new.data.2008$value.pollib_median)
```

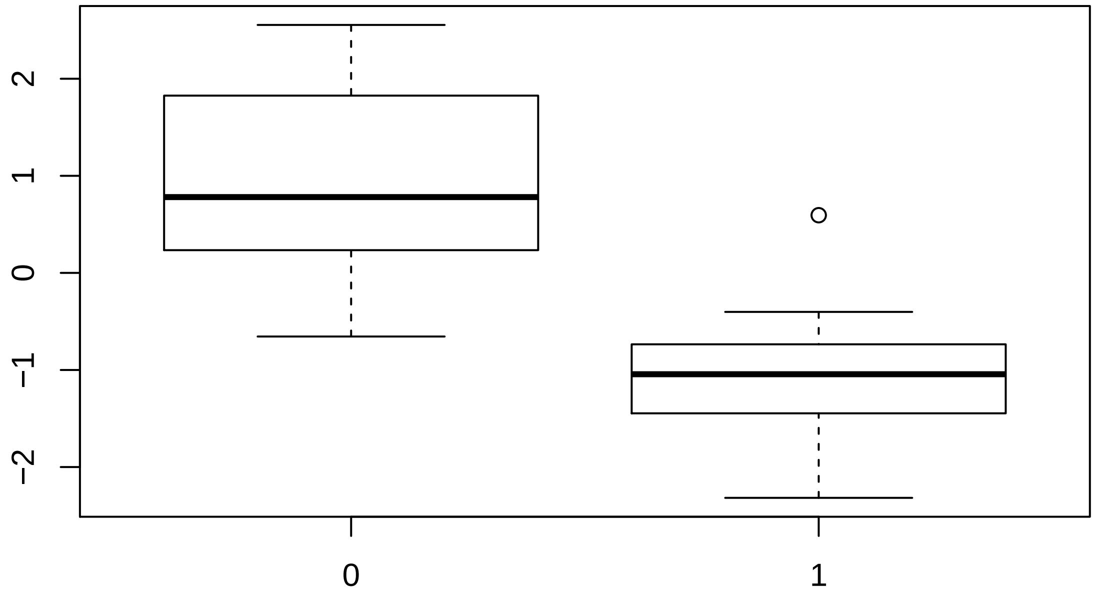
```
## [1] -0.7831318
```

Other useful descriptive statistics include `range`, `sd`, `median`, and `quantile`.

Formula

In *R*, some functions allow you to specify formula on-the-fly. Essentially, if you want to create a boxplot of one variable showing the different distributions for two groups, you can do this using formula. An example is the boxplot.

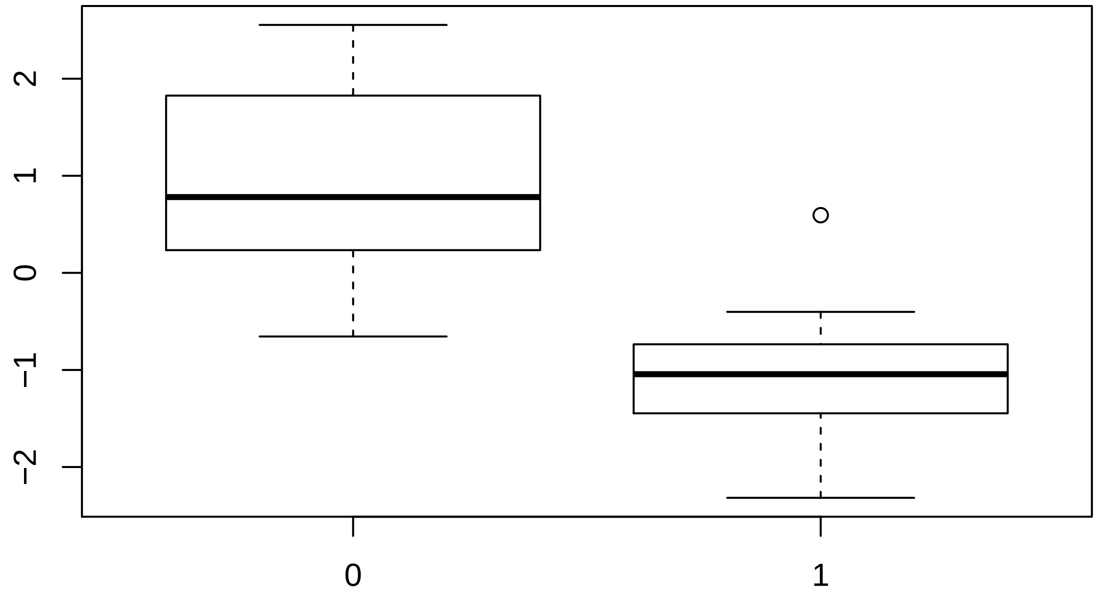
```r
boxplot(value.pollib_median ~ value.rtw, data = new.data.2008)
```



The `~` (tilde) sign allows for the specification of formula in *R*. It can be used in so many ways, but the idea here is to split the first variable by the second variable. Typically, to the left of `~` is an outcome variable and to its right are the explanatory variables.

When using formula in *R*, the safest way to do this is to specify the formula as formula using `as.formula()`. Additionally, when we use grouping/categorical variables in formula, it is best to express them as factors using `as.factor()`. Boxplots done righter:

```
boxplot(
  as.formula(value.pollib_median ~ as.factor(value.rtw)),
  data = new.data.2008
)
```



We'll revisit this in our exercises.

CRAN & working with packages

Technically, *R* would be useless without packages. The package that comes by default with *R* is the `base` package. It contains all the functions, `plot`, `c`, `subset`, `str`, . . . that we have used so far.

However, the *R* base package is quite limited. Assuming we want to import data from Excel (.xls files) SPSS (.sav files) or SAS (.dta files), unless one is an actual software programmer, this is next to impossible.

Volunteers, often uber programmers, create packages that help us do these things that are not available by default in *R* (in the `base` package).

The Comprehensive R Archive Network (CRAN) is an online database that these volunteers upload their packages to, and we can download then use these packages from CRAN within RStudio.

First, you need to find the package you need. If you want to import data from programs like SAS or SPSS, there is a very useful package called `foreign`. To install it, we type:

```
install.packages("foreign")
```

Your RStudio console should be filled with boatloads of gibberish (that's installation and it's not really gibberish). If everything works right, you should see a message like:

* DONE (foreign)

The downloaded source packages are in
some directory

Sometimes, package installation might fail with messages like `non-zero exit status`. It is possible to read through the earlier referenced gibberish to attempt to figure out what has failed.

Once a package is installed, we can make its functions available for use with:

```
library(foreign)
```

Now we have access to functions like `read.spss`, `read.dta`, which allow us to receive and produce datasets for other programs.

There are three ways to learn about a package and the features it has to offer:

- Its documentation: Most detailed option, and after you load a package, you can access the documentation of a function by typing `?function.name`.
- Its vignettes: Some packages have demonstrations with user-friendly explanations of major functions and code to execute them. If you find that you will be making use of a package often, and it has a vignette, then working through it should be worth your time. After going through a package's vignette, its more extensive documentation should be more accessible.
- Online forums: Most of the time, you pull code from an online forum that sorta kinda gets the job done. As a programmer, whether novice or expert, *The Interwebs is **ALWAYS** your friend.*

Information about a package can almost always be found at https://cran.r-project.org/web/packages/package-name/index.html. The complete documentation is the *Reference manual* and the *Vignettes* if avaialble will be listed.

Let's try the easy to use package, `openxlsx`, which allows us to read in data from Excel, and create Excel spreadsheets from *R*. First, we install the package, and load its functions:

```r
install.packages("openxlsx")
library(openxlsx)
```

`openxlsx` page on CRAN, let's use the `Examples` vignette and work with some Basic Examples.

RStudio & directories

RStudio has 4 panels by default: files in upper left; Environment/History in its upper right (UR) panel; Console in its lower left (LL) panel; and assorted panels in its lower right (LR) panel.

Location	Panel	Utility
UR	Environment	This is a list of variables that *R* has in memory. You can delete them all using the broomstick. Just above the list of variables is a point-and-click *Import Dataset* option.
	History	This is a history of all the commands you have issued to *R*. It is searchable should you want to. And you can copy previously executed commands to an *R* script (`To Source`) or to the *Console* (`To Console`).
LL	Console	This is where you can issue commands to *R*. Commands in scripts also get executed here. And it's of course where *R* throws a fit when it believes we've made a mistake.
LR	Files	This shows your current directory/folder and the files it contains. You can also obtain the list of files in the directory you are working from using `dir()`. The command `getwd` (get working directory) returns the folder you are working from. You can change the folder you are working from **not** by navigating away using the Files panel, but by using the `setwd()` command.
	Plots	As you must have noticed, this is where plots appear. You can export plots from here using the point-and-click options. When doing this, note that you can alter the size (dimensions) of the plot prior to exporting it.
	Packages	All the packages you have installed. You can load a package by ticking it. This is all right but note that if you are using a script, the executed point-and-click command is missing from the script. This is when it might be useful to go to *History* and copy the executed code to your script.
	Help	Helper documentation/content gets displayed here.
	Viewer	Never noticed this one until preparing for this tutorial :).

Generally, it is useful to locate your work within *R projects*. An *R project* keeps all of your work together in one space, and if you save, close, then open a *project*, all your work within the project directory returns and you can pick up from where you left off.

- `File` -> `New Project`; Give the project a reasonable name, you would often create it in a new directory.

To return to your work when you've saved and closed it, navigate to the directory in your computer folder navigator, and double-click the *R project* file. It will be named `project-name.Rproj` or simply `project-name`.

R Markdown

Markdown is a text formatting markup language. Essentially, you type text mixed in with keywords and it produces fancy documents like this one. *R* Markdown is markdown specially designed for use within the RStudio environment.

To get started:

```r
install.packages("rmarkdown")
```

Next create an *R* Markdown document in RStudio:

- `File` -> `New File` -> `R Markdown`

Any one of `R Presentation`, `R Notebook` and `R Presentation` are essentially *R* Markdown documents.

RStudio has an *R* Markdown cheat sheet which has all one needs to get started: https://shiny.rstudio.com/images/rm-cheatsheet.png.

To generate tables in Markdown, one of the more tedious tasks, you can use this tool: http://www.tablesgenerator.com/markdown_tables

Functions in *R*

We have been working with functions this whole time - working with functions created by others. If we find ourselves using a particular sequence of commands over and over, we can make it into a function.

Simple examples

```
num <- 5
num * num  # Squaring num
```

```
## [1] 25
```

```
num ** 2  # Squaring num
```

```
## [1] 25
```

```
# A simple function to do this would be:
sqr <- function(x) {
  x ** 2
}
sqr(num)
```

```
## [1] 25
```

```
cub <- function(x) x ** 3  # A single line function
cub(num)
```

```
## [1] 125
```

For now it seems ridiculous and wasteful. Bear with me.

Conditional statements

These are the basic means by which computers make decisions. Humans are smart, computers are dumb :).

ifelse

The simplest structure is `ifelse()`.

```
ifelse(num > 5, 1, 0)  # If num is greater than 5, return 1, else return 0
```

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

```r
(nums <- seq(1, 10))  # Essentially creates a list from a sequence starting at 1 and ending at 10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
ifelse(nums > 5, 1, 0)  # Go through each number in the list, test each one, and return the appropriate value
```

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

Let's dichotomize a continuous variable using the median-split - liberalism score.

```r
# What is the median libscore?
median(new.data.2008$value.pollib_median)
```

```
## [1] -0.1069085
```

```r
# This checks each row and returns "libs" or "cons" depending on the state's liberalism score
ifelse(new.data.2008$value.pollib_median >= -0.1069085, "lib", "cons")
```

```
##  [1] "lib"  "cons" "cons" "cons" "lib"  "cons" "lib"  "lib"  "cons" "cons"
## [11] "lib"  "lib"  "cons" "lib"  "cons" "cons" "cons" "cons" "lib"  "lib"
## [21] "lib"  "lib"  "lib"  "cons" "cons" "lib"  "cons" "cons" "cons" "lib"
## [31] "lib"  "lib"  "cons" "lib"  "lib"  "cons" "lib"  "lib"  "lib"  "cons"
## [41] "cons" "cons" "cons" "cons" "cons" "lib"  "lib"  "lib"  "lib"  "cons"
```

```r
# Alter the code a little
ifelse(new.data.2008$value.pollib_median >= median(new.data.2008$value.pollib_median), "lib", "cons")
```

```
##  [1] "lib"  "cons" "cons" "cons" "lib"  "cons" "lib"  "lib"  "cons" "cons"
## [11] "lib"  "lib"  "cons" "lib"  "cons" "cons" "cons" "cons" "lib"  "lib"
## [21] "lib"  "lib"  "lib"  "cons" "cons" "lib"  "cons" "cons" "cons" "lib"
## [31] "lib"  "lib"  "cons" "lib"  "lib"  "cons" "lib"  "lib"  "lib"  "cons"
## [41] "cons" "cons" "cons" "cons" "cons" "lib"  "lib"  "lib"  "lib"  "cons"
```

```r
# We can even create a new variable in our data
new.data.2008$lib.or.con <- ifelse(
  new.data.2008$value.pollib_median >= median(new.data.2008$value.pollib_median), "lib", "cons"
)
head(new.data.2008)  # It works
```

```
##   year st      state stateno value.pollib_median value.pollib_lower
## 1 2008 AK     Alaska       2            0.224406          -0.233003
## 2 2008 AL    Alabama       1           -1.905830          -2.382270
## 3 2008 AR   Arkansas       4           -1.693160          -2.156850
## 4 2008 AZ    Arizona       3           -0.641311          -1.033700
## 5 2008 CA California       5            2.433460           2.002240
## 6 2008 CO   Colorado       6           -0.129186          -0.510981
##   value.pollib_upper value.pollib_sd value.innovatescore_boehmkeskinner
## 1           0.621103        0.268586                           0.109617
## 2          -1.453790        0.279336                           0.052396
## 3          -1.387520        0.237595                           0.071675
## 4          -0.207705        0.264702                           0.221180
## 5           2.908600        0.272686                           0.000000
## 6           0.295059        0.248050                           0.102410
##   value.rtw lib.or.con
## 1         0        lib
## 2         1       cons
## 3         1       cons
## 4         1       cons
## 5         0        lib
## 6         0       cons
```

```
# Same thing with less typing
new.data.2008$lib.or.con <- with(
  new.data.2008, ifelse(value.pollib_median >= median(value.pollib_median), "lib", "cons")
)
head(new.data.2008)  # It works
```

```
##   year st      state stateno value.pollib_median value.pollib_lower
## 1 2008 AK     Alaska       2            0.224406          -0.233003
## 2 2008 AL    Alabama       1           -1.905830          -2.382270
## 3 2008 AR   Arkansas       4           -1.693160          -2.156850
## 4 2008 AZ    Arizona       3           -0.641311          -1.033700
## 5 2008 CA California       5            2.433460           2.002240
## 6 2008 CO   Colorado       6           -0.129186          -0.510981
##   value.pollib_upper value.pollib_sd value.innovatescore_boehmkeskinner
## 1           0.621103        0.268586                           0.109617
## 2          -1.453790        0.279336                           0.052396
## 3          -1.387520        0.237595                           0.071675
## 4          -0.207705        0.264702                           0.221180
## 5           2.908600        0.272686                           0.000000
## 6           0.295059        0.248050                           0.102410
##   value.rtw lib.or.con
## 1         0        lib
## 2          1       cons
## 3          1       cons
## 4          1       cons
## 5          0        lib
## 6          0       cons
```

When we use the `with()` function, within the parenthesis, we can refer to variables of the data frame without the `$` sign.

If-then-else

Suppose we want to be able to check a state was a right-to-work state in 2008, there are a number of ways we can go about doing this. Let's try with Alaska:

```
(alaska <- subset(new.data.2008, subset = state == "Alaska"))
```

```
##   year st  state stateno value.pollib_median value.pollib_lower
## 1 2008 AK Alaska       2            0.224406          -0.233003
##   value.pollib_upper value.pollib_sd value.innovatescore_boehmkeskinner
## 1           0.621103        0.268586                           0.109617
##   value.rtw lib.or.con
## 1         0        lib
```

```
if (alaska$value.rtw == 0) {
  print("No")
} else {
  print("Yes")
}
```

```
## [1] "No"
```

How about Michigan?

```
mich <- new.data.2008[new.data.2008$state == "Michigan", ]  # Another way of subsetting
if (mich$value.rtw == 0) {
  print("No")
} else {
  print("Yes")
}
```

```
## [1] "No"
```

And Ohio? Hopefully, you get the point, this is tedious. Let's try to write a function for this.

```r
right.to.work <- function(x) {
  state <- new.data.2008[new.data.2008$state == x, ]
  ifelse(state$value.rtw == 0, "No", "Yes")
}
right.to.work("Ohio")
```

```
## [1] "No"
```

```r
right.to.work("Alabama")
```

```
## [1] "Yes"
```

```r
right.to.work("Nevada")
```

```
## [1] "Yes"
```

Let's get a little more creative!

```r
right.to.work <- function(x) {
  filler.text <- "a right-to-work state in 2008."
  state <- new.data.2008[new.data.2008$state == x, ]
  if (state$value.rtw == 0) {
    paste("No,", x, "was not", filler.text)  # Try "?paste"
  } else {
    paste("Yes,", x, "was", filler.text)
  }
}
right.to.work("Ohio")
```

```
## [1] "No, Ohio was not a right-to-work state in 2008."
```

```r
right.to.work("Alabama")
```

```
## [1] "Yes, Alabama was a right-to-work state in 2008."
```

```r
right.to.work("Nevada")
```

```
## [1] "Yes, Nevada was a right-to-work state in 2008."
```

It is my sincere hope that at least one person finds this somewhat interesting :).

Loops

Loops are a feature in most programming languages. They take the basic form:

```
Start a counter at 0, and set counter maximum:
  Do something;
  Increment counter;
  Is counter at maximum?
    If yes: Get me out of here already!
    Else: Return to `Do something`
```

```
for (i in seq(1, 5)) {
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

Is this punishment a thing in the US?: To write `I will never tell a lie again` 1000 times and submit it to your teacher the next day? Let's take on the punishment *for-loop style*.

```
for (i in seq(1, 5)) {
  print("I will never tell a lie again.")
}
```

```
## [1] "I will never tell a lie again."
## [1] "I will never tell a lie again."
## [1] "I will never tell a lie again."
## [1] "I will never tell a lie again."
## [1] "I will never tell a lie again."
```

```
# We can also do this since we do not make use of the i inside the loop
rep("I will never tell a lie again.", 5)
```

```
## [1] "I will never tell a lie again." "I will never tell a lie again."
## [3] "I will never tell a lie again." "I will never tell a lie again."
## [5] "I will never tell a lie again."
```

Building on our `right.to.work` function, let's find the answer for a number of states using a for loop.

```r
for (i in c("Ohio", "Nevada", "Texas", "Arkansas", "Washington")) {
  print(right.to.work(i))
}
```

```
## [1] "No, Ohio was not a right-to-work state in 2008."
## [1] "Yes, Nevada was a right-to-work state in 2008."
## [1] "Yes, Texas was a right-to-work state in 2008."
## [1] "Yes, Arkansas was a right-to-work state in 2008."
## [1] "No, Washington was not a right-to-work state in 2008."
```

Apply

There is a class of `apply` functions: `lapply`, `sapply`, `vapply`, … that are very handy when we seek to apply a function to any type of object in *R*. We will focus on the simplest of these, `sapply`.

```r
sapply(c("Ohio", "Nevada", "Texas", "Arkansas", "Washington"), right.to.work)  # Same as we have above
```

```
##                                                      Ohio
##       "No, Ohio was not a right-to-work state in 2008."
##                                                    Nevada
##        "Yes, Nevada was a right-to-work state in 2008."
##                                                     Texas
##         "Yes, Texas was a right-to-work state in 2008."
##                                                  Arkansas
##      "Yes, Arkansas was a right-to-work state in 2008."
##                                                Washington
## "No, Washington was not a right-to-work state in 2008."
```

```r
# Try the function with the USE.NAMES option, see documentation for how to use it
sapply(sample(new.data.2008$state, 10), right.to.work)  # sapply shines with dataframes, sample any ten states
```

```
##  [1] "No, Vermont was not a right-to-work state in 2008."
##  [2] "Yes, Idaho was a right-to-work state in 2008."
##  [3] "No, New Hampshire was not a right-to-work state in 2008."
##  [4] "Yes, Iowa was a right-to-work state in 2008."
##  [5] "No, Washington was not a right-to-work state in 2008."
##  [6] "Yes, North Carolina was a right-to-work state in 2008."
##  [7] "No, Missouri was not a right-to-work state in 2008."
##  [8] "Yes, South Carolina was a right-to-work state in 2008."
##  [9] "No, New York was not a right-to-work state in 2008."
## [10] "No, Minnesota was not a right-to-work state in 2008."
```

The `apply` functions allow for concision in programming. Concision is nice in programming, but if you ever find that you're being overly clever, comment your code so that when you return to it in the future, probably a lot less illumined, you simply have to learn from your notes to understand it.

Codebooks in *R*

To create codebooks in *R*, we can use the `memisc` package, which has a nice codebook function.

```
install.packages("memisc")
```

Let's create some data

```
library(memisc)
```

```
## Loading required package: MASS

##
## Attaching package: 'memisc'

## The following objects are masked from 'package:stats':
##
##     contr.sum, contr.treatment, contrasts

## The following object is masked from 'package:base':
##
##     as.array
```

```
data.polling <- data.frame(
  vote = sample(c(rep(1, 3), rep(2, 3), 3, 8, 9, 97, 99), size = 300, replace = TRUE),
  region = sample(c(rep(1, 3), rep(2, 2), rep(3, 2), rep(4, 2), 97, 99), size = 300, replace = TRUE),
  income = sample(c(exp(rnorm(290, sd = 0.7)) * 2000, rep(-9, 10)), replace = FALSE)
)
str(data.polling)
```

```
## 'data.frame':    300 obs. of  3 variables:
##  $ vote  : num  2 2 97 3 2 3 1 3 9 1 ...
##  $ region: num  1 1 3 4 1 1 2 2 4 1 ...
##  $ income: num  1399 3018 4299 2073 2155 ...
```

```
data.for.codebook <- as.data.set(data.polling)
str(data.for.codebook)
```

```
## Data set with 300 obs. of 3 variables:
##  $ vote  : Itvl. item  num  2 2 97 3 2 3 1 3 9 1 ...
##  $ region: Itvl. item  num  1 1 3 4 1 1 2 2 4 1 ...
##  $ income: Itvl. item  num  1399 3018 4299 2073 2155 ...
```

And prepare its codebook

```r
data.for.codebook <- within(data.for.codebook, {
  description(vote) <- "Vote intention"
  description(region) <- "Region of residence"
  description(income) <- "Household income"
  wording(vote) <- "If a general election were to take place next Tuesday,
                    you would vote for the candidate of which party?"
  wording(income) <- "All things taken into account, how much do all
                      members of your household earn in total?"
  foreach(x = c(vote, region), {
    measurement(x) <- "nominal"
  })
  measurement(income) <- "ratio"
  labels(vote) <- c(
    "Democratic"          =  1,
    "Republican"          =  2,
    "Third-party"         =  3,
    "Don't know"          =  8,
    "Answer refused"      =  9,
    "Not applicable"      = 97,
    "Not asked in survey" = 99
  )
  labels(region) <- c(
    "Northeast"           =  1,
    "Midwest"             =  2,
    "South"               =  3,
    "West"                =  4,
    "Not applicable"      = 97,
    "Not asked in survey" = 99
  )
  labels(income) <- c(
    "Refused"             = -9
  )
  foreach(x = c(vote, region, income), {
    annotation(x)["Remark"] <- "These are actual data"
  })
  missing.values(vote) <- c(8, 9, 97, 99)
  missing.values(region) <- c(97, 99)
  missing.values(income) <- c(-9)
})
```

We can ask for the `description` of a dataset.

```
description(data.for.codebook)
```

```
##
##  vote   'Vote intention'
##  region 'Region of residence'
##  income 'Household income'
```

This is what the codebook looks like.

```
codebook(data.for.codebook)
```

```
## ===========================================================================
##
##    vote 'Vote intention'
##
##    "If a general election were to take place next Tuesday, you would vote
##    for the candidate of which party?"
##
## ---------------------------------------------------------------------------
##
##    Storage mode: double
##    Measurement: nominal
##    Missing values: 8, 9, 97, 99
##
##             Values and labels    N    Percent
##
##     1   'Democratic'            90   43.7 30.0
##     2   'Republican'            92   44.7 30.7
##     3   'Third-party'           24   11.7  8.0
##     8 M 'Don't know'            31        10.3
##     9 M 'Answer refused'        23         7.7
##    97 M 'Not applicable'        19         6.3
##    99 M 'Not asked in survey'   21         7.0
##
##    Remark:
##
##        These are actual data
```

```
##
## ===========================================================================
##
##    region 'Region of residence'
##
## ---------------------------------------------------------------------------
##
##    Storage mode: double
##    Measurement: nominal
##    Missing values: 97, 99
##
##             Values and labels    N    Percent
##
##     1   'Northeast'            88   34.6 29.3
##     2   'Midwest'              55   21.7 18.3
##     3   'South'                57   22.4 19.0
##     4   'West'                 54   21.3 18.0
##    97 M 'Not applicable'       25         8.3
##    99 M 'Not asked in survey'  21         7.0
##
##    Remark:
##
##        These are actual data
##
## ===========================================================================
##
##    income 'Household income'
##
##    "All things taken into account, how much do all members of your
##    household earn in total?"
##
## ---------------------------------------------------------------------------
##
##    Storage mode: double
##    Measurement: ratio
##    Missing values: -9
##
##        Values and labels     N     Percent
##
```

```
##        -9 M 'Refused'       10            3.3
##             (unlab.vld.)   290   100.0   96.7
##
##             Min:    247.953
##             Max:  11121.439
##            Mean:   2716.880
##        Std.Dev.:   2013.847
##        Skewness:      1.635
##        Kurtosis:      3.180
##           Miss.:     10.000
##
##    Remark:
##
##        These are actual data
```

We can proceed to print them out.

```
Write(description(data.for.codebook), file = "Polling-data-desc.txt")
Write(codebook(data.for.codebook), file = "Polling-data-cdbk.txt")
```

Let's practice the same thing with some real data - OLDA