# An Introduction to R
**The Basics of Data Management and Analysis in R**

**Political Science 2140**
**Lab Book**

John K. Wagner
University of New Mexico

# Contents

# Introduction

## 0.1 Purpose of the Lab Book

The following document is intended to provide references and the most basic instructions necessary for performing data management, alteration and analysis in R. It is meant to help assist the learning done in lab, not replace it.

## 0.2 Understanding the Code Provided

1. The code you will use in R is inside of closed boxes. This is meant to distinguish the codes/commands from the surrounding context.
2. Parts of code you should alter for your own use are in italics.
3. Some of the commands used require packages. Packages are packages of commands that R does not load for use by default. When they will be neccessary, these packages are either mentioned explicitly or precede the command(s) used and are separated by a line.
4. The first time a package appears in the lab book, a reminder to install the package is added prior to the command box. These reminders to install appear in rounded boxes with darkened backgrounds.

Package Example

```
install.packages("package")
```

```
library(package)
```

```
commandA(data)
commandB(data)
commandC(data)
commandD(data)
```

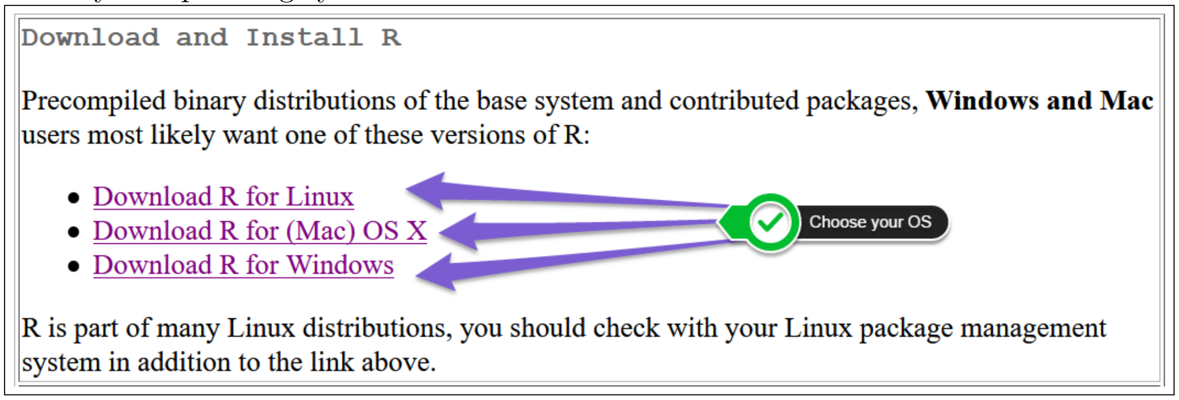
# Installing R

To use R Studio - our software of choice - you will need to install two things on your computer: R and R Studio. R holds all of the code necessary to run statistical analysis, and R Studio gives us a friendlier interface to interact with it in. To install R and R Studio, please follow the below directions *in order*. Installing R Studio before R can lead to problems in the future.

1. **Installing R:**

   To install R, navigate to cran.r-project.org. (You may also google "R cran" to find the official R webpage).

   Select your operating system[1]



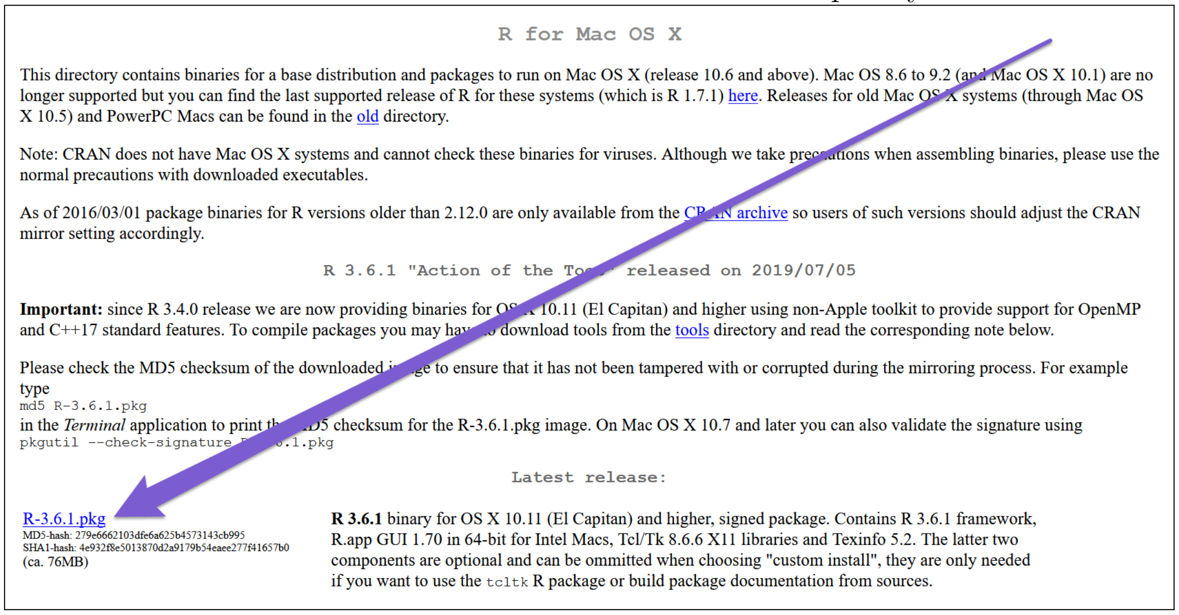
**Download and Install R**

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

   If you are using a Mac, then select the most recent R-3.x.x.pcg file. Open the downloaded file and follow the on-screen instructions to complete your installation.



**R for Mac OS X**

This directory contains binaries for a base distribution and packages to run on Mac OS X (release 10.6 and above). Mac OS 8.6 to 9.2 (and Mac OS X 10.1) are no longer supported but you can find the last supported release of R for these systems (which is R 1.7.1) here. Releases for old Mac OS X systems (through Mac OS X 10.5) and PowerPC Macs can be found in the old directory.

Note: CRAN does not have Mac OS X systems and cannot check these binaries for viruses. Although we take precautions when assembling binaries, please use the normal precautions with downloaded executables.

As of 2016/03/01 package binaries for R versions older than 2.12.0 are only available from the CRAN archive so users of such versions should adjust the CRAN mirror setting accordingly.

**R 3.6.1 "Action of the Toes" released on 2019/07/05**

**Important:** since R 3.4.0 release we are now providing binaries for OS X 10.11 (El Capitan) and higher using non-Apple toolkit to provide support for OpenMP and C++17 standard features. To compile packages you may have to download tools from the tools directory and read the corresponding note below.

Please check the MD5 checksum of the downloaded image to ensure that it has not been tampered with or corrupted during the mirroring process. For example type

```
md5 R-3.6.1.pkg
```

in the *Terminal* application to print the MD5 checksum for the R-3.6.1.pkg image. On Mac OS X 10.7 and later you can also validate the signature using

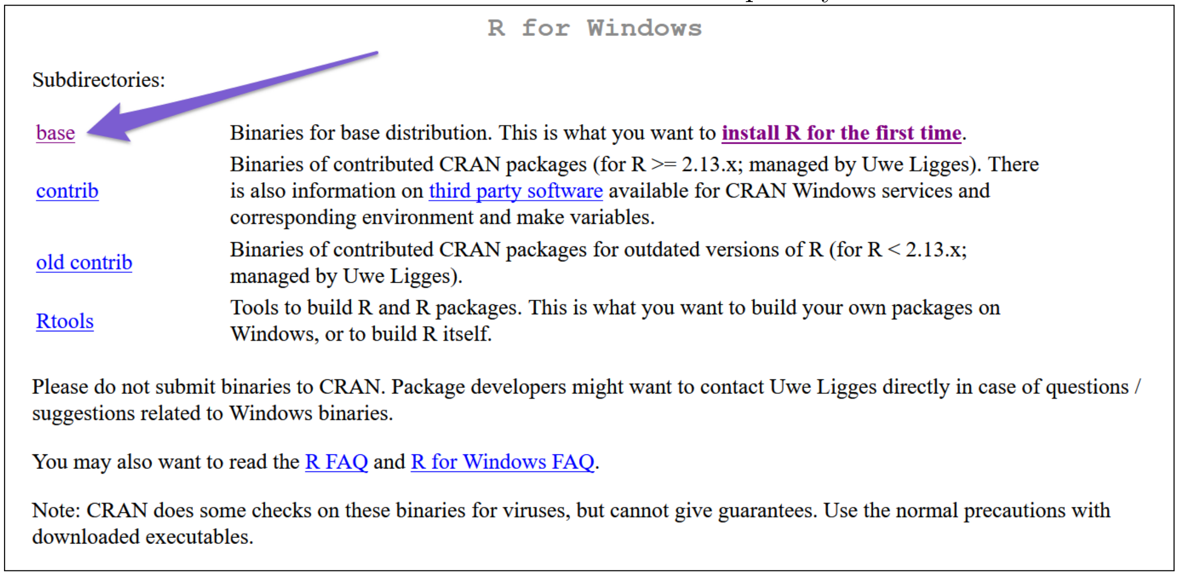
```
pkgutil --check-signature R-3.6.1.pkg
```

**Latest release:**

| | |
|---|---|
| R-3.6.1.pkg<br>MD5-hash: 279e6662103dfe6a625b4573143cb995<br>SHA1-hash: 4e932f8e5013870d2a9179b54eaee277f41657b0<br>(ca. 76MB) | **R 3.6.1** binary for OS X 10.11 (El Capitan) and higher, signed package. Contains R 3.6.1 framework, R.app GUI 1.70 in 64-bit for Intel Macs, Tcl/Tk 8.6.6 X11 libraries and Texinfo 5.2. The latter two components are optional and can be ommitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from sources. |

[1]If you are not using Windows, Mac OS X, or Linux; you may use the web version of R Studio. Found at rstudio.cloud, you can create an account and use R Studio. We don't recommend using rstudio.cloud unless you need to, as you it tends to run slower and requires you to upload data files in order to use it.

If you are using Windows, then select the "base" distribution of R. Download the indicated and follow the on-screen instructions to complete your installation.



R for Windows

Subdirectories:

| | |
|---|---|
| base | Binaries for base distribution. This is what you want to install R for the first time. |
| contrib | Binaries of contributed CRAN packages (for R >= 2.13.x; managed by Uwe Ligges). There is also information on third party software available for CRAN Windows services and corresponding environment and make variables. |
| old contrib | Binaries of contributed CRAN packages for outdated versions of R (for R < 2.13.x; managed by Uwe Ligges). |
| Rtools | Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself. |

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

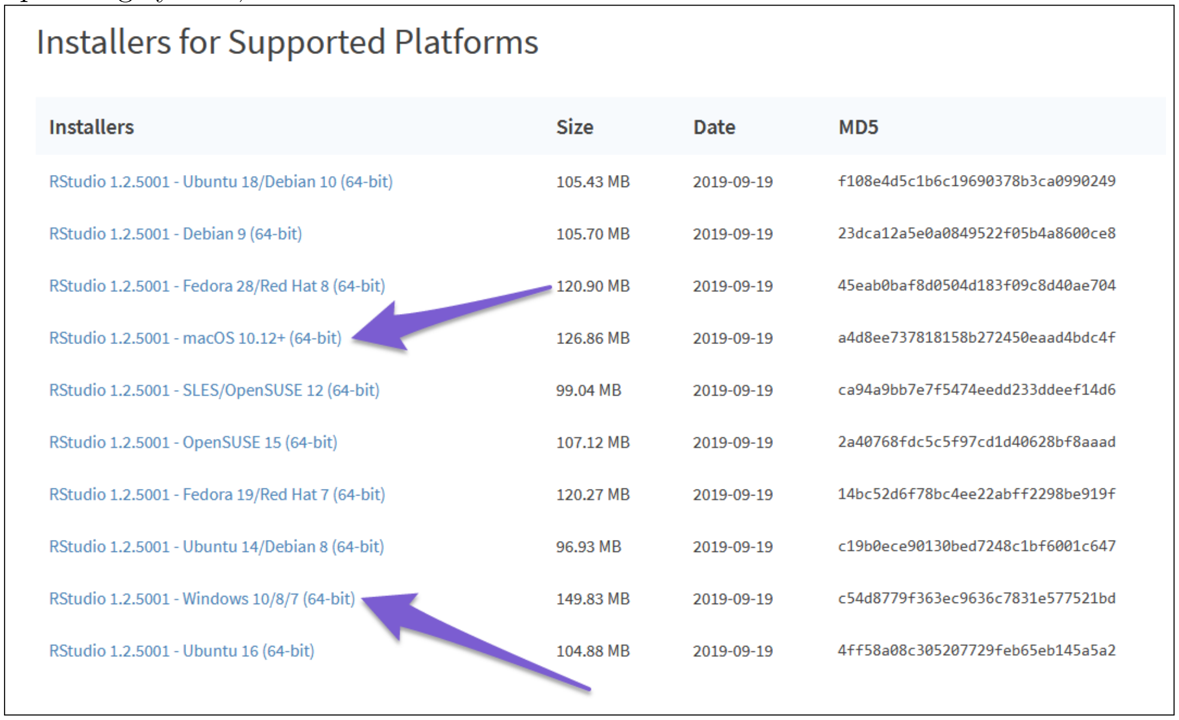
You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

If you are using Linux, then select your Linux distribution. The website will provide you with the commands necessary to install R.

2. **Installing R Studio:**

   To install R Studio, navigate to rstudio.com/products/rstudio/download. Select your operating system, and follow the installer's instructions.



Installers for Supported Platforms

| Installers | Size | Date | MD5 |
|---|---|---|---|
| RStudio 1.2.5001 - Ubuntu 18/Debian 10 (64-bit) | 105.43 MB | 2019-09-19 | f108e4d5c1b6c19690378b3ca0990249 |
| RStudio 1.2.5001 - Debian 9 (64-bit) | 105.70 MB | 2019-09-19 | 23dca12a5e0a0849522f05b4a8600ce8 |
| RStudio 1.2.5001 - Fedora 28/Red Hat 8 (64-bit) | 120.90 MB | 2019-09-19 | 45eab0baf8d0504d183f09c8d40ae704 |
| RStudio 1.2.5001 - macOS 10.12+ (64-bit) | 126.86 MB | 2019-09-19 | a4d8ee737818158b272450eaad4bdc4f |
| RStudio 1.2.5001 - SLES/OpenSUSE 12 (64-bit) | 99.04 MB | 2019-09-19 | ca94a9bb7e7f5474eedd233ddeef14d6 |
| RStudio 1.2.5001 - OpenSUSE 15 (64-bit) | 107.12 MB | 2019-09-19 | 2a40768fdc5c5f97cd1d40628bf8aaad |
| RStudio 1.2.5001 - Fedora 19/Red Hat 7 (64-bit) | 120.27 MB | 2019-09-19 | 14bc52d6f78bc4ee22abff2298be919f |
| RStudio 1.2.5001 - Ubuntu 14/Debian 8 (64-bit) | 96.93 MB | 2019-09-19 | c19b0ece90130bed7248c1bf6001c647 |
| RStudio 1.2.5001 - Windows 10/8/7 (64-bit) | 149.83 MB | 2019-09-19 | c54d8779f363ec9636c7831e577521bd |
| RStudio 1.2.5001 - Ubuntu 16 (64-bit) | 104.88 MB | 2019-09-19 | 4ff58a08c305207729feb65eb145a5a2 |

3. **Congratulations, you're ready to start using R Studio.**

# 1 The Basics

## 1.1 The R Screen

The R Studio screen is split into **4** areas.

1. The Console (Bottom Left) - where you input commands and where the output is usually displayed.
2. The File Editor and Data Browser Window (Top Left) - when open, this window allows you to look at a two-dimensional, cell-based display of your data and edit special R files.
3. The Environment & History Window (Top Right) - provides a list of objects (like datasets) currently loaded and usable in the Environment tab, and a history of all the commands you typed in in the History tab.
4. The Utility Tab (Bottom Right) - provides a list of files you can open in the Files tab, displays plots and graphs in the Plots tab, shows a list of currently installed packages in the Packages tab, and finally shows help documentation in the Help tab.

## 1.2 Loading Data into R

The first step to doing any data analysis in R is loading the data into R. If the data is in an R friendly file format - most commonly .Rda - then it can be opened like any other computer document. If the data is in another file format, it will need to be imported. Note that R can have multiple datasets loaded at once.

**Loading Datasets**

For R Data (.Rda files):
File > Open > Select the .rdata File

For Other Data Files (e.g. .xlsx, .csv, .dta):
File > Import > From File Type
Browse and Select File
Make Sure to Give Your Dataset a Name! (You'll be typing it a lot!)

## 1.3 Viewing and Editing Data in R

The second step of doing data analysis in R is viewing and sorting your data.

**Basic Data Viewing and Editing**

To browse your dataset like a spreadsheet: (Make sure to use a capital **V**!)

```
View(datasetName)
```

Basic Data Viewing and Editing (cont.)

Typically, when referring to a variable in a command you must refer to the dataset the variable is in as follows:

```
datasetName$variableName
```

Sorting can be useful to look for common categories when creating a table. To sort your data by a variable:

```
sort(datasetName$variableName)
```

If you "attach" a dataset, R will assume that you're always referencing variables in that dataset: an optional but big time saver. To attach a dataset:

```
attach(datasetName)
```

## 1.4 Viewing Summaries of the Data

The third step of doing data analysis is looking at some basic statistical attributes of your data and your variables.

Viewing Data Summaries

To view a list of all variables in your dataset:

```
names(datasetName)
```

To view the basic structure of all variables in your dataset:

```
str(datasetName)
```

To view a basic summary of a variable (e.g. Mean, Median, Min, Max):

```
summary(datasetName$variableName)
```

To view more a more detailed summary of a variable (e.g. N, SD, Kurtosis)

```
install.packages("psych")
```

```
library(psych)
describe(datasetName$variableName)
```

To view a cross-tabulation of a nominal or ordinal variable:

```
table(datasetName$variableName)
```

Viewing Data Summaries (cont.)

Or for an easier-to-read cross-tab:

```
install.packages("gmodels")
```

```
library(gmodels)
CrossTable(datasetName$variableName)
```

## 1.5 Creating and Editing Variables

A large part of data analysis is being able to generate and recode variables from an existing dataset or even data you've collected yourself. Many of the commands in this section rely on the *dplyr* package, which provides easier ways to manipulate variables in your dataset.

Recoding Variables

Before proceeding, make sure you have the *tidyverse* package installed and loaded. Installing the *tidyverse* collection of packages includes *dplyr* and a number of other useful tools we will use later on.

```
install.packages("tidyverse")
```

Once installed, we need to load the *tidyverse* package set to use it.

We do this with a library() command.

```
library(tidyverse)
```

To create a variable by applying mathematical functions to other variables:

The example here is addition, but you could also subtract, multiply and divide.

```
dataset <-
    mutate(dataset, newVariable = oldVariable1 + oldVariable2)
```

To recode a variable:

Recoding a variable involves altering the values of an existing variable and placing them into a new variable. For example, your data may have a variable for an individual's party membership. You may only care about membership in one party, so you would recode the party variable from having 15 values to only being a 0 or a 1. 0 for non-members of "Imaginary Party" and 1 for members.

```
dataset <-
    mutate(dataset,
        newVariable = case_when(oldVar == oldVal1 ~ newVal1,
                                oldVar != oldVal2 ~ newVal2))
```

## Recoding Variables (cont.)

mutate() creates a new variable based on an argument given to it.
case_when() provides an argument for mutate() which sets the New Variable equal to newVal1 if the Old Variable was equal to oldVal1, and newVal2 if the Old Variable was equal to oldVal2.
Note that the symbol between the oldVal and newVal is a tilde, typically the Shift + ‘ option on most keyboards, found above the Tab key.
! = is interpreted to mean ”not equal to” by *R*.

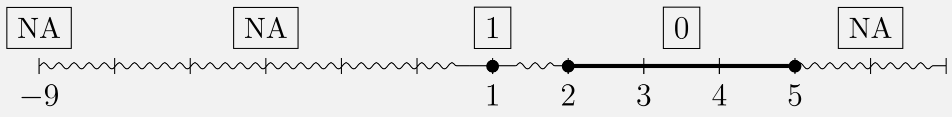
Recoding as a dummy:

Alternatively, we could recode a variable with many different categories into a dichotomous variable. The below example turns a variable with five categories into a dichotomous variable where the first category becomes one while the rest become zero. If we do not include a value in the mutate command and do not specify a new category using a logic of the original variable being greater than, less than, or not equal to a value; then the resulting new variable will treat unspecified values as missing. This is important if we want to exclude certain values from our analysis, say negative numbers or exceptionally large values which some datasets use to communicate reasons for missingness.

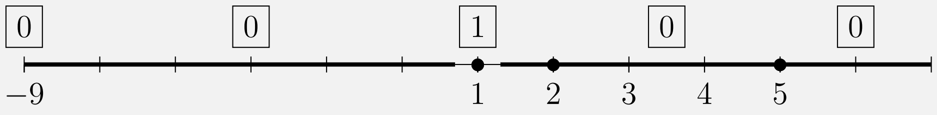
```
dataset <-
      mutate(dataset,
            newVariable = case_when(oldVar == oldVal1 ~ 1,
                                    oldVar == oldVal2 ~ 0,
                                    oldVar == oldVal3 ~ 0,
                                    oldVar == oldVal4 ~ 0,
                                    oldVar == oldVal5 ~ 0))
```

So if the above command were run, but it actually had a sixth category where -9 represents respondents who refuse to answer; those -9’s would become *NA*’s, R’s representation for missing data. If we recoded using *oldVar =! 1*, then the -9’s would become zeros. We can visualize this below.

Specifying each category (1 through 5) separately:



Using not-equal to (! =):



## 1.6 Some Final Tips on Packages, Finding Help in R, and Keyboard Shortcuts

### Packages

Packages are sets of useful commands that R does not include by default.
If we show a command that requires a library() command, then the command needs a package installed and loaded to work!

First, you need to install the package. This only needs to be done once per computer.

```
install.packages("packageName")
```

Then load the package so we can use the commands! This needs to be done once per session.

```
library(packageName)
```

Now we can access all of the commands included in the package!

### Help!

If you need any help with a command, try ...

```
?commandName
```

... and R Studio will open the documentation for the command.
As always, Google can be a great resource too if you're looking for how to use a command or do something in R!

### Shortcuts!

When typing in commands, variables, or datasets into R Studio's Command Window a list will pop-up, showing terms that may complete what you're typing. Hitting TAB will insert R Studio's current suggested command or variable. This means you only have to type out a little bit of the term into the console, and R Studio can fill in the rest!
For example typing...

```
summ
```

and hitting TAB... will produce

```
summary(|)
```

with the cursor ready to fill in the parentheses!

## 1.7 Using R-Scripts (with Example Script)

**R Scripts**

An R script is a file containing a set of R commands.

R scripts are therefore great for keeping track of and replicating all of the commands you may have used to get your results.

To create an R script in RStudio:

```
File > New File > R Script
```

The R Script should open in the upper left section of RStudio, above the Console section.

It is a good idea to immediately save your R script on creation using:

```
File > Save As
```

To run a single command in the R Script:
Place the cursor on the line you wish to run, and hit the run button at the top right corner of the R Script.

To run every command in the R Script in order:
Click the Source button at the top of the R Script.

Alternatively, click the "Source on Save" box at the top of the R script, then click on the Floppy Disk save symbol. This will save the R Script *and* run every command in the R Script.

Example of a Well-Formatted R Script

```r
# TITLE
# Author: NAME, EMAIL

# Clear Existing Data and Objects
rm(list = ls())

# Libraries
library(haven)
library(psych)
library(tidyverse)

# Load Datasets
ATP16 <- read_dta("ATP W16.dta") # Pew American Trends Panel 16

# Create Vars
## R, Republican Dummy Variable
ATP16 <- mutate(ATP16, R = case_when(F_PARTYSUM_FINAL == 1 ~ 1,
                                     F_PARTYSUM_FINAL != 1 ~ 0))
## D, Democrat Dummy Variable
ATP16 <- mutate(ATP16, D = case_when(F_PARTYSUM_FINAL == 2 ~ 1,
                                     F_PARTYSUM_FINAL != 2 ~ 0))

# Descriptive Stats
describe(ATP16$R)
describe(ATP16$D)

# Analysis, Chi-Square Test
CrossTable(ATP16$D, ATP16$R, prop.r = FALSE, prop.t = FALSE, chisq = TRUE,
                                                            format = "SPSS")
```

# 2 Descriptive Stats and Bivariate Hypothesis Testing

## 2.1 Descriptive Statistics

We seek descriptive statistics to understand the basic attributes of our data and variables. Typically, descriptive statistics involve commonly understood stats such as the mean, median, range and standard deviation. In some specific case, descriptive statistics can also include the interquartile range, skew and kurtosis.

**Descriptive Statistics**

Finding the mean, median, minimum, maximum and inter-quartile range:

```
summary(dataset$varName)
```

Finding the N, standard deviation, range, skew and kurtosis:

```
library(psych)
describe(dataset$varName)
```

Finding the mode:
Hint: Look for the value that occurs most in the table.

```
table(dataset$varName)
```

## 2.2 Subsetting Data

Subsetting data is an *incredibly* common tool we use when performing data management and analysis. By subsetting data, we mean selecting a portion of the data - hence *subset*ing - in order to understand or alter it. For example, maybe we wish to know the mean education levels for only one sex. We would select a subset of the data where only men or women are included, to find the mean education level of only those men or women.

**Subsetting Data**

Select *Variable 1* when *Variable 2* is *equal to* a given *value**

```
dataset$var1[dataset$var2 == value]
```

Select *Variable 1* when *Variable 2* is ***not*** *equal to* a given *value**

```
dataset$var1[dataset$var2 != value]
```

Select *Variable 1* when *Variable 2* is *greater than* a given *value**
(use >= for *greater than or equal to)*

```
dataset$var1[dataset$var2 > value]
```

Subsetting Data (cont.)

Select *Variable 1* when *Variable 2* is *less than* a given *value**
(use <= for *less than or equal to)*

```
dataset$var1[dataset$var2 < value]
```

Example: Means of Income for Men and Women

```
mean(anes2016$income[anes2016$sex == 1])
mean(anes2016$income[anes2016$sex == 0])
```

* Note that *Variable 2* can also be *Variable 1*, so the below is a valid command.

```
mean(anes2016$income[anes2016$income >= 1])
```

## 2.3 Recoding Using Subsetting

In cases where you want to make alterations to an existing variable, it is sometimes useful to make a quick recode with using R's subset functionalities. This is especially useful for recoding values to be missing (aka null).

Recoding Using Subset Brackets

```
dataset$variable[dataset$variable == value] <- NA
dataset$variable[dataset$variable <  value] <- NA
dataset$variable[dataset$variable <= value] <- NA
dataset$variable[dataset$variable >  value] <- NA
dataset$variable[dataset$variable >= value] <- NA
dataset$variable[dataset$variable != value] <- NA
```

Note that you could recode as missing if a variable is:
equal to (==) a value
greater than (>); or greater than or equal to (>=) a value
less than (<); or less than or equal to (<=) a value
not equal to (! =) a value

## 2.4 Chi-Square Test (Categorical X by Categorical Y)

A Chi-Square test is used to test for a correlation between a Categorical Independent Variable and a Categorical Dependent Variable.

Chi-Square Test (Categorical X by Categorical Y)

Chi-Square Table:

```
install.packages("descr")
```

```
library(descr)
```

```
CrossTable(dataset$varY, dataset$varX, prop.t = FALSE, chisq = TRUE,
                                                  format = "SPSS")
```

Basic Interpretation:

Use the p-value of the Pearson's Chi-squared Test to understand if the relationship between X and Y is statistically significant.

Use the cross-tabulation to interpret the direction of the relationship.

## 2.5 T-Test (Dichotomous X by Continuous Y)

A T-Test is used to test for a correlation between a Dichotomous Independent Variable and a Continuous Dependent Variable.

T-Test (Dichotomous X by Continuous Y)

Student's T-Test:

```
t.test(dataset$varY ~ dataset$varX, var.equal = TRUE)
```

Basic Interpretation:

Use the p-value to determine the statistical significance of the relationship between X and Y.

Look at which group mean is lower, that group has a significantly lower in Y than the other group and vice versa.

## 2.6 Correlations (Continuous X by Continuous Y)

A correlation test is specifically used to determine whether a Continuous Independent Variable and Continuous Dependent Variable are, well, correlated.

Correlations (Continuous X by Continuous Y)

Basic Correlation between X and Y:

```
cor.test(dataset$varY, dataset$varX, method = "pearson",
                                     use = "complete.obs")
```

Correlations (Continuous X by Continuous Y)(cont.)

Basic Interpretation:

If the p-value is less than 0.05, then the relationship between continuous variables X and Y is statistically significant.

Under "sample estimates ... cor" is the correlational coefficient. This number tells us the resulting change in Y for a 1 unit increase in X.

## 2.7 Correlation Tables

Sometimes, we may want to look at the correlations between multiple variables. This will be especially useful when checking for multicollinearity later on when we are using regressions. In this example, we have three variables. However, you could use as many as you desire!

Correlation Table

```
install.packages("Hmisc")
```

```
library(Hmisc)
rcorr(cbind(dataset$varY, dataset$varX, dataset$varZ))
```

Basic Interpretation:

*rcorr()* produces three cross-tables (or a table for each variable provided).

The first reports the correlational coefficient for each pair of variables.

The second reports the n after excluding missing observations for the pair of variables.

The third reports the p-values for the relationship between each pair of variables.

# 3 Bivariate Regressions, Plotting, and Paper-Ready Tables

## 3.1 Checking for Normal Distributions

When running correlations or regressions, it is important to check to see if your data is "normal".

**Checking for Normalcy**

The best way to check, is to look at the histogram for each variable you will be correlating or regressing. Simple Histogram:

```
hist(datasetName$varName)
```

**Checking for Normalcy (cont.)**

Nicer Histogram with a Normal Curve for Reference:

```
library(tidyverse)
---
ggplot(datasetName, aes(varName)) +
  geom_histogram(binwidth = 2,
      color = "black",
      aes(y = ..density.., fill = ..count..)) +
  stat_function(fun = dnorm,
      color = "blue",
      args = list(
        mean = mean(datasetName$varName[!is.na(datasetName$varName)]),
        sd = sd(datasetName$varName[!is.na(datasetName$varName)]) ) )
```

## 3.2 Plotting a Line of Best Fit

**Plotting Lines of Best Fit**

Plotting a Line of Best Fit:

```
library(tidyverse)
---
ggplot(dataset, aes(x = varX, y = varY)) +
    geom_point() +
    geom_smooth(method = lm)
```

## 3.3 Bivariate OLS Regression (Any X, Continuous Y)

Also referred to as a bivariate linear regression, this regression checks to see if one independent variable - of any type and properly prepared - leads to a significant change in a *continuous* dependent variable.

OLS Regression

Bivariate Linear Ordinary Least Squares Regression:

```
modelName <- lm(dependentVar ~ independentVar, data = datasetName)
summary(modelName)
```

Interpretation:

The p-value should be at the end of the output, and if p < 0.05, then the relationship is significant.

The Estimate of X represents the change in Y for each 1 unit increase in X.

## 3.4 Paper-Ready Tables

One issue with the output and results offered up by R is that they are - well - quite *unsightly*. One would hope to never gaze upon raw results in an actual paper. As such, transforming our statistics and results into tables - and later graphs - which are ready to be put directly into a paper is a key skill. Below are the necessary commands to:

Paper-Ready Tables

Remember to install Stargazer before beginning to use it:

```
install.packages("stargazer")
```

Summary Statistics:

```
library(stargazer)
stargazer(as.data.frame(dataset[c("var1", "var2")]), type = "text")
```

Note: You can use more than 2 variables by adding a comma and another variable.

Regression Table:

```
library(stargazer)
model1 <- lm(dependentVar ~ independentVar, data = datasetName)
model2 <- lm(dependentVar ~ independentVar, data = datasetName)
stargazer(model1, model2, type = "text")
```

## Paper-Ready Tables Example (cont.)

Example with Variable Labels:

```
model1 <- lm(conTherm ~ libTherm, data = anes_timeseries_2016)
stargazer(model1,
          dep.var.labels = "Conservative Feeling Thermometer",
          covariate.labels = c("Liberal Thermometer"), type = "text")
```

Example Output:

```
=================================================================
                              Dependent variable:
                        -----------------------------------------
                        Conservative Feeling Thermometer
-----------------------------------------------------------------
Liberal Thermometer                 -0.413***
                                     (0.014)

Constant                            77.470***
                                     (0.821)

-----------------------------------------------------------------
Observations                          3,566
R2                                    0.193
Adjusted R2                           0.192
Residual Std. Error           22.415 (df = 3564)
F Statistic               850.692*** (df = 1; 3564)
=================================================================
Note:                              *p<0.1; **p<0.05; ***p<0.01
```

Note: You can Copy + Paste this table into Word!
Just make sure the table is formatted as Lucida Console, 10pt font.

Additional Note: If you have more the one covariate, simply add a comma after the first label and add the label for the next variable. The labels *must* be placed in the same order the variables are listed in the lm() command.

Example:

```
 model1 <- lm(conTherm ~ libTherm + age,
                                        data = anes_timeseries_2016)
stargazer(model1,
          dep.var.labels = "Conservative Feeling Thermometer",
          covariate.labels = c("Liberal Thermometer", "Age"),
          type = "text")
```

# 4 Multiple Regressions and Coefficient Plots

## 4.1 Multiple Regression (Any X, Continuous Y, Many Z)

A multiple linear regression attempts to predict a dependent continuous variable using an independent variable of interest *while* controlling for other variables.

**Multiple Regression**

Multiple Regression Command:

```
modelName <- lm(depVar ~ indVar1 + indVar2 ... indVarn, data = dataset)
```

Viewing the Results:

```
summary(modelName)
```

Interpreting the Results:

Is the p-value of the model significant? If so, then look at the p-values for each variable.

If the p-value of a variable is significant, then that variable is significantly related to the dependent variable.

Then look at the independent variable's estimate to see the amount the dependent variable will change based on a 1-value increase in the independent variable.

Remember, you can use stargazer() to view the results as well, and use it to paste your results into your paper! See Lab 3 for more on using Stargazer.

```
library(stargazer)
stargazer(modelName, type = "text")
```

## 4.2 Prepping Categorical Variables for Multiple Regression

R needs to know a variable is categorical (with more than 2 categories) before it is entered into the regression. There are two ways of prepping categorical variables: dummy-ing and factorizing.

*Dummy-ing Out Categorical Variables:*

Dummy-ing Out variables refers to taking a variable of multiple values, and transforming it into a set of variables each with the values 0 or 1. To do this, we will use the ifelse() command.

The *ifelse()* command is used to create a variable that is 1 if a condition is met, and 0 if it not. In the case of the below commands, if our categorical variable (*var)* is equal to a *value*$_1$, then our new dummy variable, *var*$_1$ will be set equal to 1. In all other cases, this new *var*$_1$ will be 0. Repeat the *ifelse()* command until every category of our original var

(*var*) has its own dummy variable. Just remember to exclude one category from the regression itself to act as a reference category.

Prepping Categorical Dependent Variables for Multiple Regression (Dummy-ing)

```
dataset$var1 <- ifelse(dataset$var == value1, 1, 0)
dataset$var2 <- ifelse(dataset$var == value2, 1, 0)
...
dataset$varn <- ifelse(dataset$var == valuen, 1, 0)
```

*Factorizing a Categorical Variable*

Factorizing a categorical variable involves manually telling R each value of the categorical variable and what label it represents. Therefore R can then dummy-out the variable under the hood each time we use it.

Prepping Categorical Dependent Variables for Multiple Regression (Factorizing)

First, check to see if the variable is recognized as a categorical (factor) variable.

```
is.factor(dataset$var)
```

If this command returns TRUE, you're good to go!
If this command returns FALSE, you'll need to do the following:

```
dataset$var <- factor(dataset$var, levels = c(0, 1 ,...n),
                      labels = c("label1", "label2", ... "labeln"))
```

Note!
R will use the first factor level as the reference category, so make sure to put the reference category you want first. The levels do not need to go in numeric order.

## 4.3 Coefficient Plots

Coefficient plots allow us to visually display the sometimes overwhelming numbers created by regression.

Coefficient Plots

Required Package:

```
install.packages("arm")
```

```
library(arm)
```

## Coefficient Plots (cont.)

Simple Coefficient Plot:

```
coefplot(modelName)
```

Interpreting a Coefficient Plot:

Each dot on a coefficent plot represents the regression estimate for a given variable.

The lines extending from each dot represent the 95% confidence interval surrounding that estimate.

If a confidence interval crosses the 0 line, then that variable has an insignificant relationship with the dependent variable.

If a confidence interval does not touch the 0 line, then that variable does have a significant relationship with the dependent variable.

If a independent variable is significant, then we are 95% confident that the value of a 1 unit increase in the variable leads to a change in the dependent variable equal to a value somewhere along the confidence interval line.

Advanced Coefficient Plot (including Options):

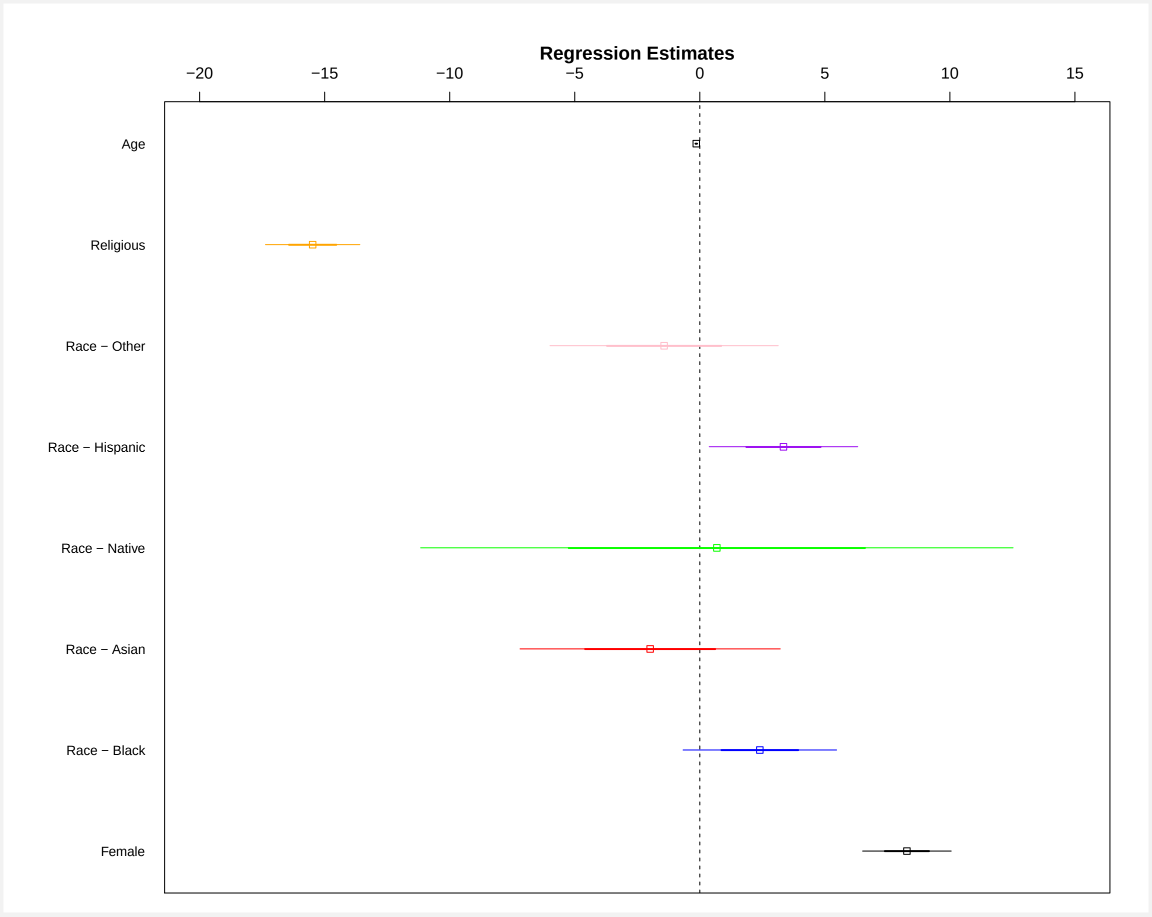
```
coefplot(modelName,
          pch.pts = pointShape,
          frame.plot = TRUE,
          xlim = c(lowerBound, upperBound),
          col.pts = c("color1", "color2", ... , "colorn"),
          mar = c(bottomMargin, leftMargin, topMargin, rightMargin),
          varnames = c("label1", "label2", ... "labeln"))
```

*pointshape* will change the shape of the dot on the plot (e.g. circle, square, etc)
*lowerbound* and *upperbound* alter the min and max values of the graph plane.
*margins* set the margins around the plot. It is common to change the left margin if the variable names are too long (and therefore run off the screen).
*label*$_n$ is used to label each variable according to a chosen name. The labels must be in the same order as in the original *lm* model, though *label*$_1$ will always be "Constant".

## Coefficient Plots (cont.)

Example Plot:

```
model.1 <- lm(TransTherm ~ FEMALE + RACE + RELIGION + AGE,
                data = anes_timeseries_2016)
coefplot(model.1,
         pch.pts = 0,
         frame.plot = TRUE,
         xlim = c(-20, 15),
         col.pts = c("black", "blue", "red", "green", "purple",
           "pink", "orange"),
         mar = c(1,6,5.1,2),
         varnames = c("Constant", "Female", "Race - Black",
           "Race - Asian", "Race - Native", "Race - Hispanic",
           "Race - Other", "Religious", "Age"))
```



# 5 Interaction Effects and Logistic Regressions (Any X, Dummy Y, Many Z)

## 5.1 Interaction Effects

An interaction effect is useful if you think the effect of your independent variable is contingent on the presence of another independent variable.

**Interaction Effects**

Required Package:

```
library(tidyverse)
```

Multiple Regression with Interaction between $X_1$ and $X_2$:

```
modelName <- lm(depVar ~ X1 * X2 + X2 + ... Xn)
```

Interpretation:

Check to see if the interaction, $X_1$:$X_2$, is significant ($p < .05$).

If so, the estimate of $X_1$ is the value of $X_1$ when $X_2$ is zero; and vice versa.

Note: It might be useful to center continuous variables around the mean, so when $X_1$ is 0, it is at the mean.

However, the best way to interpret a significant interaction is to graph it...

Plotting an Interaction ($X_2$ Variable *must* be Categorical):

```
qplot(x = X1, y = depVar, facets = ~X2, data = na.omit(dataset)) +
    geom_smooth(method = "lm")
```

One additional way to interpret interactions is using predicted probabilities. See the below section on Predicted Probabilities for more info!

## 5.2 Logistic Regression (Any X, Dichotomous Y, Many Z)

Logistic regressions, similar to OLS / Linear Regressions, attempt to predict a dependent variable using an independent variable *while* controlling for any number of other control variables. However instead of a continuous dependent variable, logits - short for logistic regressions - work on *dichotomous* dependent variables. A crucial side effect of the dependent variable being dichotomous is that it drastically complicates interpretation of the regression results.

Logistic Regression

Logistic Regression Model:

```
modelName <- glm(depVar ~ X1 + X2 + X2 + ... Xn, family = binomial,
                  data = dataset)
summary(modelName)
```

Interpretation:

A logistic regression tells us the contribution of each independent variable to the likelihood that our dependent variable will be present (have a value of 1).

If $X_n$ is significant (p < .05), then $X_n$ significantly alters the likelihood that our *depVar* will be present.

Unfortunately, we cannot directly interpret the coefficients of a logistic regression like we would a linear regression. For something similar, we use odds-ratio coefficients ...

Required Package for Odds Ratios:

```
install.packages("mfx")
```

```
library(mfx)
```

Odds Ratio Coefficients for Logistic Regression:

```
logitor(depVar ~ X1 + X2 + X2 + ... Xn, data = dataset)
```

Interpretation:

Check to see if $X_n$ is significant (p < .05).

If so, the odd-ratio is the increased likelihood of your outcome for a 1-unit increase in$X_n$.

For example, if the odds-ratio is 0.5, then the increase of $X_n$ by 1 will correspond to a .5 odds that our *depVar* is 1.

However, it is often more useful to look at the predicted probabilities of the *depVar* for a given $X_n$...

## 5.3 Predicted Probabilities

When discussing the results of a logistic regression, it is often useful to discuss the likelihood that your binary outcome will occur when your independent and/or control variables are of a certain variable
(e.g. A man, all other variables at their means, has an 87% chance of being registered to vote while a woman has a 90% chance.)

Predicted Probabilities

Required Package:

```
install.packages("ggeffects")
```

```
library(ggeffects)
```

Probabilities Given a Variable, $X_i$, with all Other Variables at their Means*:

```
ggpredict(modelName, terms = "Xi")
```

* If a variable is defined as a factor, R will choose the first category.

Probabilities Given a Variable, $X_i$, with Specified Values of Other Variables:

```
ggpredict(modelName, terms = "Xi", condition = c(X1 = V1,
                                    X2 = V2, ... Xn = Vn))
```

Probabilities Given Multiple Variables:

```
ggpredict(modelName, terms = c("Xi1", "Xi2"), condition = c(X1 = V1,
                                    X2 = V2, ... Xn = Vn))
```

Plotting Probabilities*:

```
objectName <- ggpredict(modelName, terms = "Xi")
plot(objectName, log.y = TRUE)
```

* You can use any of the other ggpredict() commands when plotting.

Notes on Interpretation of ggpredict() Commands:

First, look at top of the output for the "x = $X_i$", this will tell you what x means in the rest of the output.

Second, look at the bottom of the output for the section on "Adjusted for:" This tells you the values ggpredict() assumes for all of the covariates in your model. (It will not include variables you manually specified).

Third, look at the cross-table in the middle of the output. This tells you the probability of your dependent variable being present for each value of $X_i$.

You can use ggpredict() with Ordinal Logistic Regressions and Linear Regressions as well! Just remove *log.y = TRUE* from the plot() command.

## 5.4 Paper-Ready Tables: Logit Edition

While we have already covered creating paper-ready tables using stargazer, the necessity of reporting odds ratio coefficients complicates creating a paper-ready table for a logit.

Paper-Ready Tables: Logit Edition

Stargazer can take your *modelName* the same as it would a Regression:

```
stargazer(modelName, type = "text")
```

However, we may want to include odds ratios and confidence intervals in our tables.

Stargazer with Odds Ratios and Confidence Intervals:

```
OR.vector <- exp(modelName$coef)
CI.vector <- exp(confint(modelName))
p.values <- summary(modelName)$coefficients[, 4]
stargazer(modelName, coef = list(OR.vector), ci = TRUE,
          ci.custom = list(CI.vector), p = list(p.values),
          type = "text")
```

## 5.5 Ordinal Logistic Regressions

An ordinal logistic regression is a version of a logistic regression for ordinal (rather than dichotomous) variables.

Ordinal Logistic Regressions

Required Package:

```
install.packages("ordinal")
```

```
library(ordinal)
```

Ordered Logistic Regression:

```
modelName <- clm(depVar ~ X1 + X2 + X2 + ... Xn, data = dataset)
summary(modelName)
```

Interpretation:

Check to see if $X_n$ is significant (p < .05).

If so, look at the predicted probabilities (see the Predicted Probabilities section of this Lab on how to generate probabilities) for each variable to understand $X_n$'s effect on your *depVar*.

## Glossary of Terms

**Categorical Variable**
Sometimes referred to as a nominal or factor variable, categorical variables consist of discrete categories where the order of the categories does *not* matter. Examples include religion, race, and gender.

**Coding**
In data analysis, coding refers to the numeric values representing each category of a variable. For example, if a person in a dataset is male, then the coding of the variable ”sex” for that person might be 0. If said person were female, then the coding of ”sex” might be 1 for that person. Recoding is the act of altering the numeric values underlying the categories of a variables. This may be done for computation reasons (e.g. dummy variables are typically coded 0 or 1) or to make variables more parsimonious so we can more meaningfully interpret them.

**Dataset**
A dataset is a collection of many datum - pieces of information - organized into a matrix. In most datasets, columns of the matrix represent a variable while rows represent observations, observations which often but not always have a known value for each variable. For example, a column in a dataset may represent the GDP of each observation with each row representing a state.

**Chi-Square Test**
A chi-square test provides the correlation between a categorical X and categorical Y variable.

**Console**
In R, the console is the portion of the screen where R actually runs its commands and where R outputs its results. When running an R script, the commands are actually fed sequentially to the console.

**Continuous Variable**
Sometimes also referred to as an interval variable, a continuous variable consists of an infinite number of values - in theory at least. Examples include time or money. In practice, ordinal variables of 5, 6 or 7 + categories can often be treated as continuous for simplicity's sake.

**Correlation**
In general terms, a correlation is when two variables tend to change together. A correlation test in particular provides the correlation between a continuous X and continuous Y variable.

**Correlation Table**
A table providing correlation coefficients between two or more variables. Often used to determine if multi-colinearity is present in a regression analysis.

**Cross-Tabulation**
See *Tables.*

**Descriptive Statistics**
Descriptive statistics are statistics which describe the distributions and attributes of one or more variables. Sometimes referred to as summary statistics, the descriptive statistics most commonly referenced in data analysis are the mean, median, mode, range and standard deviation.

**Dichotomous Variable**
Often called a dummy variable, dichotomous variables are categorical variables that consist of no more and no less than *two* categories.

**Dummy Variable**
See *Dichotomous Variable.*

**Histogram**
A plot displaying the distribution of a single variable. Histograms are commonly used to check for normalcy.

**Interaction Effects**
In regressions, independent variables are treated as if they affect the dependent variable in isolation from one another. An interaction effect is used to look at the *combined* effect of two independent variables on the dependent.

**Interval Variable**
See *Continuous Variable.*

**library()**
In R, the *library()* command loads a package's commands and function for you to use. Once a package has been installed on a computer, it's commands are not always loaded for use. This is because you can install thousands of packages, and having them all loaded would be difficult to manage. As such, every time you start a new session of R, you must load the package using the *library()* command before you can use its commands. For more info on packages, see *Packages in R.*

**Line of Best Fit**
A line of best fit plots the line representing the slope of the line which best represents the relationship between a continuous X and continuous Y variable.

**Linear Regression**
See *Regression.*

**Logistic Regression**
A logistic regression estimates coefficients for a dichotomous dependent variable's relationship with one or more dependent variables. Unlike a typical OLS regression, this

does not represent the slope of the line and cannot be interpreted as such. Instead logistic regressions should be altered to provide odds ratio coefficients which communicate the change in the odds that Y occurs / is present.

**Multiple Regression**
A multiple regression is a regression, typically an OLS one, which includes multiple independent variables.

**Nominal Variable**
See *Categorical Variable.*

**OLS Regression**
Standing for Ordinary Least Squares Regression, this type of regression uses a specific estimation technique for coefficients ideal for a continuous dependent variable. For info on regressions in general see *Regression.*

**Ordinal Logistic Regression**
An ordinal logistic regression is a version of a logistic regression which uses an ordinal dependent variable instead of a dichotomous one.

**Ordinal Variable**
Similar to a categorical variable, ordinal variables consist of discrete categories. However, order *does* matter for ordinal variables. Examples may include education, a Likert scale, or a left-right ideological scale.

**Packages in R**
A package in R is a collection of commands, functions and/or data bundled into an easy-to-install and load kit. One of R's greatest advantages is that any user can create a package for anyone to use. However, in order to avoid overwhelming users with packages, commands and data; a package must first be installed using the *install.packages()* command. It must then be loaded each session with a *library()* command. It is recommended that you place all *library()* commands at the start of an R script, so the commands used in the script will always be loaded for use.

**Predicted Probablities (Predicted Values)**
Predicted probabilities - or values - calculate the predicted value of a dependent variable given certain values of an independent variables(s) from the results of a regression. For example, a regression may tell us that women are more likely to vote than men. Using predicted probabilities, we may discover that women are have a 65% chance of voting and men have a 62% chance.

**R Scripts**
R Scripts are a text file which contain a series of commands for R to run. An R script runs its list of commands in sequential order. You may run one line in a script by pressing *Run* or the whole script of commands by pressing *Source.* R scripts are invaluable because they allow you to easily trace-back to potential mistakes in your commands, coding or analysis.

They also more easily allow others to check and reproduce your work.

**Recoding**
See *Coding.*

**Regression**
A regression estimates the relationship between one dependent variable and one or more independent variables as a line. The relationship between the dependent variable and each independent variable is expressed as a coefficent, which represents the slope of the line. Regressions also create a constant which represents the Y-axis intercept of the line.

**Scripts**
See *R Scripts.*

**stargazer()**
Stargazer is a package used to convert the text output of regressions and cross-tabulations into a format ready for use in an actual paper.

**Subsetting**
Subsetting is the selection of a portion of a variable or dataset matching a condition. This is often done in order to recode or view descriptive statistics of the subset matching the condition. In R, subsetting is done using [ ] brackets.

**Summary Statistics**
See *Descriptive Statistics.*

**Tables**
A table is a two-dimensional matrix consisting of columns and rows. Tables are typically used to display descriptive statistics or compare the distributions of two variables.

**T-Test**
A T-Test provides the correlation between a dummy X and continuous Y variable.

**tidyverse()**
Tidyverse is a notable R package in its widespread usage, and in the special affordances it allows pertaining to coding and recoding variables in R. [Fun fact: The team that makes R Studio also makes the tidyverse package]

**Variable Types**
See entries on *Categorical Variables*, *Continuous Variables* or *Ordinal Variables.*