# # # # # # # # # # # # #

# TITLE: 2.1 Univariate Statistics in R

# SOCI832: Advanced Research Methods

# Week 2: Linear Regression (1 of 3 files)

# Author: Nicholas Harrigan

# Last updated: 4 Aug 2018

# # # # # # # # # # # # #

#

# WHAT DOES THIS SCRIPT DO?

# \* Introduces methods for making univariate statistics in R

#

# WHAT DOES THIS TEACH ME ABOUT R?

# (1) Basic commands for univariate statistics

# (2) Handy packages and tricks for getting your data in

# publishable formats (e.g. table out; summary stats; etc.)

#

# WHAT DOES THIS TEACH ME ABOUT STATISTICS?

# (1) Honestly most of the statistical knowledge is assumed

# as it can be found in a simple google search or any

# simple stats textbook.

#

# TIPS, COMMENTS, TRIVIA

# \* The introduction to the package summarytools,

# continues into the next script (Week 2, Part 2),

# because one of the tools creates cross-tabulations

# of two variables (i.e. a bivariate cross tabulation)

# Structure part 3

# \* regression

# \* part 1: running basic models, including R-square

# \* and exporting your data

# \* part 2: running multiple parallel models - heirarchical

# \* part 3: running diagnostics

# \* part 4: what to do if you fail diagnostics - robust etc

# STUFF YOU NEED TO DO BEFORE STARTING

# 1. Change this to your working directory

setwd("C:/G/2018, SOCI832/Datasets/AES 2013/")

# 2. Put the file "elect\_2013.csv" into that folder

# This file can be found here:

# https://mqsociology.github.io/learn-r/soci832/elect\_2013.csv

# 3. Keep the codebook openned in a browser so you

# can refer to it when you need it. The codebook is here:

# https://mqsociology.github.io/learn-r/soci832/codebook%20aes%202013.html

#

# START HERE

# Import the data

elect\_2013 <- read.csv("elect\_2013.csv") # loads dataset

# This command gets rid of the first column

# which is not needed.

# FYI the command works by saying

# "copy all columns except the first".

# NOTE: Only run this command once after you run the

# 'read.csv' command. Each time you run it, it deletes

# the first variable.

elect\_2013 <- elect\_2013[,2:ncol(elect\_2013)]

######################################

# LESSON 1: BASIC UNIVARIATE FUNCTIONS

######################################

# This section introduces the basic

# functions for running univariate

# statistics. In most cases there are

# are easier commands to run, but we

# will learn about those later.

######################################

# MEAN

# Mean is just the sum of a variable/number of cases

# There is a built in function that calcuates the mean

mean(elect\_2013$likelihood\_vote)

# Oh damn! We got:

# [1] NA

# This happens because there are missing values in

# the variable 'likelihood\_vote'.

# Why are there missing values? Because some people didn't

# answer that survey question.

# There is an easy way to deal with this.

# We add one more argument\* to the function\*\* 'mean()'

# The argument is 'na.rm = TRUE'

mean(elect\_2013$likelihood\_vote, na.rm = TRUE)

# You should get:

# [1] 4.362201

# We can get the median with:

median(elect\_2013$likelihood\_vote, na.rm = TRUE)

# We can also extract various quartiles and quintiles

# The default setting gives us min, max, 25%, 50%, and 75%

quantile(elect\_2013$likelihood\_vote, na.rm = TRUE)

# You should see the following in the console window:

# 0% 25% 50% 75% 100%

# 1 4 5 5 5

# We can also specify particular probablities with

# the argument 'probs ='

quantile(elect\_2013$likelihood\_vote, na.rm = TRUE,

probs = c(0,0.2,0.4,0.6,0.8,1))

# You should get the min, max, and quintiles:

# 0% 20% 40% 60% 80% 100%

# 1 4 5 5 5 5

# NOTE: when we provide the 'probs' argument, we need

# specify the set of numbers inside the function 'c()'.

# 'c()' is a function in R which joins together a set

# of numbers and makes them into a vector.

# Vectors are particular data type, but in essence

# they are set of numbers, like row or column.

# The thing to note is just that some functions

# and some arguments in R will need numbers passed to

# them is special ways. Some will need them passed

# in 'c()', others will need them passed in double

# inverted commas "", and some other formats. The main

# thing at this stage is to know to look out for this,

# as they are the sort of typo that will cause bugs

# in your code, and frustrate you for hours as you try

# to work out why your code isn't working.

# VARIANCE AND STANDARD DEVIATION

# Variance and standard deviation are calculated

# with very simple commands: var() and sd()

var(elect\_2013$likelihood\_vote, na.rm = TRUE)

# [1] 1.147758

sd(elect\_2013$likelihood\_vote, na.rm = TRUE)

# [1] 1.071335

# Let's now compare out results with those of McAllister

# 2016. If you haven't already, download the article

# from here: https://doi.org/10.1080/13676261.2016.1154936

#

# Once you have downloaded it go and look at Appendix 1

# and look at the mean of the first variable

# 'Likelihood of voting' and compare the mean reported

# with the mean we calculated.

#

# You will notice that the mean which they report (4.20)

# is different to ours (4.362201).

#

# Why do you think this is?

#

# One reason is that McAllister 2016 reports 'weighted

# means'. What does that mean? It means that the cases

# (i.e. the respondants in the survey) are given

# different weightings when calcuating statistics.

# Why does he do this? Because the survey aims to

# be representative of the Australian population

# but has the problem that the people who did the survey

# are systematically different from the Australian

# population? How? We don't know exactly, but generally

# there tends to be overrepresentation of educated

# and older persons in surveys.

#

# Regardless, McAllister and the people who collected

# this data have calculated a number that represents

# how much we should weight each person in this survey.

# People who from over-represented demographics will

# have a value for the variable 'weight' below 1, e.g. 0.6

# While those who are from under-represented demographics

# will have higher weights (up to 6).

#

# We are lucky that many of the statistical functions

# in R have a way to incorporate weighting into

# their calculations.

#

# For mean, there is a special command

# 'weighted.mean'. Note that we use three arguments:

# \* the variable we are calculating the mean on

# \* the weighting variable

# \* the command to remove cases with missing values.

weighted.mean(elect\_2013$likelihood\_vote,

elect\_2013$weight,

na.rm = TRUE)

# What weighted mean do you get? Is it similar to

# that reported in McAllister, Appendix 1 (he reports 4.20).

##########################################

# LESSON 2: BASIC UNIVARIATE VISUALISATION

##########################################

# This section we are going to learn one

# simple univariate visualisation:

# the histogram.

##########################################

# A histogram is one of the most basic and fundamental

# graphs in statistics. It graphs only one variable.

# The x-axis of a histogram is the various values of

# the variable, while the y-axis is the count of cases

# (units of analysis) that have each value of the variable.

#

# Variables displayed in histograms must be discrete (i.e.

# they need to have a limited number of values, eg. 1,2,3,...).

# They can't be perfectly continuous (e.g. 1.00123, 1.3233, etc).

#

# One of the ways which we deal with the need for discrete

# variables in a histogram is that we often create 'bins'.

# Bins collect cases (units of analysis) who all have a

# value for the variable that is in a particular range.

#

# We actually use 'bins' in lots of different statistical

# applications. For example, say we have a survey question

# where respondents have given their age (between 0 and 120).

# Often we transform that into a smaller number of variables

# such as 0-19, 20-39, 40-59, etc. Each of these categories,

# e.g. 0-19, is a 'bin'.

#

# R has a built in histogram command (called 'hist()'), but when I have been

# testing it, and reading about it, I've found that there

# is some problems that make it unreliable, so I'm going to

# teach another simple method.

#

# We are going to use two commands: 'table()' and 'barplot()'.

#

# - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# Histogram Example 1: Likelihood of voting

# - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# First, let's see what table() does

table(elect\_2013$likelihood\_vote)

# You should get this result in the console:

# 1 2 3 4 5

# 126 269 236 721 2574

#

# So of the 3955 respondents to our survey, 126 said that

# they would definitely not vote if voting was voluntary,

# while 2574 said they definitely would vote.

#

# We can visualise this with the the 'barplot()' function:

barplot(table(elect\_2013$likelihood\_vote))

# in the 'Plots' windown on the bottom right corner of

# RStudio, you will see these five numbers graphed

# as a histogram.

#

# Often we want to graph a histogram as a probability

# density graph, where the y-axis is the proportion of

# cases in each bin. To do this we need to confirm

# the number of cases (removing those that are missing)

# The number of cases can be calculated with the following

# code:

cases <- length(elect\_2013$likelihood\_vote

[!is.na(elect\_2013$likelihood\_vote)])

# To see how many cases there are, just type 'cases'

# and view the contents of the cases variable.

cases

# There are 3926 cases (from a total of 3955 people who

# did the survey).

#

# To display the histogram as a probablity density, we just

# run:

barplot(table(elect\_2013$likelihood\_vote)/cases)

# - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# Histogram Example 2: Political knowledge

# - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# We are now going to move on to a few slighly more

# complex examples.

#

# Let's start by visualising the histogram for

# the 'political knowledge' variable (from 0 to 10), which

# represents the number of Australian politics quiz

# questions the survey respondent got right.

barplot(table(elect\_2013$pol\_knowledge))

# We can see that there is pretty even distribution of

# respondents across the levels of answers, but only

# a very small number (around 130) got 10/10 for the quiz.

#

# Imagine that for some reason we actually wanted to only

# have five bins, 0-1, 2-3, 4-5, etc.

#

# How do we do this?

#

# First we create a variable called 'bins', using one of two

# commands: either 'seq()' or 'c()'

#

# seq() allows us to specify the min and max, and then the

# width of bins. The following commands creates bins between

# 0 and 10, with a width of 2:

bins <- seq(0, 10, by=2)

# You can look at the variable bins by calling it.

bins

# [1] 0 2 4 6 8 10

# You could also make the bins with c(). With c() you just

# specify the exact bins

bins <- c(0,2,4,6,8,10)

# We then go through two steps to make a histogram based

# on these bins. We first use 'cut()' to create a new

# variable called 'x'. This creates a variable, x, which

# the values are simply the bins (so a person who had

# got one quiz question right, and had a 1 for

# 'political knowledge' would have that '1' replaced with

# the name of the 'bin', in this case "(0,2]")

# We then plot x as done previously with barplot() and table()

x <- cut(elect\_2013$pol\_knowledge, breaks=bins)

barplot(table(x))

# - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# Histogram Example 3: Age

# - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# We will do one quick last example with age.

# Let's look at the age histogram, with bins just one year

# wide:

barplot(table(elect\_2013$age))

# Let's now look at it, with the variable divided into

# bins of width 5 years.

bins <- seq(0, 110, by=5)

x <- cut(elect\_2013$age, breaks=bins)

barplot(table(x))

# And as a probablity density graph:

cases <- length(elect\_2013$age

[!is.na(elect\_2013$age)])

barplot(table(x)/cases)

# There is lots more you can do with 'barplot()'. A simple

# extension is to give the graph colour.

barplot(table(x)/cases, col="Red")

# There are also lots of more powerful and beautiful graphs

# that can be made in R. Later in semester Young will teach

# you some of these techniques. If you want to teach yourself

# one place to start for graphing univariate statistics

# with the package ggplot2 is here:

# http://www.sthda.com/english/articles/32-r-graphics-essentials/133-plot-one-variable-frequency-graph-density-distribution-and-more/#density-plots

########################################################

# LESSON 3: UNIVARIATE STATISTICS WITH SUMMARYTOOLS

########################################################

# summarytools is a powerful package that allows

# users to quickly and easily generate tables that

# can be cut and pasted directly into papers,

# presentations, and/or codebooks.

########################################################

# RESOURCES

# Excellent introduction by the author:

# https://cran.r-project.org/web/packages/summarytools/vignettes/Introduction.html

# The full manual:

# https://cran.r-project.org/web/packages/summarytools/summarytools.pdf

########################################################

# WHY USE IT? Because it quickly and easily

# generates beautiful tables that you will need

# for almost every paper, presentation, or codebook

# that you write.

########################################################

# CORE FUNCTIONS: freq(), ctable(), descr(), dfSummary().

########################################################

# TIPS:

# \* for beginners it is almost always easier to use

# html output, rather than rmarkdown.

# \* when writing code, send files to browser so you

# can see your output immediately

# \* when you are ready to publish, you can (1) screenshot;

# (2) cut and paste the tables into excel or word; or

# (3) save as a html file, using the 'file=' argument

# in the print() function.

#########################################################

install.packages("summarytools") # install the package

# (once, and then # it out)

library(summarytools) # load the library

####################################

# LESSON 3.1: freq() function

####################################

# This function gives the count

# and proportions of each value

# of a variable. Note that it

# only takes a single variable

# as input (not a whole dataframe).

####################################

# 1. DON'T DO THIS

# you almost NEVER want to run 'summarytools' functions

# WITHOUT putting them inside the 'print()'

# function and sending them to the browser.

# To understand why, run the next line and look

# at the ugly output sent to the console.

freq(elect\_2013$pol\_knowledge)

# OK, it's not terrible, but you wouldn't want to paste

# that straight into an article or presentation.

#

# 2. BASIC COMMAND

# Now let's run the same command, but we will put it

# inside a print() function, and send it to browser.

print(freq(elect\_2013$pol\_knowledge), method = "browser")

# A table should have openned in your browser (such as Chrome

# Safari, Internet Explorer, Firefox, or Edge.

#

# Notice how it has a much more attractive layout.

# 3. SETTINGS

# There are a few different settings we can use to make

# this table prettier.

# We can omit the headings

print(freq(elect\_2013$pol\_knowledge,

omit.headings = TRUE),

method = "browser")

# We can omit the totals

print(freq(elect\_2013$pol\_knowledge,

totals = FALSE),

method = "browser")

# We can omit the reporting of NAs (missing)

print(freq(elect\_2013$pol\_knowledge,

report.nas = FALSE),

method = "browser")

# We can remove the footnote

print(freq(elect\_2013$pol\_knowledge,

report.nas = FALSE),

method = "browser", footnote = NA)

# 4. PUTTING IT ALL TOGETHER

# And we can put all that together

print(freq(elect\_2013$pol\_knowledge,

omit.headings = TRUE,

totals = FALSE,

report.nas = FALSE),

method = "browser", footnote = NA)

# 5. SAVE TO FILE

# If we want to save this to a file, then we use

# the same command, but replace the 'method ='

# argument with a 'file =' argument, as below:

print(freq(elect\_2013$pol\_knowledge,

omit.headings = TRUE,

totals = FALSE,

report.nas = FALSE),

file = "pol\_know\_freq.html", footnote = NA)

# In the console window you will see

# > Output file written: pol\_know\_freq.html

# and if you go to your default folder (set at the

# beginning of this session), then you will find the

# file 'pol\_know\_freq.html'. If you double click on it

# then it will open in a browser.

####################################

# LESSON 3.2: descr() function

####################################

# This function calculates a wide

# range basic univariate statistics

# such as mean, standard deviation,

# min, max, skewness, etc.

# This function can calcuate these

# on all variables in a dataset

# meaning it can be used to summarise

# an entire dataset very quickly.

# This command is great for doing

# a descriptive statistics table

# - a table which is expected in

# almost all academic papers.

####################################

# This is the simplest form of the command

print(descr(elect\_2013$pol\_knowledge), method = "browser")

# However, when reporting variables, we often like them

# to be presented as the rows, not columns. To change

# this we used the argument 'transpose'

print(descr(elect\_2013$pol\_knowledge, transpose = TRUE),

method = "browser")

# Generally we don't want all these statistics, so we

# can limit the statistics reported with the argument

# 'stats'.

print(descr(elect\_2013$pol\_knowledge,

stats = c("mean", "sd", "min", "max"),

transpose = TRUE),

method = "browser")

# Note that when using the stats argument, we put

# the names of the stats we want using the

# "c()" function. If you want to know the commands

# for each of the different stats, then look at

# the help file (type ?descr)

#

# However, remember that the data from the Australian

# Electoral Study needs to be weighted to account

# for the difference between the sample and the population.

# We can do this with the argument 'weights'

print(descr(elect\_2013$pol\_knowledge,

stats = c("mean", "sd", "min", "max"),

transpose = TRUE,

weights = elect\_2013$weight),

method = "browser")

# The real power of descr() is that it can calculate these

# statistics for all variables in a dataset. We can do this

# by just calling descr() on the data frame.

# Note: I've also removed the headings and footnote.

print(descr(elect\_2013,

omit.headings = TRUE,

stats = c("mean", "sd", "min", "max"),

transpose = TRUE,

weights = elect\_2013$weight),

method = "browser", footnote = NA)

####################################

# LESSON 3.3: dfSummary() function

####################################

# While the 'descr()' function is

# good for making publication quality

# tables, the dfSummary() function

# is more for the private use of

# the data analyst.

# dfSummary() is the kind of function

# you use to 'get a quick feel' for

# your data. It lets you quickly

# see the number of missing values

# and a rough histogram of the values

# and also see the percent of cases

# of each value. This can be very

# useful when you are trying to assess

# the quality of data, or looking for

# interesting patterns you may want

# to explore later.

####################################

# The following command gives a summary of just

# one variable:

print(dfSummary(elect\_2013$pol\_knowledge), method="browser")

# The next command gives a summary of the entire

# dataset:

print(dfSummary(elect\_2013), method="browser")

# This is the end of the Univariate statistics

# R code/ R-script file. The class continues

# in the code "Week 2, Part 2, Bivariate statistics.R'