308A R Practice Assignment

Brady C. Jackson

2024/09/29

Assignment goals and statement:

For this assignment, you will be practicing the skills you learned in the workshop that are important for completing analyses in the remainder of the 308 sequence. Use the dataset labeled '308A.R Practice Assignment.HomeworkData' from canvas. Write all code below the following tasks:

- begin every chunk with a comment explaining what that chunk is going to do (these are notes for you, so make them something you will understand looking at it later)
- load the following libraries: psych, jmv, summarytools, dplyr

```
# We start our code by loading the necessary dependencies into our environment.
# This is expected to generate masking warnings as some functions in these
# libraries will overload the standard packages versions of these functions.
library('psych')
library('jmv')
```

```
##
## Attaching package: 'jmv'
```

```
## The following objects are masked from 'package:psych':
##
##     pca, reliability
```

```
library('summarytools')
library('dplyr')
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
# Loading some additional libraries so I can be a bit extra in this assignment:
renv::install("cli", prompt=FALSE)
```

```
## The following package(s) will be installed:
## - cli [3.6.3]
## These packages will be installed into "~/programming/r/psyd/cgu/coursework/psych_308/308A/renv/libra
##
```

```
## # Installing packages --------------------------------------------------------
## - Installing cli ...                            OK [copied from cache]
## Successfully installed 1 package in 51 milliseconds.
```

```
library('cli')
```

- find the library that contains the function 'pivot_wider', install and load the appropriate library
  - tidyr Refs: https://www.rdocumentation.org/packages/tidyr/versions/1.3.1/topics/pivot_wider

```
# The pivot_wider function is part of the tidyr package. Refs:
#  https://www.rdocumentation.org/packages/tidyr/versions/1.3.1/topics/pivot_wider
# NOTE: This package is already installed in your R environment for this project,
# but code is included here for assignment credit.
#
# NOTE: I'm using renv::install rather than install.packages deliberatley as I
#       develop my R-code in an R environment using the renv package.
#       This is because I use R professionally on other projects and I need
#       to avoid dependency collisions between projects. So I can't have one
#       library for my system that I use for every project. That said, I promise,
#       renv::install does the same thing as install.packages() in the context
#       of this projec (308A).
renv::install('tidyr', prompt=FALSE)
```

```
## The following package(s) will be installed:
## - tidyr [1.3.1]
## These packages will be installed into "~/programming/r/psyd/cgu/coursework/psych_308/308A/renv/libra
##
## # Installing packages --------------------------------------------------------
## - Installing tidyr ...                          OK [copied from cache]
## Successfully installed 1 package in 39 milliseconds.
```

```
library('tidyr')
```

- read in the data, name it dat2

```
# The data for this HW project is stored in a file collocated with this Rmd file
# named: 308A.R Practice Assignment.HomeworkData.csv
# Since it's collocated we just load from the current working directory with
# "./" but for good measure we reconstruct the filename, full path, and extension
# once we load the file in case we need it again later.

# Building up the filename bits and pieces
this_dir = "."
here = getwd()
dat_filename = "308A.R Practice Assignment.HomeworkData"
dat_ext = ".csv"
dat_filename_full = paste(dat_filename, dat_ext, sep = '')

# Note we use .Platform$file.sep here so that this code will run regardless of
# OS it's executed on. Since I'm developing in WSL this should still enable
# it to run in Windows despite being dev'ed in a linux-like environment
fullfile = paste(here, dat_filename_full, sep=.Platform$file.sep)
local_file = paste(this_dir, dat_filename_full, sep=.Platform$file.sep)

# Before we load the file, let's check if it exists and print an error if
# there's a problem.
if( !(file.exists(local_file)) ){
```

```
    err_msg = paste(
        "The following file does not exist at the location specified: \n",
        "    ", local_file, " \n",
        "Please check your filename and path inputs and try again. \n",
        sep = ""
    )
    cat(col_red(err_msg))
} else {
    # If we threw no error load the file and print a success message
    dat2 = read.csv(local_file, header=TRUE)

    succ_msg = paste(
        "The following file loaded successfully: \n",
        "    ", local_file, " \n",
        "Congratulations! \n",
        sep = ""
    )
    cat( col_green( succ_msg) )
}
```

```
## The following file loaded successfully:
##     ./308A.R Practice Assignment.HomeworkData.csv
## Congratulations!
```

- check the data type of the columns in the data set

```
# We can use the class function to check the variable type of data stored in any
# given column in dat2. First we need to know how many columns there are so we
# can loop through them and print a report of the type of each column.
cnames = colnames(dat2)
ncols = length(cnames)

# We're going to define an empty string that we can concatenate output text onto
# as we loop through each column. This isn't super memory efficient as the
# space for the character string has to get dynamically allocated with each pass
# of the loop, but oh well.
out_str = ''

# We could loop through each column name directly but then we'd lose control
# of any index into the cnames vector so we'll loop through an appropriately
# sized index instead
for(iii in c(1:ncols)){
    # Unpack our column names data. Note that we store the data itself from the
    # column in this_col
    this_cname = cnames[iii]
    this_col = dat2[, this_cname]
    this_type = class( this_col )

    # Now we can format the report string for this column and append it to our
    # output string. NOTE: Color formatting only works for command line (CLI)
    # output. Need to add a dedicated function to use span or latex wrappers
    # for colored output in knit output files
    this_str = paste(
        "The data stored in the dat2 column named: ", col_blue(this_cname), " \n",
        "...is of datatype: ", col_blue(this_type), "\n\n",
```

```
        sep = ""
    )

    out_str = paste(out_str, this_str, sep="")
}

# Print the output string using "cat" instead of "print" so newline characters
# are appropriately handled
cat(out_str)
```

```
## The data stored in the dat2 column named: Pnum
## ...is of datatype: integer
##
## The data stored in the dat2 column named: Stress
## ...is of datatype: numeric
##
## The data stored in the dat2 column named: Test
## ...is of datatype: numeric
##
## The data stored in the dat2 column named: Gender
## ...is of datatype: integer
```

- run descriptives for the variable 'Test' by using the following code:
- an error has occured. Interpret and fix the error from the code above.

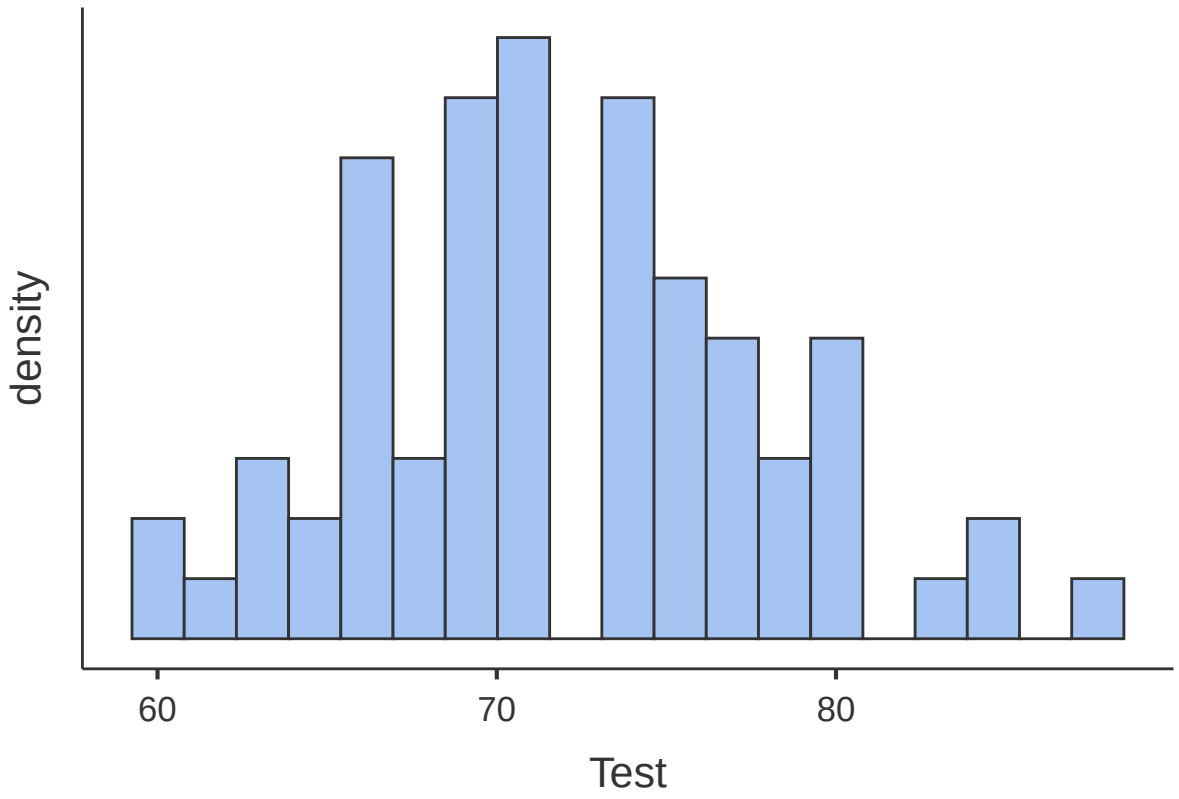
```
desc <- descriptives(
  data =dat,
  vars = c(
    'Test'),
  hist = TRUE,
  sd = TRUE,
  se = TRUE,
  skew = TRUE,
  kurt = TRUE)
```

```
# This code is going to fail because the data input is misnamed (uses 'dat'
# instead of 'dat2')
#
# Wrap it in an error catch flag set manually so you can toggle between throwing
# the error and running the fixed code.
run_bugged = 0
run_fixed = !(run_bugged)

# This switch runs the bugged code. To run it and generate the error code set
# "run_bugged" above to 1
if(run_bugged){
  desc <- descriptives(
      data = dat,
      vars = c('Test'),
      hist = TRUE,
      sd = TRUE,
      se = TRUE,
      skew = TRUE,
      kurt = TRUE
```

```
  )

}

# Fix for second part of question is contained here.
if(run_fixed){
  desc <- descriptives(
      data =dat2,
      vars = c('Test'),
      hist = TRUE,
      sd = TRUE,
      se = TRUE,
      skew = TRUE,
      kurt = TRUE
  )

  out_str = paste(
      "Congratulations! You ran your fixed code correctly. Revel in your\n",
      "glorious output! \n\n"
  )
  cat(col_green(out_str))

  desc

}
```

```
## Congratulations! You ran your fixed code correctly. Revel in your
##  glorious output!
```

```
##
##  DESCRIPTIVES
##
##  Descriptives
##
##                            Test
##
##    N                             70
##    Missing                        0
##    Mean                    72.01857
##    Std. error mean         0.7229148
##    Median                  71.20000
##    Standard deviation      6.048339
##    Minimum                 59.90000
##    Maximum                 87.60000
##    Skewness                0.2558200
##    Std. error skewness     0.2867505
##    Kurtosis               -0.2059856
##    Std. error kurtosis     0.5662651
##
```



- make the variable 'Gender' a factor

```
# Print the class type of the Gender column before the conversion and after the
# conversion as proof.
cat("The original datatype of the 'Gender' column is:", col_blue(class(dat2$Gender)), '\n' )
```

```
## The original datatype of the 'Gender' column is: integer
```

```
# Now do the Gender swap. Recall 'factor' is strictly-allowed-values array datatype
dat2$Gender <- as.factor(dat2$Gender)

# Print the new Gender type
cat("The new datatype of the 'Gender' column is:", col_blue(class(dat2$Gender)), '\n\n' )
```

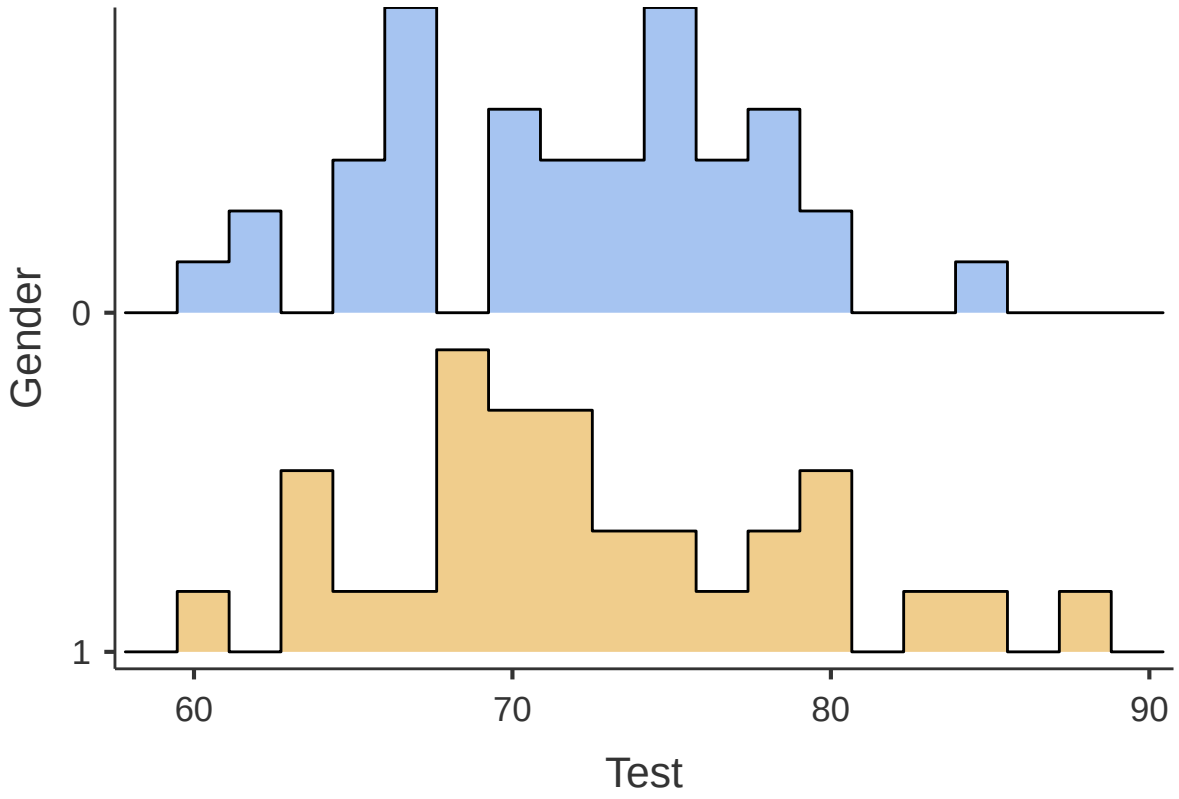
```
## The new datatype of the 'Gender' column is: factor
```

- run descriptives for the variable 'Test,' splitting the participants into groups based on 'Gender'

```
# Nothing really fancy to do here. The splitBy argument in the descriptives function
# does all the heavy lifting here.
#
# What this is doing is creating an object, test_desc, using the descriptives
# function from the jmv package that first sorts the data based on the
# value of gender (only two genders are modeled), and then describes the
# data associated with the "Test" column as separated by those two Gender values
#
# Note, I include the package name in the descriptives call to help future-Brady
# and others reading this code understand where the descriptives function is loaded
# from
test_desc <- jmv::descriptives(
    dat2,
```

```
    vars = c('Test'),
    splitBy = 'Gender',
    hist = TRUE,
    sd = TRUE,
    se = TRUE,
    skew = TRUE,
    kurt = TRUE
)

# Print the new descriptives object / dump it to output
test_desc
```

```
##
##  DESCRIPTIVES
##
##  Descriptives
##
##                            Gender    Test
##
##    N                       0                 38
##                            1                 32
##    Missing                 0                  0
##                            1                  0
##    Mean                    0            71.75526
##                            1            72.33125
##    Std. error mean         0           0.9330710
##                            1            1.142264
##    Median                  0            72.35000
##                            1            71.00000
##    Standard deviation      0            5.751836
##                            1            6.461621
##    Minimum                 0            60.00000
##                            1            59.90000
##    Maximum                 0            85.30000
##                            1            87.60000
##    Skewness                0         -0.009744702
##                            1            0.4606167
##    Std. error skewness     0            0.3828184
##                            1            0.4144573
##    Kurtosis                0           -0.4792797
##                            1          -0.02659626
##    Std. error kurtosis     0            0.7497004
##                            1            0.8093713
##
```



- Use pipes to rename the variable 'Test' to 'Score' and then create a new variable by multiplying 'Score' and 'Stress', call the new variable 'TestxStress'

```
# Would've been better to do this at the top with the other library
# loads but we need magrittr for fancy-piping
library(magrittr)
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
# Before we try to force a rename in-place on an existing object, we'll
# check if the column we want to rename is actually in the data. If it is,
# proceed with the rename. Assuming we've run all chunks above, the column
# names should still be named in cnames:
if( any( grepl('Test', cnames) ) ){
    dat2 %<>% rename(Score = Test)
    cat(paste(colnames(dat2), collapse = "\n") )
    cat("\n\n")
}
```

```
## Pnum
## Stress
## Score
## Gender
```

```
# Finally, let's create a column for the product of the newly named
# "Score" column and "Stress". This time we save the new column
```

```r
# names as new_cnames for potential future use.
dat2$TestxStress <- dat2$Score * dat2$Stress
new_cnames = colnames(dat2)
cat(paste(new_cnames, collapse = "\n"))
```

```
## Pnum
## Stress
## Score
## Gender
## TestxStress
```

```r
cat("\n\n")
```

- enter, and run the following code into its own chunk: dat <- rename(dat2, Group = gender)
- interpret and fix the error from the code given above

```r
# The code below is copied from the assignment requirements and will throw an
# error. Wrap it in a "run_bugged_2" flag so we can turn it off
# to run the whole Rmd file later
run_bugged_2 = 0   # set to 0 to run the fixed code. Set to 1 to throw the error
run_fixed_2 = !(run_bugged_2)
if(run_bugged_2){
    dat <- rename(dat2, Group = gender)
}

# run_fixed_2 is NOT run_bugged_2 so it runs the fixed code when we don't
# run the buggy code
if(run_fixed_2){
    # The bug in the original code was the it was looking for 'gender' with
    # a lowercase spelling but our column names still have the first letter
    # capitalized. Code below fixes this.
    dat <- rename(dat2, Group = Gender)

    # Print the colnames to prove it got renamed
    cat(paste(colnames(dat), collapse = "\n") )
    cat("\n\n")
}
```

```
## Pnum
## Stress
## Score
## Group
## TestxStress
```

- knit the markdown file to word or pdf and submit on canvas

Aye, Aye, Captain!