Analytics Programming Assessment

Ryan Greenup - 17805315

Contents

Preamble

Load Packages

```
## (01) Clean up the Iris Data

# Preamble
## Install Pacman
load.pac <- function() {

  if(require("pacman")){
    library(pacman)
  }else{
    install.packages("pacman")
    library(pacman)
  }
```

```
  pacman::p_load(xts, sp, gstat, ggplot2, rmarkdown, reshape2, ggmap,
                 parallel, dplyr, plotly, tidyverse, reticulate, UsingR, Rmpfr,
                 swirl, corrplot, gridExtra, mise, latex2exp, tree, rpart, knitr,
                 bookdown, tidyverse, tidyr)

}

load.pac()
```

```
## Loading required package: pacman
```

Knitr Configuration

Make Chunks verbose

```
knitr::opts_chunk$set(echo = TRUE, eval = TRUE)
```

Set Figure Locations

```
knitr::opts_chunk$set(
  fig.path = "figure/"
)
```

Question 1

d. Submission Receipt

The submission receipt for the preliminary assessment is provided as a screenshot at figure 1 and the recipt number is:

```
    95d652b5-e2e7-4f43-b7e5-3290b9e4ef0b
```
```
include_graphics("./Receipt.png")
```

e. Set Variables

```
studentname <- "Ryan Greenup"
studentno   <- 17805315
```

f. Print Variables

i. Show the Code Without the Results

```
print(studentname)
print(studentno)
```

Success! Your submission appears on this page. The submission confirmation number is 95d652b5-e2e7-4f43-b7e5-3290b9e4ef0b. Copy and save this number as proof of your submission. View all of your submission receipts in My Grades.

Review Submission History: Assignment Preliminary

Assignment Instructions ⌄

box

Question 1

Ryan Greenup

26/05/2020

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed          dist
## Min.   : 4.0   Min.   :  2.00
## 1st Qu.:12.0   1st Qu.: 26.00
## Median :15.0   Median : 36.00
## Mean   :15.4   Mean   : 42.98
## 3rd Qu.:19.0   3rd Qu.: 56.00
## Max.   :25.0   Max.   :120.00
```

Including Plots

You can also embed plots, for example:

Figure 1: Receipt from preliminary submission

ii. Show the Evaluated Output in the Document

```
## [1] "Ryan Greenup"
```

```
## [1] 17805315
```

ii. Show the Evaluated Output and corresponding code in the Document

```r
print(studentname)
```

```
## [1] "Ryan Greenup"
```

```r
print(studentno)
```

```
## [1] 17805315
```

Question 2

1. Write a function that takes a vector of numbers as its input data and randomly picks 10% of thenumbers, then writes that subsample into a new vector

the `sample` function can be used to take a sample from a vector:

```r
subsample <- function(vec) {
  n <- length(vec)
  sample(vec, size = 0.1*n, replace = FALSE)
                        ## For a bootstrap replace should be TRUE
}
```

2. Test your function by creating a vector of 1000 random samples from a normal distribution withmean 1 and standard deviation 2, and then sampling 100 numbers from it.

The `rnorm` function can be used to generate normally distributed values and passing this to the previously defined `subsample` function will extract a random sample of 10% of the values:

```r
norm_values <- rnorm(n = 1000, mean = 1, sd = 2)
(norm_ss <- subsample(norm_values))
```

```
##    [1]  0.201644034  1.010102218  1.552736209 -2.004674126  0.913091009
##    [6] -0.722039227  1.543477707 -0.469466584  3.962285914 -0.969641803
##   [11] -1.770956862  1.091770844  1.200408627 -1.613127697  1.825270918
##   [16] -0.004877450  1.367463627 -1.800776765  0.789464362  3.193450612
##   [21]  2.714239972  2.140412908  0.099039638 -0.541974411  0.008204049
##   [26]  0.082625963  2.447308691  4.178895276  1.361066171  6.567977805
##   [31] -0.740830187  2.667036418  0.258344541  0.868426100  0.903986868
##   [36]  0.429331815  1.921341130 -2.512019316  1.620617246  3.012727811
##   [41] -1.417300458  3.284462040 -1.824781739  5.633232740  2.655965182
##   [46]  0.714705767  3.640892749  0.336724759  2.017815369  1.362896816
##   [51]  1.527897235  0.198000225 -1.137644927  2.239314584  0.537668316
##   [56] -3.116351913 -1.332987850 -3.147590523  2.099663277 -0.346118849
##   [61]  2.246064930  3.135852953  0.939113595 -1.650537972  2.939958694
```

```
## [66]  1.848738443  2.511276128 -3.135827683 -2.682225675  1.754897678
## [71] -1.825083131 -1.683479951  1.957437679  1.554577441 -1.188585643
## [76]  3.126929818  1.168646809  3.231676138  1.830996084  1.351080603
## [81] -0.796662113  0.750308125  0.843884019  1.522282547 -1.943944808
## [86]  4.078400025 -3.847186157  0.188499049  2.487532896 -0.443725827
## [91]  4.302740661 -2.683897091 -0.332989630  1.497678622  3.835935366
## [96]  3.354749505  1.724029631  2.300562429  3.550241751  2.286705906
```

3. Output the mean and standard deviation of the 100-number subsample

```
rbind(c(
  "Mean"   = mean(norm_ss),
  "StdDev" = sd(norm_ss))
  )
```

```
##           Mean    StdDev
## [1,] 0.9081548 2.055567
```

4. Call the function 100 times to get 100 different 100-number subsamples of the same 1000-numbervector as used above, calculate their means and standard deviations, and then plot histograms of themeans and standard deviations of the subsamples.

First resample the vector of values 100 times and produce a matrix using `replicate`[1]:

```
## Resample the Values
boot_vals <- replicate(100, {
  subsample(norm_values)
})
```

Each Column of the output matrix corresponds to a seperate sample, so in order to get the mean and sd of each sample the `apply` function can be used, specifying the `MARGIN` argument as `2` in order to denote function operating column-wise.

```
## Collumns correspond to the sample number and the rows to obsevations
##
## Calculate the mean values
sample_means <- apply(boot_vals, 2, mean)

## Calculate the STD Dev
sample_sd    <- apply(boot_vals, 2, sd)
```

In order to plot the histograms it is first necessary to manipulate the data such that it is in a `tidy` format, this can be done by using the `tidyr` package, then the histograms can be plotted using `ggplot2`.

```
data <- data.frame(means = sample_means, sds = sample_sd)
tidy_data <- tidyr::pivot_longer(cols = c(means, sds), data = data, names_to = "Parameter")
tidy_data$Parameter <- factor(tidy_data$Parameter, labels = c("Mean", "SD"), ordered = FALSE)
tidy_data
```

```
## # A tibble: 200 x 2
##    Parameter value
```

---

[1]`replicate` is a wrapper for `sapply` and is usually quicker than using a loop, loops using a dynamic vector are often slow, `replicate` has the added benefit that it is not necessary to define a static vector either.

```
##    <fct>       <dbl>
## 1 Mean        0.899
## 2 SD          1.93
## 3 Mean        1.04
## 4 SD          2.17
## 5 Mean        0.837
## 6 SD          2.20
## 7 Mean        1.09
## 8 SD          2.13
## 9 Mean        0.823
## 10 SD         2.19
## # ... with 190 more rows
```

```
ggplot(tidy_data, aes(x = value, fill = Parameter, y = ..density..)) +
  geom_histogram(col = "black", binwidth = 0.1) +
  facet_grid(. ~ Parameter, scales = "free_x") +
  theme_bw()  +
  labs(x = "Value", y = "Density",
       title = TeX('Bootstrapped Descriptive statistics from $X \\sim \\textit{N} (2,1)$'))
```


Bootstrapped Descriptive statistics from X ~ *N*(2,1)
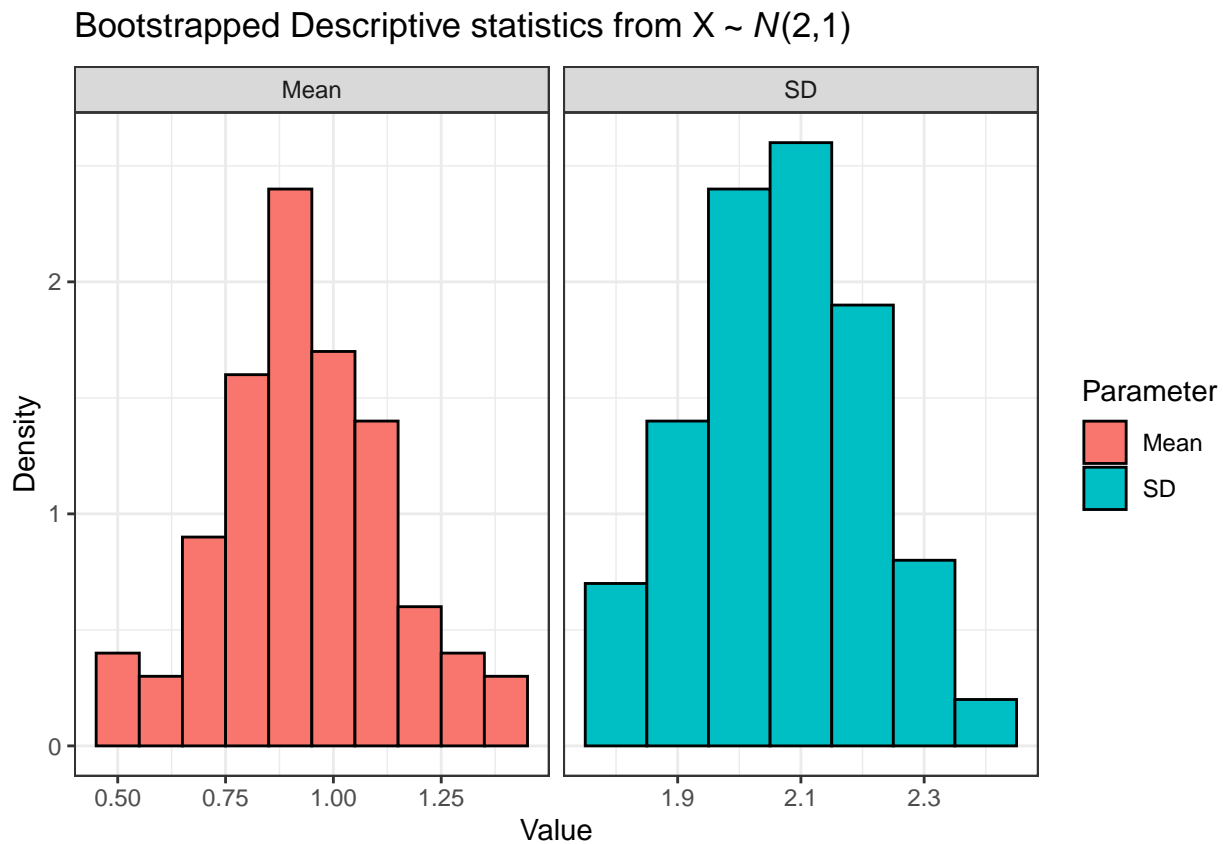
```
##  stat_function(fun = dnorm, args = list(mean = mean(data$means),
##                                         sd = sd(data$means)))
```

Question 3

1. Display the image in a plot.

2. Plot only the red colour plane.

The JPEG image comprises separate images (or "planes") in the three primary colours, Red, Greenand Blue. Plot only the Red colour plane. ## 3. Plot a histogram of the Red values of all the pixels in the image. ## 4. Plot the red, green and blue pixel values of all the pixels in a horizontal line across the middle of theimage (i.e. all pixels with a vertical index of 256). # Question 4 ## a.Imimtate a sequence of Dice Rolls Try to imitate a sequence of dice rolls. Create a vector of numbers (1, 2, 3, 4, 5, 6), of length 51. Beas random as you can. Do this before you read the rest of the question. (Or, get someone else to doit.) Print your sequence.My fake sequence: 1326545442136512455235112516565452221615243662154433 ## b. Devise a test to real dice-rolling, Can we devise a test to distinguish between the fake dice-rolling and real dice-rolling, i.e. a trulyrandom process? One method is based on the observation that people tend to put fewer runs ofnumbers in their sequences than you find in random sequences. Write code to count the number ofruns of 1 (i.e. each number that is not followed by the same number), 2 (i.e. 2 consecutive numbersthe same), 3 and 4. ## c.How many runs of each length would we get if the data were truly random? How many runs of each length would we get if the data were truly random? Write a loop to simulate105sets of 51 tosses, storing the number of runs of each length (1, 2, 3, 4). Plot the histogram of runs(i.e. total number of runs of each length added up over al