IST 687 HW 03

Darrell Nelson II

January 31, 2019

```
# HW 03
# Darrell Nelson II

# Step 1: Create a function (named readStates) to read a CSV file into R
readStates <- read.csv(url("http://www2.census.gov/programs-surveys/popest/tables/2010-2011/state/totals

# Step 2: Clean the dataframe
readStates <- readStates[-1:-8,] # Remove the table description section and all cardinal direction rela
readStates <- readStates[,-6:-10] # Removes the empty columns at the end of the dataset
readStates <- readStates[-52:-58,] # Removes the footnotes & citations at the bottom of the dataset
newcolnames <- colnames(readStates) # Places all column headers in a new object
newcolnames[1] <- "stateName" # Naming first column in new object; Don't use spaces in new column name
newcolnames[2:5] <- c("base2010", "base2011", "Jul2010", "Jul2011") # Naming columns 2 through 5
colnames(readStates) <- newcolnames #Place new column headers on readStates dataset
readStates$stateName <- gsub("\\.", "", readStates$stateName) #removes "." from stateName column
# Remove commas from columns [,2:5]
readStates$base2010 <- gsub(",", "", readStates$base2010)
readStates$base2011 <- gsub(",", "", readStates$base2011)
readStates$Jul2010 <- gsub(",", "", readStates$Jul2010)
readStates$Jul2011 <- gsub(",", "", readStates$Jul2011)
# Remove all unwanted spaces and convert data type to numeric in columns [,2:5]
readStates$base2010 <- as.numeric(gsub(" ", "", readStates$base2010))
readStates$base2011 <- as.numeric(gsub(" ", "", readStates$base2011))
readStates$Jul2010 <- as.numeric(gsub(" ", "", readStates$Jul2010))
readStates$Jul2011 <- as.numeric(gsub(" ", "", readStates$Jul2011))
# Disregard row numbering from .csv file
rownames(readStates) <- NULL

# Step 3: Store and Explore the dataset
# Storing dataset into a dataframe
dfStates <- readStates

# Calculate mean of July 2011
a1 <- mean(dfStates$Jul2011)
sprintf("The mean population per state in July 2011 is: %.3f", a1)
```

```
## [1] "The mean population per state in July 2011 is: 6109645.431"
```

```
# Step 4: Find the state with the Highest Population
# Based on the July2011 data, what is the population of the state with the highest
# population? What is the name of that state?
coljul2011 <- which(colnames(dfStates) == 'Jul2011')
rowjul2011 <- which.max(dfStates$Jul2011)
maxpopjul2011 <- dfStates[rowjul2011, coljul2011]
sprintf("Highest population in July 2011 is: %s", maxpopjul2011)
```

```
## [1] "Highest population in July 2011 is: 37691912"
```

```
maxpopname <- dfStates$stateName[rowjul2011]
sprintf("State with highest population in July 2011 is: %s", maxpopname)
```

```
## [1] "State with highest population in July 2011 is: California"
```

July 2011 data sorted in increasing order:

```
# Sorting data in increasing order based on July 2011 data
dfStates[order(dfStates$Jul2011) , ]
```

```
##                stateName base2010 base2011  Jul2010  Jul2011
## 51               Wyoming   563626   563626   564554   568158
## 9   District of Columbia   601723   601723   604912   617996
## 46               Vermont   625741   625741   625909   626431
## 35          North Dakota   672591   672591   674629   683932
## 2                 Alaska   710231   710231   714146   722718
## 42          South Dakota   814180   814180   816598   824082
## 8               Delaware   897934   897934   899792   907135
## 27               Montana   989415   989415   990958   998199
## 40          Rhode Island  1052567  1052567  1052528  1051302
## 30         New Hampshire  1316470  1316472  1316807  1318194
## 20                 Maine  1328361  1328361  1327379  1328188
## 12                Hawaii  1360301  1360301  1363359  1374810
## 13                 Idaho  1567582  1567582  1571102  1584985
## 28              Nebraska  1826341  1826341  1830141  1842641
## 49         West Virginia  1852994  1852996  1854368  1855364
## 32            New Mexico  2059179  2059180  2065913  2082224
## 29                Nevada  2700551  2700551  2704283  2723322
## 45                  Utah  2763885  2763885  2775479  2817222
## 17                Kansas  2853118  2853118  2859143  2871238
## 4               Arkansas  2915918  2915921  2921588  2937979
## 25           Mississippi  2967297  2967297  2970072  2978512
## 16                  Iowa  3046355  3046350  3050202  3062309
## 7            Connecticut  3574097  3574097  3575498  3580709
## 37              Oklahoma  3751351  3751354  3760184  3791508
## 38                Oregon  3831074  3831074  3838332  3871859
## 18              Kentucky  4339367  4339362  4347223  4369356
## 19             Louisiana  4533372  4533372  4545343  4574836
## 41        South Carolina  4625364  4625364  4637106  4679230
## 1                Alabama  4779736  4779735  4785401  4802740
## 6               Colorado  5029196  5029196  5047692  5116796
## 24             Minnesota  5303925  5303925  5310658  5344861
## 50             Wisconsin  5686986  5686986  5691659  5711767
## 21              Maryland  5773552  5773552  5785681  5828289
## 26              Missouri  5988927  5988927  5995715  6010688
## 43             Tennessee  6346105  6346110  6357436  6403353
## 3                Arizona  6392017  6392013  6413158  6482505
## 15               Indiana  6483802  6483800  6490622  6516922
## 22         Massachusetts  6547629  6547629  6555466  6587536
## 48            Washington  6724540  6724540  6742950  6830038
## 47              Virginia  8001024  8001030  8023953  8096604
## 31            New Jersey  8791894  8791894  8799593  8821155
## 34       North Carolina  9535483  9535475  9560234  9656401
## 11              Georgia  9687653  9687660  9712157  9815210
## 23             Michigan  9883640  9883635  9877143  9876187
## 36                 Ohio 11536504 11536502 11537968 11544951
## 39         Pennsylvania 12702379 12702379 12717722 12742886
## 14             Illinois 12830632 12830632 12841980 12869257
## 10              Florida 18801310 18801311 18838613 19057542
## 33             New York 19378102 19378104 19395206 19465197
## 44                Texas 25145561 25145561 25253466 25674681
## 5            California 37253956 37253956 37338198 37691912
```

Functions!!

```
# Function wizardry
percentbelow <- function(myVector, mynumber)
{
  sorted <- sort(myVector) # makes sure vector is in ascending order
  Newvector <- sorted[sorted<mynumber] # truncate vector so highest value is less than 'mynumber'
  Mx_index <- which.max(Newvector) # finds highest value in x and stores row index
  Maxnum <- Newvector[Mx_index] # puts value of index into an object
  distribution <- ecdf(sorted) # creates empirical distribution function of original sorted vector
  ans <- distribution(Maxnum) # returns % of numbers in list that are smaller than 'mynumber'
  sprintf("Function01 -> The cumulative distribution below %.3f is: %.3f", mynumber, ans)
}

# Test the function
df <- dfStates$Jul2011 # storing July 2011 census data into new variable
meandf <- mean(df) # storing the mean of July 2011 census data into new variable

# Run the function
percentbelow(df,meandf)
```

```
## [1] "Function01 -> The cumulative distribution below 6109645.431 is: 0.667"
```

```
# Another FUNCTION with WAY less overhead
percentbelow2 <- function(myVector, mynumber)
{
  myVector <- sort(myVector) # puts incoming vector into ascending order
  a <- length(myVector) # calculates the length of the inputted vector
  b <- length(myVector[myVector<mynumber]) # calculates length of truncated vector
  c <- b/a # storing the division of the top two variables into another variable
  sprintf("Function02 -> The cumulative distribution below %.3f is: %.3f", mynumber, c)
}

# Test the function
percentbelow2(df,meandf)
```

```
## [1] "Function02 -> The cumulative distribution below 6109645.431 is: 0.667"
```

Although I'd love to say the first program is better (because I spent so much more time on it), the second one is clearly superior in both overhead and simplicity. I discovered this method while I was trying to test my first function. The second function doesn't deal with the actual numbers in the passed in vector like the first function does. Instead, once the data is sorted: it simply 1) counts the length of the vector, 2) counts the length of the same vector that has been truncated to include all numbers less than the passed in number, and 3) divides those two lengths to reach the proper result. No need to call built-in distribution functions in R!