DanBurkeHW3

Dan Burke

8/2/2021

Dan Burke

IST 687 Homework 3 - Cleaning/munging Dataframes

Assignment Due: 8/3/2021

Submitted: 8/3/2021

Step 1: Create a function (named readStates) to read a CSV file into R

Often, in data science, when you get a dataset, it is not in the exact format you want/need. So, you have to refine the dataset into something more useful - this is often called "data munging". In this lab, you need to read in a dataset and work on that dataset (in a dataframe) so that it can be useful. Then, we will explore the distribution within the dataset. Step 1: Create a function (named readStates) to read a CSV file into R

1. Note that you are to read a URL, not a file local to your computer.

2. The file is a dataset on state populations (within the United States).

The URL is: http://www2.census.gov/programs-surveys/popest/tables/2010-2011/state/totals/nst-est2011-01.csv Hint: google "read.csv" and "url" with respect to R commands

```
readStates <- function(){
  states <- read.csv("http://www2.census.gov/programs-surveys/popest/tables/2010-2011/state/totals/nst-
  return(states)
}
```

Step 2: Clean the dataframe

3. Note the issues that need to be fixed (removing columns, removing rows, changing column names).

4. Within your function, make sure there are 51 rows (one per state + the district of Columbia). Make sure there are only 5 columns with the columns having the following names (stateName, base2010, base2011,Jul2010, Jul2011).

```
readStates <- function(){

  states <- read.csv("http://www2.census.gov/programs-surveys/popest/tables/2010-2011/state/totals/nst-

  #Capture only the States and populations, ignoring the "NA" Values
  states <- states[9:59,1:5]

  #Set Up for Looping through and renaming
  startYear <- 2010
  oldColNames <- colnames(states)
  newColNames <- c("stateName",  "base2010",    "base2011","Jul2010", "Jul2011")

  #Loop through for the length of the Column Names and rename
  for(i in 0:length(colnames(states))){
    names(states)[names(states) == oldColNames[i]] = newColNames[i]
  }

  return(states)
}
```

5. Make sure the last four columns are numbers (i.e. not strings).

```
readStates <- function(){

  states <- read.csv("http://www2.census.gov/programs-surveys/popest/tables/2010-2011/state/totals/nst-

  #Capture only the States and populations, ignoring the "NA" Values
  states <- states[9:59,1:5]

  #Set Up for Looping through and renaming
  startYear <- 2010
  oldColNames <- colnames(states)
  newColNames <- c("stateName",  "base2010",    "base2011","Jul2010", "Jul2011")

  #Loop through for the length of the Column Names and rename
  for(i in 0:length(colnames(states))){
    names(states)[names(states) == oldColNames[i]] = newColNames[i]
  }

  #Also cleaning up the '.' before each state name
states$stateName <- gsub("[:.:]","",states$stateName)

#Converting to numeric
states$base2010 <- as.numeric(gsub(",","",states$base2010))
states$base2011 <- as.numeric(gsub(",","",states$base2011))
states$Jul2010 <- as.numeric(gsub(",","",states$Jul2010))
states$Jul2011 <- as.numeric(gsub(",","",states$Jul2011))

sapply(states, class)


  return(states)
}
```

Step 3: Store and Explore the dataset

6. Store the dataset into a dataframe, called dfStates.

```
dfStates <- data.frame(readStates())
```

7. Test your dataframe by calculating the mean for the July2011 data, by doing:

mean(dfStates$Jul2011) à you should get an answer of 6,109,645

```
mean(dfStates$Jul2011)
```

```
## [1] 6109645
```

Step 4: Find the state with the Highest Population

8. Based on the July2011 data, what is the population of the state with the highest population? What is the name of that state?

```
mostPopState <- dfStates[which.max(dfStates$Jul2011),]
#State Name
mostPopState$stateName
```

```
## [1] "California"
```

```
#State Population
mostPopState$Jul2011
```

```
## [1] 37691912
```

9. Sort the data, in increasing order, based on the July2011 data.

```
dfStates[order(dfStates$Jul2011),]
```

```
##                stateName base2010 base2011  Jul2010  Jul2011
## 59               Wyoming   563626   563626   564554   568158
## 17 District of Columbia   601723   601723   604912   617996
## 54               Vermont   625741   625741   625909   626431
## 43          North Dakota   672591   672591   674629   683932
## 10                Alaska   710231   710231   714146   722718
## 50          South Dakota   814180   814180   816598   824082
## 16              Delaware   897934   897934   899792   907135
## 35               Montana   989415   989415   990958   998199
## 48          Rhode Island  1052567  1052567  1052528  1051302
## 38         New Hampshire  1316470  1316472  1316807  1318194
## 28                 Maine  1328361  1328361  1327379  1328188
## 20                Hawaii  1360301  1360301  1363359  1374810
## 21                 Idaho  1567582  1567582  1571102  1584985
## 36              Nebraska  1826341  1826341  1830141  1842641
## 57         West Virginia  1852994  1852996  1854368  1855364
## 40            New Mexico  2059179  2059180  2065913  2082224
## 37                Nevada  2700551  2700551  2704283  2723322
## 53                  Utah  2763885  2763885  2775479  2817222
## 25                Kansas  2853118  2853118  2859143  2871238
## 12              Arkansas  2915918  2915921  2921588  2937979
## 33           Mississippi  2967297  2967297  2970072  2978512
## 24                  Iowa  3046355  3046350  3050202  3062309
## 15           Connecticut  3574097  3574097  3575498  3580709
## 45              Oklahoma  3751351  3751354  3760184  3791508
## 46                Oregon  3831074  3831074  3838332  3871859
## 26              Kentucky  4339367  4339362  4347223  4369356
## 27             Louisiana  4533372  4533372  4545343  4574836
## 49        South Carolina  4625364  4625364  4637106  4679230
```

```
## 9               Alabama  4779736  4779735  4785401  4802740
## 14             Colorado  5029196  5029196  5047692  5116796
## 32            Minnesota  5303925  5303925  5310658  5344861
## 58            Wisconsin  5686986  5686986  5691659  5711767
## 29             Maryland  5773552  5773552  5785681  5828289
## 34             Missouri  5988927  5988927  5995715  6010688
## 51            Tennessee  6346105  6346110  6357436  6403353
## 11              Arizona  6392017  6392013  6413158  6482505
## 23              Indiana  6483802  6483800  6490622  6516922
## 30        Massachusetts  6547629  6547629  6555466  6587536
## 56           Washington  6724540  6724540  6742950  6830038
## 55             Virginia  8001024  8001030  8023953  8096604
## 39           New Jersey  8791894  8791894  8799593  8821155
## 42       North Carolina  9535483  9535475  9560234  9656401
## 19              Georgia  9687653  9687660  9712157  9815210
## 31             Michigan  9883640  9883635  9877143  9876187
## 44                 Ohio 11536504 11536502 11537968 11544951
## 47         Pennsylvania 12702379 12702379 12717722 12742886
## 22             Illinois 12830632 12830632 12841980 12869257
## 18              Florida 18801310 18801311 18838613 19057542
## 41             New York 19378102 19378104 19395206 19465197
## 52                Texas 25145561 25145561 25253466 25674681
## 13           California 37253956 37253956 37338198 37691912
```

Step 5: Explore the distribution of the states

10. Write a function that takes two parameters. The first is a vector and the second is a number.

11. The function will return the percentage of the elements within the vector that is less than the same (i.e. the cumulative distribution below the value provided).

12. For example, if the vector had 5 elements (1,2,3,4,5), with 2 being the number passed into the function, the function would return 0.2 (since 20% of the numbers were below 2).

13. Test the function with the vector 'dfStates$Jul2011Num', and the mean of dfStates$Jul2011Num'

```
# 10 through 13 answers will be included within this code block.
testVector <- c(1,2,3,4,5)

ExploreDist <- function(vec, i){
  return(sum(vec < i)/length(vec))


}

#Test The Function as stated
ExploreDist(testVector, 2)
```

```
## [1] 0.2
```

```
#Testing with dfStates

ExploreDist(dfStates$Jul2011, mean(dfStates$Jul2011))
```

```
## [1] 0.6666667
```

There are many ways to write this function (described in #10 above) – so please try to write multiple versions of this function – which do you think is best?

```
ExploreDistOtherWay <- function(vec, i){
boolList <-vec < i
trueCount <- length(boolList[boolList==TRUE])
return(trueCount/length(vec))
}

#Test Alternative
ExploreDistOtherWay(testVector,2)
```

```
## [1] 0.2
```

```
ExploreDistOtherWay(dfStates$Jul2011, mean(dfStates$Jul2011))
```

```
## [1] 0.6666667
```

There are many ways to approach writing this function, however two important aspects come to mind. First, readability. When writing a function or any code, the author will want to construct it in such a way that it is easy for others to quickly read and understand. Second is efficency. A user could write a function which creates additional vectors and variables, however this will decrease efficiency and increase memory usage. In this assignment the datasets are small and memory utilization is not much of a concern, however within an production/comercial enviroment it has the potential quickly or immediately become an issue.